Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol III,
IMECS 2010, March 17 - 19, 2010, Hong Kong

# Evaluation of Simulation Strategy on Single-Player Monte-Carlo Tree Search and its Discussion for a Practical Scheduling Problem

Shimpei Matsumoto*, Noriaki Hirosue†, Kyohei Itonaga‡,
Kazuma Yokoo‡ and Hisatomo Futahashi‡

*Abstract*—**Monte-Carlo Tree Search (MCTS) is a best-first search where the pseudorandom simulations guide the solution of problem. Recent improvements on MCTS have produced strong computer Go program, which has a large search space, and the success is a hot topic for selecting the best move. So far, most of reports about MCTS have been on two-player game, and MCTS has been applied rarely in one-player games. MCTS does not need an admissible heuristic, so the application of MCTS for one-player games might be an interesting alternative. Additionally, one-player games changed its situation by player's decision like puzzles are describable as network diagrams like PERT with the representation of interdependences between each operation. Therefore if MCTS for one-player games is developed as a meta-heuristic algorithm, we would use this for not only many practical problems, but also combinatorial optimization problems. This paper investigated the application of Single Player MCTS (SP-MCTS) introduced by Schadd et al. to a puzzle game called Bubble Breaker. Next this paper showed the effectiveness of new simulation strategies on SP-MCTS by numerical experiments, and found the differences between the search methods and their parameters. Based on the results, this paper discussed the application potentiality of SP-MCTS for a practical scheduling problem.**

***Keywords:*** ***one-player game, bubble breaker, monte-carlo tree search, scheduling problem***

## 1 Introduction

Games are separated into several classes mathematically according to the characteristics such as the number of players, completeness of information, uncertainty, and the total of benefits (zero/nonzero-sum) [2]. Among them, perfect-information games do not have accidental factor, so it is theoretically possible to foresee the moves to the end. However, the computational effort actually explodes with following the situations of game, so it is virtually impossible to calculate optimal moves for most of games. To solve this problem, positional evaluation functions are given to decide the moves as criterion to determine the search range and to measure the situations. As one of the typical search algorithm using an evaluation function to foresee the opponent decision, Mini-Max method is a powerful tool for the design of game algorithm. Many algorithm refined Mini-Max method such as $\alpha\beta$ method have been proposed. However giving a proper evaluation function is extremely difficult for problems with large search space.

Recently in computer Go program, a revolutionary search algorithm without an evaluation function, Monte-Carlo Tree Search (MCTS) was proposed by Coulom [5]. MCTS performs many play-outs (playing until a simulation encounters the end of the game) with pseudorandom moves, and each situation is evaluated based on the results [1]. Moves are selected in self-play until the end of the game. It is well-known that the use of an adequate simulation strategy improves the level of play significantly [6, 4], and the main idea is to use heuristics. To control simulations, UCT (Upper bound Confidence for Tree) is mainly used, which is one of the most effective algorithm and is based on UCB1 (Upper Confidence Bounds) for multi-armed bandit problem [7]. Schadd et al. proposed a new MCTS variant, called Single-Player Monte-Carlo Tree-Search (SP-MCTS) for SameGame, a puzzle game [3]. They mentioned A* and IDA* to compare the performance of SP-MCTS, and SP-MCTS obtained the highest score than the others for benchmark problems, so it turned out that SP-MCTS is able to be considered as a new method for NP-complete puzzles. Basing on the results, SP-MCTS is thought to be a meta-heuristic algorithm, so it might be applied for not only perfect/imperfect information one-player games but also other problems, as long as search spaces of the problems are describable as tree structure, and the problems have

*Shimpei Matsumoto is with the Department of Computer and Control Engineering, Oita National College of Technology, 1666 Oaza-Maki, Oita City, Oita 870-0152 Japan; Tel/Fax: +81-97-552-7421; Email: smatsu@oita-ct.ac.jp

†Noriaki Hirosue is with the Department of Artificial Intelligence, Kyushu Institute of Technology, Kawazu 680-4, Iizuka 820-8502, Japan; Email: i231209n@ai.kyutech.ac.jp

‡Kyohei Itonaga, Kazuma Yokoo, and Hisatomo Futahashi are with the Department of Computer and Control Engineering, Oita National College of Technology, 1666 Oaza-Maki, Oita City, Oita 870-0152 Japan; Email: {s0506, s0540, s0528}@cc.oita-ct.ac.jp

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

termination of operations.

The purpose of this paper is to evaluate SP-MCTS for one-player perfect-information games, Bubble Breaker given similar rules with SameGame, and to examine application potentiality of the solution as a meta-heuristic algorithm. The one-player perfect-information game is so-called puzzle games, and most of them give closely-defined rules. Previous researches have shown that most of puzzle games are equal to optimization problems belonging to the class of NP-Complete, which are difficult to find an optimal solution, and puzzle games are able to be considered as typical combinatorial optimization problems. Therefore if we can clarify the effectiveness of SP-MCTS for such problems, we will open up new possibilities of SP-MCTS for many practical problems that is able to be described as combinatorial optimization problems. This paper firstly applies SP-MCTS for Bubble Breaker already reported NP-Completeness [9], and evaluates its effectiveness. This paper develops a softawre of Bubble Breaker with Action Script 3.0 to present the sequence of moves visually until a game is solved. This paper develops SP-MCTS based on concept of Schadd et al., and proposes new heuristics. This paper shows the results of numerical experiments, and discusses the characteristic of SP-MCTS. Finally based on the efforts of this paper, this paper examines the availability of SP-MCTS for practical problems by generalizing the procedures of SP-MCTS for one-player games. This paper mentions a reentrant scheduling problem as an example of practical problem [8].

## 2 Problem

There is a rectangular playing screen initially filled with several, typically 4 or 5, kinds of blocks (colors) at random. By selecting one of a group of adjoined blocks, a player may remove them from the screen. A move consists of removing a group of (at least two) orthogonally adjacent blocks with the same color. The blocks on top of the removed group will fall down, and a column without any blocks will be trimmed away by other columns sliding to the left. Usually, there will be no time constraints in the game, however some implementations gradually push the rows upward or drop blocks from above. The game is over if no more blocks can be removed.

For each removed group points are rewarded. As the most versions of the game, the amount of points is dependent on the number of blocks removed and is given by the formula $(n-k)^2$ for removing $n$ tiles (the size of the removed group) at once, where $k = 1$ or 2 depending on the implementation. Some versions also offer a large bonus for removing all the blocks on the screen. Yet others reduce the final score based on the number of blocks remaining at the end of the game. The game is over if either the player has removed all blocks, or is left with a position where no adjacent blocks have the same color. In the first case, some bonus points are usually rewarded, and in the second case, points are deducted. The formula for deduction is similar to the formula for rewarding points, and applied for blocks left on the board. During deduction it is assumed that all blocks of the same color are connected.

Bubble Breaker, the problem in this paper, adopts the following scoring function, which the score $S_k$ of $k$th turns is given as

$$S_k = (n-2)^2 . \tag{1}$$

The final score is given as follows.

$$T_s = \begin{cases} \sum_{i=1}^{k_{max}} S_k + 1000 & ; \text{ if } \quad R = 0, \\ \sum_{i=1}^{k_{max}} S_k - (R-2)^2 & ; \text{ otherwise.} \end{cases} \tag{2}$$

where $R$ is the number of blocks left on the board, and $k_{max}$ is the number of turns at the termination of the problem.

In SameGame given the similar rule with Bubble Breaker, the number of leaf notes for a random initial position is estimated to be approximately $10^{85}$ in average, and the total number of possible states is approximately $10^{159}$, which is calculated by the number of combinations for columns, $C^k = \sum_{n=0}^{r} c^n$ where $r$ is the height of the column, $c$ is the number of colors, and $k$ is the number of columns. As the similar problems, Clickomania with 5 colors and 2 Columns was proven to be NP-complete by [9] where no points are rewarded and the only objective is to clear the board. In SameGame, a variant of Clickomania, there are two terminal positions with no blocks or some blocks on the board. SameGame is proven as a harder problem than Clickomania by Schadd et al. and shown as NP-complete, too [3]. The problem in this paper, Bubble Breaker adopts the same rule of SameGame except scoring formula, so we can understand that Bubble Breaker is also NP-complete.

## 3 Algorithm

### 3.1 Concept

In MCTS, one new node on search tree is expanded per iteration. Usually the following four steps are repeated until the time runs out, but this paper iterates the steps until the number of expanded nodes reaches the predetermined threshold. One-player games do not have the opponent unlike two-player games, so we do not have to worry about the opponent's decision; for example, it is not needed to wait for an unknown reply of an opponent. Therefore MCTS can perform the search with wide space from the initial position at once. Eventually, nodes of the tree from the root to the leaf (end of the game) with the

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

highest score are obtained as solution procedures. Details of each procedure are shown as follows.

**Selection Strategy:** Selection is the strategic task that selects one node on the search tree. It controls the balance between exploitation and exploration [7]. This procedure selects one child node from root node recursively. The search tree has only the root node at first simulation, so the root node is definitely selected. After several simulations, some child nodes are generated on the search tree. The node with some child nodes is defined as parent node, and this paper selects one node with the highest value of UCT including the parent node. Specifically, UCT on the root node and its child nodes are firstly compared, then as the result, one node with the highest UCT is selected. If the root node as parent node has the highest UCT, it is selected even the parent node, otherwise the child node is selected. This operation is repeated until the leaf node appears. It is important that the parent node is always assumed as leaf node. This paper gives following exceptions.

- When a node has no pattern to expand new nodes, the node is removed from consideration even though the node has the highest UCT.
- This strategy does not surely select the node with child nodes presenting all patterns to the end of the problem because simulations will never get new patterns.

This paper uses a modified UCT version for SP-MCTS introduced by Schadd et al [3]. At the selection of parent node $N$ with child nodes $N_i$, the selection strategy chooses the node to move with the value of UCT $U(N_i)$, which maximizes the following formula.

$$\begin{aligned} U(N_i) &= \bar{X}_{N_i} + C \cdot \sqrt{\frac{\ln t(N)}{t(N_i)}} \\ &+ D \cdot \sqrt{\frac{\sum_{k=1}^{t(N_i)} x(N_i)_k^2 - t(N_i)\bar{X}_{N_i}^2}{t(N_i)}}, \quad (3) \end{aligned}$$

where $C$ and $D$ are constants, $t(N)$ is the number of times that the parent node $N$ was visited, $t(N_i)$ is the number of times that child node $N_i$ was visited, and $\bar{X}_{N_i}$ is the average score of node $N_i$ to give an upper confidence bound. The first two terms constitute the original UCT formula. The third term is for one-player game, $x(N_i)_k^2$ is the score obtained by simulation $k$ of node $N_i$ . The third term is basing on standard deviation, therefore both the benefit and the risk increases when the value of third term is large.

**Simulation:** Starting from a leaf node in the search tree developed, a pseudorandom simulation with some heuristics is performed until the end of the problem. Each leaf node memorizes the state of problem, and the simulation regards the state of the leaf node as initial input. The search tree has only root node when a experiment starts, so the root node is necessarily selected as leaf note. The root node memorizes the state of problem unprocessed, complete initial input. The selection of leaf node depends on the selection strategy described below steps. In order to improve the quality of solution, some heuristics based on knowledge are usually applied for simulations. For SameGame, a game with similar rules in this paper's problem, Schadd proposed two static simulation strategies, Tabu Random, and Tabu Color Random [3] that are described as solution 2 and 3 in the section of numerical experiments. All selected nodes by a play-out and the obtained score are memorized each simulation, so the best score and moves (procedures to remove blocks) are updated when the simulation exceeds the previous best score, which are finally obtained as a solution.

**Expansion:** The expansion procedure decides which nodes are added to the tree. This paper applies Coulom's method that expands one child per simulation [**?**]. The new child node is the first encountered position of each simulation that was not present in the tree.

**Backpropagetion:** The backpropagation procedure updates UCT for simulated nodes at the leaf to the root including expanded node. All selected nodes from the root to the leaf node are memorized on the selection strategy when one simulation is performed, so the result of simulation is propagated backward to the root rode from the expanded child node of selected leaf node. This procedure updates the average score $\bar{X}_i$ of node $i$ used by simulation, and adds the obtained score $x_m$ to the total $\sum_{k=1}^{m} x_k$, where $m$ is the number of simulations performed. each node's UCT is propagated to the node's ancestors up to the root.

### 3.2 Simulation Knowledge

In the simulation step, moves are randomly or pseudo-randomly decided, and heuristics generate pseudorandom moves. The random simulation is described as Solution 1 in the section of Numerical Experiment, and as pseudo-random methods, this paper mentions Tabu Random and Tabu Color Random proposed by Schadd et al., which are described as Solution 2 and Solution 3 in this paper respectively. Bubble Breaker gives greater score according to the number of blocks to remove that can be understood from the scoring formula $(n-2)^2$, so making large groups of blocks is advantageous. Both heuristics aim at making large groups of one color. Tabu Random decides a color at the start of a simulation randomly. During the random simulations, it is not allowed to remove blocks with this color unless there are no other moves possible. With this strategy, large groups of the chosen color will be formed automatically. Tabu Color Random chooses a color with the most frequently occurring at the start of the simulation, and performs similar operation with Tabu Random. This may increase the probability of hav-

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

ing large groups during the random simulation. This paper designs 4 heuristics based on the usual methods as follows.

**Modified Tabu Color Random:** At first, moves are decided with Tabu Color Random. Then Random method is applied when the number of blocks falls below a criterion for all colors. This method is based on the assumption that the possibility of removing all the blocks on the screen might decrease with only Tabu Color Random. This method is described as Solution 4.

**Probabilistic Tabu Color Random:** The percentage of the number of blocks with every color is calculated, and each color is given its place accordingly to its percentage, like on the roulette wheel. This method decides a color to choose by giving real value (0-100) randomly, and performs similar operation with Tabu Random. This method is described as Solution 5.

**Probabilistic Weighted Tabu Color Random:** This method is modified based on the solution 5, and is described as Solution 6. Color with high rate is susceptible to select, and every color's rate is given by the percentage of squared value of the number of blocks.

**Dynamic Probabilistic Weighted Tabu Color Random:** This method is modified based on the solution 6, and is described as Solution 7. Solution 6 does not change the selected color until blocks with the color cannot be removed, but the Solution 7 chooses color to remove in each turn based on the rate determined by the method in Solution 6.

## 4 Numerical Experiments

### 4.1 Performance Evaluation of Solutions

This paper conducted numerical experiments to compare the performance of simulation strategies based on the usual methods (solution 1, 2 and 3) with the proposal methods (solution 4, 5, 6 and 7). Experimental circumstance is Windows XP Pro. SP3, AMD Athlon 64 Processor 3500+ 2.20GHz, 3.00GB RAM, and programs are developed by JDK 6.0. 20 benchmark problems previously mentioned were repeatedly-solved 1000 times, and differences of performance depending on the number of nodes are examined. This paper verifies the performance of solutions with 6 sizes of nodes $10^0, 10^1, \cdots, 10^5$, which the maximum number of nodes $10^5$ was given due to the performance of computer. Concerning constants $(C; D)$ in UCT, the previous work examined 3 different settings in order to investigate which balance between exploitation and exploration gives the best results [3]. Schadd et al. represented the parameter pair $(0.1; 32)$ as exploitation, $(1; 20,000)$ as exploration, and $(0.5; 10,000)$ as balance in [3], and based on the simulation results, they showed that the pair $(0.1; 32)$ obtained the most efficient value when the number of nodes is $10^5$. Therefore this paper adopted the pair $(0.1; 32)$.

Experimental results of one benchmark problem are shown in Fig.1 because the other problems showed similar results. Fig.1 shows the histogram of solutions, which the vertical axis is frequency of solutions, and the horizontal axis is score obtained. To clarify the differences, vertical dotted line is described at 0 point in each graph. The mode of score was better with increasing of the number of nodes, so we can understand that the number of nodes improves the performance of solutions. Fig.1(a) is the result when the solutions is given $10^0 = 1$ node, and it denotes that only one simulation is executed. Therefore the graph (a) shows characteristic of each solution (heuristic) not depending on UCT. Solutions with some heuristics (solution 2 to 7) obtained better score than solution 1 randomly-selecting brocks, so the graph (a) shows the importance of heuristics. Especially we can evaluate the performance of each heuristic around 500 points, and in the case of solution with $10^0$ node, solution 4 was the most effective heuristic on average. Most of simulation obtained negative value for all solutions, so the result shows that many indelible blocks left on the board might incur the large deduction of score as penalty.

From the results of $10^1 - 10^5$ nodes shown in Fig.1(b)-(f), the effects of UCT on simulation strategy were shown. We can see the affection of bonus or deduction for the results of score from the graph (b), for example, solution 1 showed a bimodal distribution, but the other solutions showed monomodal distribution. Solution 1 with random selection only showed highly variable in score, so appropriate heuristic might ensure stable behavior. The graph (b) shows the differences of mode value depending on the accuracy of heuristics, which the solutions 2 and 5-7 had the mode value at 400 points, but the solutions 3 and 4 had the mode value at 800 points. Both solutions 3 and 4 with higher score were based on the heuristic focusing on the color with the greatest number of blocks on the board, so this strategy can be said to be effective. In the case of $10^2 - 10^5$ nodes shown in graph (c)-(f), all solutions had positive score but solutions 1 and 7 had lower mode value than the other solutions, which the both solutions changes color to remove every turn on simulation. This result denotes that the heuristic based on solutions 1 and 7 might not be efficient for this problem. The graphs (d)-(f) shows bimodal distribution for solution 1 and 7, but the other solutions had monomodal distribution. This result can be thought that solutions 2-6 were not given large deduction, and especially almost all simulations with solutions 2-6 might get bonus points for removing all blocks in the graphs (e) and (f) because they obtained points much above 1000 points. This paper also found from the graph (d) that solution with $10^3$ nodes or above can remove all blocks even with random selection (without knowledge). The attainment of bonus points

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

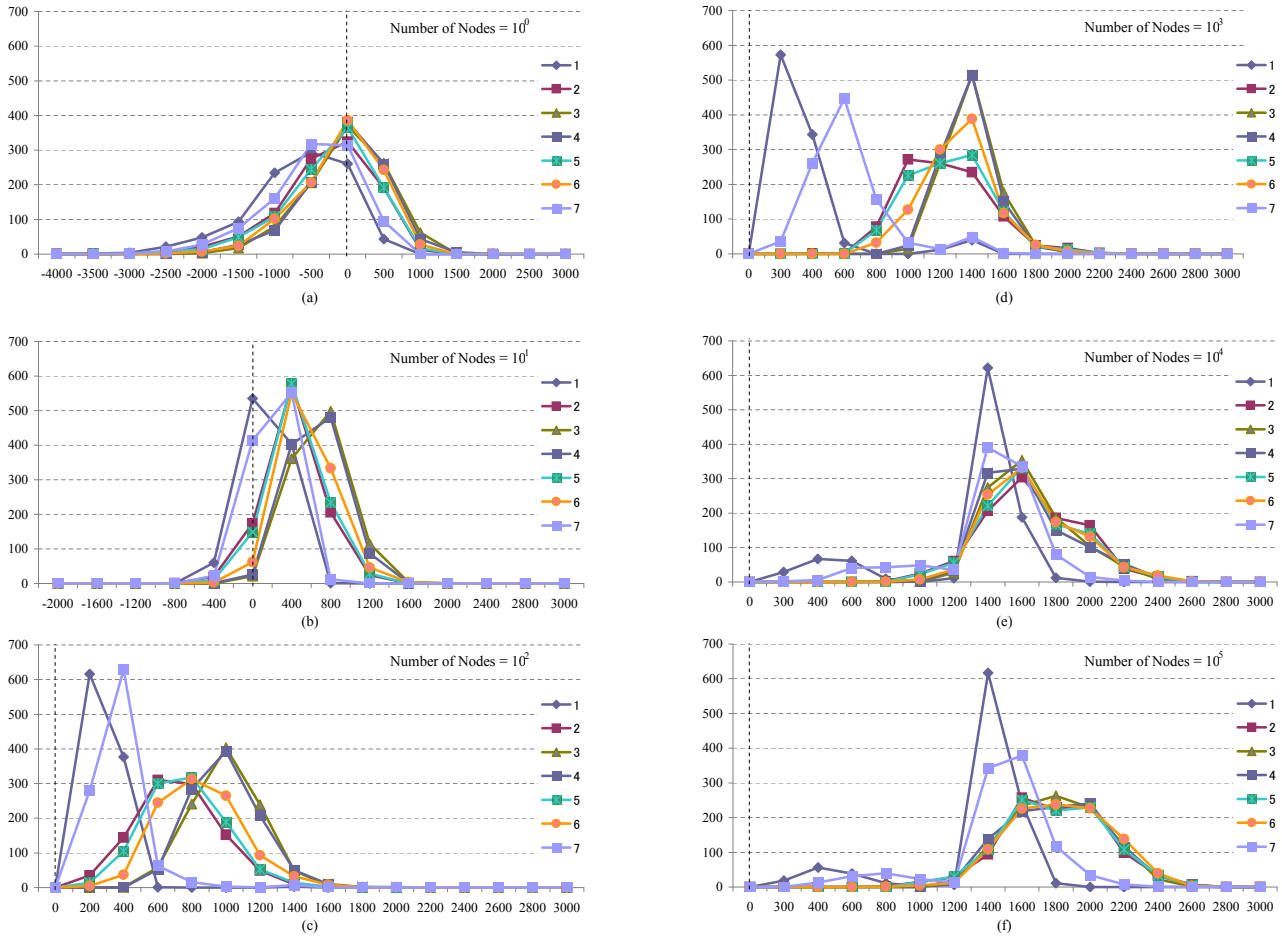


Figure 1: Simulation Results (Variance of Scores)

(removing all blocks) is apparent that the frequency of results with solution 1 was near zero around 800-1000 points. In the case of $10^3$ nodes shown in the graph (d), solutions 3 and 4 only showed remarkable results, on the other hand, in the case of $10^4$ nodes or above shown in the graphs (e) and (f), there were little differences between efficient heuristics (solutions 2-6) even thought solutions with $10^4$ nodes obtained higher scores than $10^3$ nodes. We can make the most of the effectiveness of knowledge between $10^3$ and $10^4$ nodes because the solutions with $10^4$ nodes or above require a long computational time. Interestingly, in the case of $10^4$ nodes, solutions 2, 5 and 6 had the better score around 2000 points than solutions 3 and 4 which is the most efficient solutions for the other nodes. Solutions 3 and 4 definitely keep the color of blocks with the greatest numbers intact, but solutions 2, 5 and 6 are not exactly, which select blocks based on the probabilistic operation. Increasing of the number of nodes might lead to solutions to the discovery of unexpected knowledge, and new heuristics will be obtained by recording solution procedures. There was no great difference of scores between efficient solutions when given $10^5$ nodes shown in the graph (f) except solutions with inefficient heuristics. The difference in outcome shown in the graph (f) is thought to be within the margin of error.

After reviewing the results obtained by simulations shown in the graphs (a)-(f), it was found that the average scores of solutions will definitely get higher with increasing of the number of nodes. The number of simulations and the number of nodes are equal, so the increase of nodes will inevitably enhance the possibility to get an high score, and in addition, more efficient nodes might be selected by the selection strategy based on UCT. Second, it was proved to be possible to obtain an extremely effective score (in this problem, the effective score was from bonus points) without knowledge when a solution with $10^3$ nodes or above was applied. As further interest, the graph (d) showed the point of maximum frequency in the left distribution, but the graph (e) showed it in the right distribution about solution 1. It denotes that the number of nodes might be a quantitative criterion for the difficulty of problem.

## 4.2 Computational Time

To show the differences of computational time for solutions, the simulation time of all trials was measured as shown in Fig.2, as the double logarithmic chart, which

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

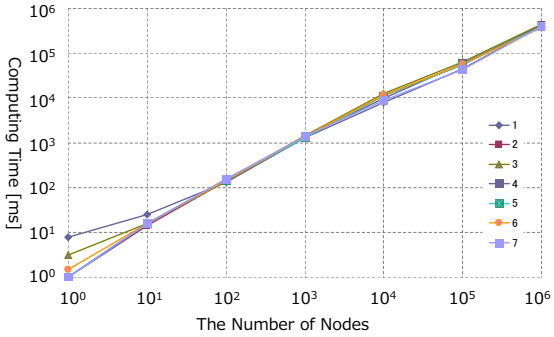


Figure 2: Comparison of computational time

the vertical axis is computational time [ms], and the horizontal axis is the number of nodes. Fig.2 plotted the average time of all solutions for 1000 trials. All solutions increased approximately linearly, and showed similar changes. The remarkable difference between each solution was confirmed at $10^0$ node, but the difference was progressively narrower with increasing of nodes. Particularly the computational time of all solutions at $10^6$ nodes was almost same. Based on the results, variance at $10^0$ node is assumed to be due to the operations of initialization. The interesting thing Fig.2 showing is that the difference of heuristics did not make a significant influence for the computational time, so to implement some knowledge might be considered to be effective.

## 5 Discussions and Conclusions

The experiments in this paper showed the effectiveness of heuristics to improve the performance of solution, and the heuristics had little impact on computational time. Solutions with $10^4$ nodes or above needed a lot of computational time and reduced the efficacy of heuristics because perfectly-random method could obtain efficient solutions, so we concluded that it is sufficient to give $10^4$ nodes for Bubble Breaker. From the results, it is thought that exact number of nodes might depend on the characteristic of problems. This paper also considers that the combination with a pruning algorithm like beam search will be important to obtain more efficient results quickly.

This paper considers that SP-MCTS might be a good match with practical scheduling problems, especially a reentrant scheduling problem [8]. We have been focused on the improvement of a printing process as a practical scheduling. In the printing process, dial plates used for car tachometers are printed with various colors and character plates. At this time, the production lead time can be shortened by collecting the products printed by the same type of color or character plate. When the type of color or character plate is switched to another type, the process requires "setup operation" with production idle time. So the problem can be formulated by the minimization of setup operations. In our previous efforts, the printing process had been improved by only theoretical discussion, however it could not flexibly respond to the change of production circumstance in spite of the actual field can handle these. As this reason, we assumed that our earlier heuristic solution method had not considered the expert's technical knowledge. Therefore we regarded that the tacit knowledge in the actual printing field is absolutely necessary corresponding to the change of production condition, and then the problem and its solution were modeled on the basis of expert's technique. We described a rule-based solution that can obtain a result quickly and it is developable by workers themselves. We think that the motivation using SP-MCTS is not only to get effective results. SP-MCTS might be of value in observation of processes of obtaining an efficient result from probabilistic simulations. The observation will support to find a new knowledge, and then ambiguities of solution will be closely described by adding a new knowledge.

## References

[1] B. Brügmann, "Monte Carlo Go, Technical report," *Physics Department*, Syracuse University, 1993.

[2] M. Sakuta, "Studies on Imperfect-Information Games," *Operations Research*, Vol.52, No.1, pp.27-34, 2007 (In Japanese).

[3] M. Schadd, M. Winands, H. van den Herik and H. Aldewereld, "Addressing NP-Complete Puzzles with Monte-Carlo Methods." *Proc. of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning*, Vol. 9, pp.55-61, 2008.

[4] M. Winands, Y. Bjornsson and J. Saito, "Monte-Carlo Tree Search Solver," *Computers and Games*, Vol.5131, pp.25-36, 2008.

[5] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," *Computers and Games*, Vol.4630, pp.72-83, 2007.

[6] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," *Proc. of the International Conference on Machine Learning*, No.227, pp.273-280, 2007.

[7] S. Gelly, Y. Wang, R. Munos and O. Teytaud, "Modification of uct with patterns in monte-carlo go," *Technical Report RR-6062*, INRIA, 2006.

[8] S. Matsumoto et al., "Business Process Analysis to Obtain Empirical Lot Sizing Rule in Printing Process", *Proc. of 2008 IEEE Conference on Automation Science and Engineering*, pp.591-596, 2008.

[9] T. Biedl, E. Demaine, M. Demaine, R. Fleischer, L. Jacobsen and J Munro, "The Complexity of Clickomania", *Proc. of MSRI Workshop on Combinatorial Games*, pp.389-404, 2002.

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)