Addressing NP-Complete Puzzles with Monte-Carlo Methods[1]

Maarten P.D. Schadd and **Mark H.M. Winands**
H. Jaap van den Herik and **Huib Aldewereld**[2]

Abstract.

NP-complete problems are a challenging task for researchers, who investigate tractable versions and attempt to generalise the methods used for solving them. Over the years a large set of successful standard methods have been developed. We mention A* and IDA* which have proven to be reasonably successful in solving a set of NP-complete problems, particularly single-agent games (puzzles). However, sometimes these methods do not work well. The intriguing question then is whether there are new methods that will help us out.

In this paper we investigate whether Monte-Carlo Tree-Search (MCTS) is an interesting alternative. We propose a new MCTS variant, called Single-Player Monte-Carlo Tree-Search (SP-MCTS). Our domain of research is the puzzle SameGame. It turned out that our SP-MCTS program gained the highest scores so far on the standardised test set. So, SP-MCTS can be considered as a new method for addressing NP-complete puzzles successfully.

1 INTRODUCTION

Creating and improving solvers for tractable versions of NP-complete problems is a challenging task in the field of Artificial Intelligence research. As Cook [9] proved: all problems in the class of NP-complete problems are translatable to one another [16]. This implies that a solution procedure for one problem also holds for other problems. Otherwise stated: if an effective method is found to solve a particular instance of a problem, many other problems may be solved as well using the same method.

Games are often NP-complete problems. The rules for games are well-defined and it is easy to compare different methods. For our investigations we have chosen a one-person perfect-information game (a puzzle[3]) called $SameGame$. In Section 2 we will prove that this puzzle is NP-complete.

The traditional methods for solving puzzles, such as the 15×15 puzzle and Sokoban, are A* [15] or IDA* [19]. Other problems, such as the Travelling Salesman Problem (TSP) [3] require different methods (e.g., Simulated Annealing [12] or Neural Networks [23]). These methods have been shown to solve the puzzles mentioned above reasonably well. An example of a practical and successful use of these methods are pathfinders which are, for example, used inside an increasing number of cars. A drawback of the methods is that they need an admissible heuristic evaluation function. The construction of such a function may be difficult.

An alternative to these methods can be found in Monte-Carlo Tree Search (MCTS) [7, 10, 18] because it does not need an admissible heuristic. Especially in the game of Go, which has a large search space [5], MCTS methods have proven to be successful [7, 10]. In this paper we will investigate how MCTS addresses NP-complete puzzles. For this purpose, we introduce a new MCTS variant called SP-MCTS.

The course of the paper is as follows. In Section 2 we present the background and rules of SameGame. Also, we prove that SameGame is NP-complete. In Section 3 we discuss why classical methods are not suitable for SameGame. Then we introduce our SP-MCTS approach in Section 4. Experiments and results are given in Section 5. Section 6 shows our conclusions and indicates future research.

2 SAMEGAME

We start by presenting some background information on SameGame in Subsection 2.1. Subsequently we explain the rules in Subsection 2.2. Finally, we prove that SameGame is NP-complete in Subsection 2.3.

2.1 Background

SameGame is a puzzle invented by Kuniaki Moribe under the name *Chain Shot!* in 1985. It was distributed for Fujitsu FM-8/7 series in a monthly personal computer magazine called *Gekkan ASCII* [20]. The puzzle was afterwards re-created by Eiji Fukumoto under the name of *SameGame* in 1992. So far, the best program for SameGame has been developed by Billings [24].

2.2 Rules

SameGame is played on a rectangular vertically-placed 15×15 board initially filled with blocks of 5 colours at random. A move consists of removing a group of (at least two) orthogonally adjacent blocks of the same colour. The blocks on top of the removed group will fall down. As soon as empty columns occur, the columns to the right are shifted to the left. For each removed group points are rewarded. The amount of points is dependent on the number of blocks removed and can be computed by the formula $(n-2)^2$, where n is the size of the removed group.

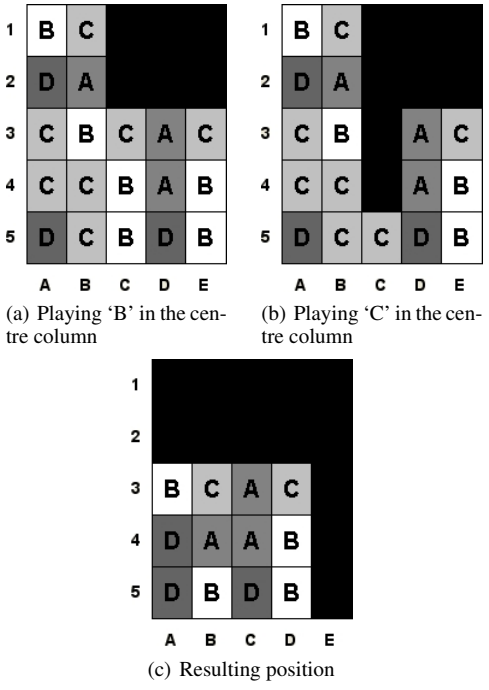
We show two example moves in Figure 1. When the 'B' group in the third column of position 1(a) is played, it will be removed from the game and the 'C' block on top will fall down, resulting in position

[1] This contribution is a revised version of an article under submission to CG 2008.

[2] Maastricht University, Maastricht, The Netherlands, email: {maarten.schadd, m.winands, herik, h.aldewereld}@micc.unimaas.nl

[3] Although arbitrary, we will call these one-player games with perfect information for the sake of brevity *puzzles*.

1(b). Because of this move, it is now possible to remove a large group of 'C' blocks (n=5). Owing to an empty column the two columns at the right side of the board are shifted to the left, resulting in position 1(c).[4] The first move is worth 0 points; the second move is worth 9 points.



(a) Playing 'B' in the centre column

(b) Playing 'C' in the centre column

(c) Resulting position

Figure 1. Example SameGame moves.

The game is over if either the player (1) has removed all blocks or (2) is left with a position where no adjacent blocks have the same colour. In the first case, 1,000 bonus points are rewarded. In the second case, points will be deducted. The formula for deduction is similar to the formula for rewarding points but is now iteratively applied for each colour left on the board. During deduction it is assumed that all blocks of the same colour are connected.

There are variations that differ in board size and the number of colours, but the 15×15 variant with 5 colours is the accepted standard. If a variant differs in scoring function, it is named differently (e.g., Jawbreaker, Clickomania) [2, 21].

2.3 Complexity of SameGame

The complexity of a game indicates a measure of hardness of solving the game. Two important measurements for the complexity of a game are the game-tree complexity and the state-space complexity [1]. The game-tree complexity is an estimation of the number of leaf nodes that the complete search tree would contain to solve the initial position. The state-space complexity indicates the total number of possible states.

For SameGame these complexities are as follows. The game-tree complexity can be approximated by simulation. For SameGame, the game-tree complexity for a random initial position is 10^{85} in average. The state-space complexity is computed rather straightforwardly. It is possible to calculate the number of combinations for one column by $C = \sum_{n=0}^{r} c^n$ where r is the height of the column and c is the number of colours. To compute the state-space complexity we take C^k where k is the number of columns. For SameGame we have 10^{159} states. This is not the exact number because a small percentage of the positions are symmetrical.

Furthermore, the hardness of a game can be described by deciding to which complexity class it belongs [16]. The similar game Clickomania was proven to be NP-complete by [2]. However, the complexity of SameGame can be different. The more points are rewarded for removing large groups, the more the characteristics of the game differ from Clickomania. In Clickomania the only goal is to remove as many blocks as possible, whereas in SameGame points are rewarded for removing large groups as well. In the following, we prove that SameGame independently from its evaluation function belongs to the class of NP-complete problems, such as the 3-SAT problem [9].

Theorem 1 *SameGame is NP-complete*

For a proof that it is NP-complete, it is sufficient to reduce SameGame to a simpler problem. We reduce SameGame to Clickomania, which has been proven to be NP-complete with 5 colours and 2 Columns [2]. A SameGame instance with 2 columns is easier to solve than the standard SameGame instance with 15 columns. Instead of proving that finding the optimal path is NP-complete, we prove that checking whether a solution s is optimal is already NP-complete. A solution is a path from the initial position to a terminal position. Either s (1) has removed all blocks from the game or (2) has finished with blocks remaining on the board. Even in the second case a search has to be performed to investigate whether a solution exists that clears the board and improves the score. If we prove that searching all solutions which clear the board is NP-complete, then SameGame is NP-complete as well.

Clickomania is a variant of SameGame where no points are rewarded and the only objective is to clear the board. Finding one solution to this problem is easier than finding every solution. Therefore, it is proven that SameGame is a harder problem than Clickomania; SameGame is NP-complete, too.

3 CLASSICAL METHODS: A* AND IDA*

The classical approach to puzzles involves methods such as A* [15] and IDA* [19]. A* is a best-first search where all nodes have to be stored in a list. The list is sorted by an admissible evaluation function. At each iteration the first element is removed from the list and its children are added to the sorted list. This process is continued until the goal state arrives at the start of the list.

IDA* is an iterative deepening variant of A* search. It uses a depth-first approach in such a way that there is no need to store the complete tree in memory. The search will continue depth-first until the cost of arriving at a leaf node and the value of the evaluation function pass a certain threshold. When the search returns without a result, the threshold is increased.

Both methods are heavily dependent on the quality of the evaluation function. Even if the function is an admissible under-estimator, it still has to give an accurate estimation. Well-known puzzles where this approach works well are the Eight Puzzle with its larger relatives [19, 22] and Sokoban [17]. Here a good under-estimator is the well-known Manhattan Distance. The main task in this field of research is to improve the evaluation function, e.g., with pattern databases [11, 13].

[4] Shifting the columns at the left side to the right would not have made a difference in points. For consistency, we will always shift columns to the left.

These classical methods fail for SameGame because it is not easy to make an admissible under-estimator that still gives an accurate estimation. An attempt to make such an evaluation function is by just rewarding points to the groups on the board without actually playing a move. However, if an optimal solution to a SameGame problem has to be found, we may argue that an "over-estimator" of the position is needed. An admissible over-estimator can be created by assuming that all blocks of the same colour are connected and would be able to be removed at once. This function can be improved by checking whether there is a colour with only one block remaining on the board. If this is the case, the 1,000 bonus points at the end can be deducted. However, such an evaluation function is far from the real score on a position and does not give good results with A* and IDA*. Tests have shown that using A* and IDA* with the proposed "over-estimator" resemble a simple breadth-first search. The problem is that after expanding a node, the heuristic value of a child is significantly lower than the value of its parent, unless a move removes all blocks with one colour from the board.

Since no good evaluation function has been found yet, SameGame presents a new challenge for the puzzle research. In the next section we will discuss our SP-MCTS method.

4 MONTE-CARLO TREE SEARCH

This section first gives a description of SP-MCTS in Subsection 4.1. Thereafter we will explain the Meta-Search extension in Subsection 4.2.

4.1 SP-MCTS

MCTS is a best-first search method, which does not require a positional evaluation function. MCTS builds a search tree employing Monte-Carlo evaluations at the leaf nodes. Each node in the tree represents an actual board position and typically stores the average score found in the corresponding subtree and the number of visits. MCTS constitutes a family of tree-search algorithms applicable to the domain of board games [7, 10, 18].

In general, MCTS consists of four steps, repeated until time has run out [8]. (1) A *selection strategy* is used for traversing the tree from the root to a leaf. (2) A *simulation strategy* is used to finish the game starting from the leaf node of the search tree. (3) The *expansion strategy* is used to determine how many and which children are stored as promising leaf nodes in the tree. (4) Finally, the result of the MC evaluation is propagated backwards to the root using a *back-propagation strategy*.

Based on MCTS, we propose an adapted version for puzzles: Single-Player Monte-Carlo Tree Search (SP-MCTS). Below, we will discuss the four corresponding phases and point out differences between SP-MCTS and MCTS.

Selection Strategy Selection is the strategic task that selects one of the children of a given node. It controls the balance between $exploitation$ and $exploration$. Exploitation is the task to focus on the move that led to the best results so far. Exploration deals with the less promising moves that still may have to be explored, due to the uncertainty of their evaluation so far. In MCTS at each node starting from the root a child has to be selected until a leaf node is reached. Several algorithms have been designed for this setup [7, 10].

Kocsis and Szepesvári [18] proposed the selection strategy UCT (Upper Confidence bounds applied to Trees). For SP-MCTS, we use a modified UCT version. At the selection of node N with children N_i, the strategy chooses the move, which maximises the following formula.

$$\overline{X} \quad + \quad C \cdot \sqrt{\frac{ln\ t\,(N)}{t\,(N_i)}} \quad + \quad \sqrt{\frac{\sum x^2 - t\,(N_i) \cdot \overline{X}^2 + D}{t\,(N_i)}}. \quad (1)$$

The first two terms constitute the original UCT formula. It uses the number of times $t\,(N)$ that node N was visited and the number of times $t\,(N_i)$ that child N_i was visited to give an upper confidence bound for the average game value $\overline{X}$. For puzzles, we added a third term, which represents the deviation [10, 6]. This term makes sure that nodes, which have been rarely explored, are not under-estimated. $\sum x^2$ is the sum of the squared results achieved in this node so far. The third term can be tuned by the constant D. Coulom [10] chooses a move according to the selection strategy only if $t\,(N_i)$ reached a certain threshold (here 10). Before that happens, the simulation strategy is used, which will be explained later. Below we describe two differences between puzzles and two-player games, which may affect the selection strategy.

First, the essential difference between the two is the *range of values*. In two-player games, the results of a game can be summarised by *loss*, *draw*, and *win*. They can be expressed as numbers from the set $\{-1, 0, 1\}$. The average score of a node will always stay in $[-1,1]$. In a puzzle, a certain score can be achieved that is outside this interval. In SameGame there are positions, which can be finished with a value above 4,000 points. If the maximum score for a position would be known, then it is possible to scale this value back into the mentioned interval. However, the maximum score of a position might not be known. Thus, much higher values for the constants C and D have to be chosen than is usual in two-player games.

A second difference for puzzles is that there is *no uncertainty on the opponent's play*. This means that solely the line of play has to be optimised regarding the top score and not the average of a subtree.

Simulation Strategy Starting from a leaf node, random moves are played until the end of the game. In order to improve the quality of the games, the moves are chosen pseudo-randomly based on heuristic knowledge.

In SameGame, we have designed two static simulation strategies. We named these strategies "TabuRandom" and "TabuColourRandom". Both strategies aim at making large groups of one colour. In SameGame, making large groups of blocks is advantageous.

"TabuRandom" chooses a random colour at the start of a simulation. It is not allowed to play this colour during the random simulations unless there are no other moves possible. With this strategy large groups of the chosen colour will be formed automatically.

The new aspect in the "TabuColourRandom" with respect to the previous strategy is that the chosen colour is the colour most frequently occurring at the start of the simulation. This may increase the probability of having large groups during the random simulation.

Expansion Strategy The expansion strategy decides which nodes are added to the tree. Coulom [10] proposed to expand one child per simulation. With his strategy, the expanded node corresponds to the first encountered position that was not present in the tree. This is also the strategy we used for SameGame.

Back-Propagation Strategy During the back-propagation phase, the result of the simulation at the leaf node is propagated backwards to the root. Several back-propagation strategies have been proposed in the literature [7, 10]. The best results that we have obtained was by using the plain average of the simulations. Therefore, we update (1) the average score of a node. Additional to this, we also update (2) the

sum of the squared results because of the third term in the selection strategy (see Formula 1), and (3) the best score achieved so far for computational reasons.

The four phases are iterated until the time runs out.[5] When this happens, a final move selection is used to determine, which move should be played. In two-player games (with an analogous run-out-of-time procedure) the best move according to this strategy will be played by the player to move and the opponent then has time to calculate his response. But in puzzles this can be done differently. In puzzles it is not needed to wait for an unknown reply of an opponent. Because of this, it is possible to perform one large search from the initial position and then play all moves at once. With this approach all moves at the start are under consideration until the time for SP-MCTS runs out.

4.2 Meta-Search

A Meta-Search is a search method that does not perform a search on its own but uses other search processes to arrive at an answer. For instance, Gomes *et al.* [14] proposed a form of iterative deepening to handle heavy-tailed scheduling tasks. The problem was that the search was lost in a large subtree, which would take a large amount of time to perform, while there are shallow answers in other parts of the tree. The possibility exists that by restarting the search a different part of the tree was searched with an easy answer.

We discovered that it is important to generate deep trees in SameGame (see Section 5.2). However, by exploiting the most-promising lines of play, the SP-MCTS can be caught in local maxima. So, we extended SP-MCTS with a straightforward form of Meta-Search to overcome this problem. After a certain amount of time, SP-MCTS just restarts the search with a different random seed. The best path returned at the end of the Meta-Search is the path with the highest score found in the searches. Section 5.3 shows that this form of Meta-Search is able to increase the average score significantly.

5 EXPERIMENTS AND RESULTS

Subsection 5.1 shows tests of the quality of the two simulation strategies TabuRandom and TabuColourRandom. Thereafter, the results of the parameter tuning are presented in Subsection 5.2. Next, in Subsection 5.3 the performance of the Meta-Search on a set of 250 positions is shown. Finally, Subsection 5.4 compares SP-MCTS to IDA* and Depth-Budgeted Search (used in the program by Billings [4]).

5.1 Simulation Strategy

In order to test the effectiveness of the two simulation strategies we used a test set of 250 randomly generated positions.[6] We applied SP-MCTS without the Meta-Search extension for each position until 10 million nodes were reached in memory. These runs typically take 5 to 6 minutes per position. The best score found during the search is the final score for the position. The constants C and D were set to 0.5 and 10,000, respectively. The results are shown in Table 1.

Table 1 shows that the TabuRandom strategy has a significant better average score (i.e., 700 points) than plain random. Using the TabuColourRandom strategy the average score is increased by another 300 points. We observe that a low standard deviation is achieved for the random strategy. In this case, it implies that all positions score almost equally low.

Table 1. Effectiveness of the simulation strategies

	Average Score	StDev
Random	2,069	322
TabuRandom	2,737	445
TabuColourRandom	3,038	479

5.2 SP-MCTS Parameter Tuning

This subsection presents the parameter tuning in SP-MCTS. Three different settings were used for the pair of constants $(C; D)$ of Formula 1, in order to investigate which balance between exploitation and exploration gives the best results. These constants were tested with three different time controls on the test set of 250 positions, expressed by a maximum number of nodes. The three numbers are 10^5, 10^6 and 5×10^6. The short time control refers to a run with a maximum of 10^5 nodes in memory. In the medium time control, 10^6 nodes are allowed in memory, and in long time control 5×10^6 nodes are allowed. We have chosen to use nodes in memory as measurement to keep the results hardware-independent. The parameter pair (0.1; 32) represents $exploitation$, (1; 20,000) performs $exploration$, and (0.5; 10,000) is a balance between the other two.

Table 2 shows the performance of the SP-MCTS approach for the three time controls. The short time control corresponds to approximately 20 seconds per position. The best results are achieved by exploitation. The score is 2,552. With this setting the search is able to build trees that have on average the deepest leaf node at ply 63, implying that a substantial part of the chosen line of play is inside the SP-MCTS tree. Also, we see that the other two settings are not generating a deep tree.

In the medium time control, the best results were achieved by using the balanced setting. It scores 2,858 points. Moreover, Table 2 showed that the average score of the balanced setting increased most compared to the short time control, viz. 470. The balanced setting is now able to build substantially deeper trees than in the short time control (37 vs. 19). An interesting observation can be made by comparing the score of the exploration setting in the medium time control to the exploitation score in the short time control. Even with 10 times the amount of time, exploring is not able to achieve a significantly higher score than exploiting.

The results for the long experiment are that the balanced setting again achieves the highest score with 3,008 points. Now its deepest node on average is at ply 59. However, the exploitation setting only scores 200 points fewer than the balanced setting and 100 fewer than exploration.

From the results presented we may draw two conclusions. First we may conclude that it is important to have a deep search tree. Second, exploiting local maxima can be more advantageous than searching for the global maxima when the search only has a small amount of time.

[5] In general, there is no time limitation for puzzles. However, a time limit is necessary to make testing possible.

[6] The test set can be found online at http://www.cs.unimaas.nl/maarten.schadd/SameGame/TestSet.txt

Table 2. Results of SP-MCTS for different settings

10^5 nodes	Exploitation (0.1; 32)	Balanced (0.5; 10,000)	Exploration (1; 20,000)
Average Score	2,552	2,388	2,197
Standard Deviation	572	501	450
Average Depth	25	7	3
Average Deepest Node	63	19	8
10^6 nodes	(0.1; 32)	(0.5; 10,000)	(1; 20,000)
Average Score	2,674	2,858	2,579
Standard Deviation	607	560	492
Average Depth	36	14	6
Average Deepest Node	71	37	15
5×10^6 nodes	(0.1; 32)	(0.5; 10,000)	(1; 20,000)
Average Score	2,806	3,008	2,901
Standard Deviation	576	524	518
Average Depth	40	18	9
Average Deepest Node	69	59	20

5.3 Meta-Search

This section presents the performance tests of the Meta-Search extension of SP-MCTS on the set of 250 positions. We remark that the experiments are time constrained. Each experiment could only use 5×10^5 nodes in total and the Meta-Search distributed these nodes fairly among the number of runs. It means that a single run can take all 5×10^5 nodes, but that two runs would only use 250,000 nodes each. We used the exploitation setting (0.1; 32) for this experiment. The results are depicted in Figure 2.

Figure 2 indicates that already with two runs instead of one, a significant performance increase of 140 points is achieved. Furthermore, the maximum average score of the Meta-Search is at ten runs, which uses 50,000 nodes for each run. Here, the average score is 2,970 points. This result is almost as good as the best score found in Table 2, but with the difference that the Meta-Search used one tenth of the number of nodes. After ten runs the performance decreases because the generated trees are not deep enough.

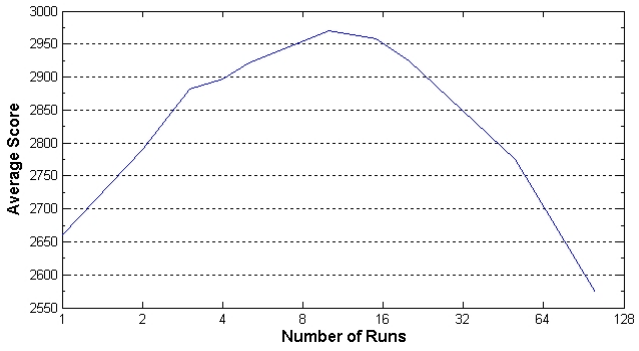


Figure 2. The average score for different settings of the Meta-Search

5.4 Comparison

The best SameGame program so far has been written by Billings [4]. This program performs a non-documented method called Depth-Budgeted Search (DBS). When the search reaches a depth where its budget has been spent, a greedy simulation is performed. On a standardised test set of 20 positions[7] his program achieved a total score of 72,816 points with 2 to 3 hours computing time per position. Using the same time control, we tested SP-MCTS on this set. We used again the exploitation setting (0.1; 32) and the Meta-Search extension, which applied 1,000 runs using 100,000 nodes for each search process. For assessment, we tested IDA* using the evaluation function described in Section 3. Table 3 compares IDA*, DBS, and SP-MCTS with each other.

Table 3. Comparing the scores on the standardised test set

Position nr.	IDA*	DBS	SP-MCTS
1	548	2,061	2,557
2	1,042	3,513	3,749
3	841	3,151	3,085
4	1,355	3,653	3,641
5	1,012	3,093	3,653
6	843	4,101	3,971
7	1,250	2,507	2,797
8	1,246	3,819	3,715
9	1,887	4,649	4,603
10	668	3,199	3,213
11	1,073	2,911	3,047
12	602	2,979	3,131
13	667	3,209	3,097
14	749	2,685	2,859
15	745	3,259	3,183
16	1,647	4,765	4,879
17	1,284	4,447	4,609
18	2,586	5,099	4,853
19	1,437	4,865	4,503
20	872	4,851	4,853
Total:	22,354	72,816	73,998

[7] The positions can be found at the following address: http://www.js-games.de/eng/games/samegame.

SP-MCTS outperformed DBS on 11 of the 20 positions and was able to achieve a total score of 73,998. Furthermore, Table 3 shows that IDA* does not perform well for this puzzle. It plays at the human beginner level. The best variants discovered by SP-MCTS can be found on our website.[8] There we see that SP-MCTS is able to clear the board for all of the 20 positions. This confirms that a deep search tree is important for SameGame as was seen in Subsection 5.2.

By combining the scores of DBS and SP-MCTS we computed that at least 75,152 points can be achieved for this set.

6 CONCLUSIONS AND FUTURE RESEARCH

In this paper we have shown how MCTS addresses NP-complete puzzles. As a representative puzzle, we have chosen the game SameGame and have proven that it is NP-complete. We proposed a new MCTS variant called Single-Player Monte-Carlo Tree Search (SP-MCTS) as an alternative to more classical approaches that solve (NP-complete) puzzles, such as A* and IDA*. We adapted MCTS by two modifications resulting in SP-MCTS. The modifications are (1) the selection strategy and (2) the back-propagation strategy. Below we provide three observations and subsequently two conclusions.

6.1 Conclusions

First, we observed that our TabuColourRandom strategy (i.e., reserving the most frequent occurring colour to be played last) significantly increased the score of the random simulations in SameGame. Compared to the pure random simulations, an increase of 50% in the average score is achieved.

Next, we observed that it is important to build deep SP-MCTS trees. Exploiting works better than exploring at short time controls. At longer time controls the balanced setting achieves the highest score, and the exploration setting works better than the exploitation setting. However, exploiting the local maxima still leads to comparable high scores.

Third, with respect to the extended SP-MCTS endowed with a straightforward Meta-Search, we observed that for SameGame combining a large number of small searches can be more beneficial than doing one large search.

From the results of SP-MCTS with parameters (0.1; 32) and with Meta-Search set on a time control of around 2 hours we may conclude that SP-MCTS produced the highest score found so far for the standardised test set. It was able to achieve 73,998 points, breaking Billings' record by 1,182 points. So, our program with SP-MCTS may be considered at this moment the world's best SameGame program.

A second conclusion is that we have shown that SP-MCTS is applicable to a one-person perfect-information game. SP-MCTS is able to achieve good results on the NP-complete game of SameGame. This means that SP-MCTS is a worthy alternative for puzzles where a good admissible estimator cannot be found. Even more, SP-MCTS proves to be an interesting solution to solving similar tractable instances of NP-complete problems.

6.2 Future Research

In the future, more enhanced methods will be tested on SameGame. We mention three of them. First, knowledge can be included in the selection mechanism. A method to achieve this is called *Progressive Unpruning* [8]. Second, this paper demonstrated that combining small searches can achieve better scores than one large search. However, there is no information shared between the searches. This can be achieved by using a transposition table, which is not cleared at the end of a small search. Third, the Meta-Search can be parallelised asynchronously to take advantage of multi-processor architectures.

Furthermore, to test our theories about the successfulness of SP-MCTS in solving other NP-Complete problems, we would like to investigate how well this method performs on, for instance, (3-) SAT problems.

[8] The best variations can be found at the following address: http://www.cs.unimaas.nl/maarten.schadd/SameGame/Solutions.html

ACKNOWLEDGEMENTS

This work is funded by the Dutch Organisation for Scientific Research (NWO) in the framework of the project TACTICS, grant number 612.000.525.

REFERENCES

[1] L. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Rijksuniversiteit Limburg, Maastricht, The Netherlands, 1994.

[2] T. C. Biedl, E. D. Demaine, M. L. Demaine, R. Fleischer, L. Jacobsen, and I. Munro. The Complexity of Clickomania. In R. J. Nowakowski, editor, *More Games of No Chance, Proc. MSRI Workshop on Combinatorial Games*, pages 389–404, MSRI Publ., Berkeley, CA, Cambridge University Press, Cambridge, 2002.

[3] N. L. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory 1736-1936*. Clarendon Press, Oxford, UK, 1976.

[4] D. Billings. Personal Communication, University of Alberta, Canada, 2007.

[5] B. Bouzy and T. Cazanave. Computer Go: An AI-Oriented Survey. *Artificial Intelligence*, 132(1):39–103, 2001.

[6] G. M. J. B. Chaslot, S. De Jong, J-T. Saito, and J. W. H. M. Uiterwijk. Monte-Carlo Tree Search in Production Management Problems. In P. Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, pages 91–98, Namur, Belgium, 2006.

[7] G. M. J. B. Chaslot, J-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik. Monte-Carlo Strategies for Computer Go. In P. Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, pages 83–91, Namur, Belgium, 2006.

[8] G. M. J. B. Chaslot, M. H. M. Winands, J. W. H. M. Uiterwijk, H. J. van den Herik, and B. Bouzy. Progressive strategies for Monte-Carlo Tree Search. In P. Wang et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.

[9] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[10] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *Proceedings of the 5th International Conference on Computer and Games*, volume 4630 of *Lecture Notes in Computer Science (LNCS)*, pages 72–83. Springer-Verlag, Heidelberg, Germany, 2007.

[11] J. C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[12] R. W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46(3):271–281, 1990.

[13] A. Felner, U. Zahavi, Jonathan Schaeffer, and R. C. Holte. Dual Lookups in Pattern Databases. In *IJCAI*, pages 103–108, Edinburgh, Scotland, UK, 2005.

[14] C. P. Gomes, B. Selman, K. McAloon, and C. Tretkoff. Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems. In *AIPS*, pages 208–213, Pittsburg, PA, 1998.

[15] P. E. Hart, N. J. Nielson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernatics*, SSC-4(2):100–107, 1968.

[16] D. S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 67–161. 1990.

[17] A. Junghanns. *Pushing the Limits: New Developments in Single Agent Search*. PhD thesis, University of Alberta, Alberta, Canada, 1999.

[18] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo Planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proceedings of the EMCL 2006*, volume 4212 of *Lecture Notes in Computer Science (LNCS)*, pages 282–293, Berlin, 2006. Springer-Verlag, Heidelberg, Germany.

[19] R. E. Korf. Depth-first iterative deepening: An optimal admissable tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[20] K. Moribe. Chain shot! *Gekkan ASCII*, (November issue), 1985. (In Japanese).

[21] PDA Game Guide.com. Pocket PC Jawbreaker Game. *The Ultimate Guide to PDA Games*, Retrieved 7.1.2008. http://www.pdagameguide.com/jawbreaker-game.html.

[22] A. Sadikov and I. Bratko. Solving 20 $\times$ 20 Puzzles. In H. J. van den Herik, J. W. H. M. Uiterwijk, M. H. M. Winands, and M. P. D. Schadd, editors, *Proceedings of the Computer Games Workshop 2007 (CGW 2007)*, pages 157–164, Universiteit Maastricht, Maastricht, The Netherlands, 2007.

[23] J. J. Schneider and S. Kirkpatrick. *Stochastic Optimization*, Chapter Application of Neural Networks to TSP, pages 405–413. Springer-Verlag, Berlin Heidelberg, Germany, 2006.

[24] University of Alberta GAMES Group. GAMES Group News (Archives), 2002. http://www.cs.ualberta.ca/ games/archives.html.