CMPUT 657 Project Documentation

Shan Lu
slu1@ualberta.ca

1 Introduction

This report describes the designs, implementations, and experiments of several Single-Player Monte-Carlo Tree Search (SP-MCTS) variants on the puzzle SameGame. Section 2 introduces the designs of several SP-MCTS variants implemented in this project. Section 3 describes my implementations of the puzzle SameGame and the SP-MCTS variants. And section 4 presents and compares the experimental results of the SP-MCTS variants on SameGame.

2 Design

SP-MCTS is a variant of Monte-Carlo Tree Search (MCTS) proposed for single-player games like SameGame. Like MCTS, SP-MCTS builds a search tree using four steps: selection, expansion, simulation, and backpropagation. The following subsections describe the four steps in detail.

2.1 Selection

The selection step of SP-MCTS is to select a leaf node in the search tree while balancing exploration and exploitation. Starting from the root of the search tree, the selection step selects one of the child nodes based on the given strategy. And the selection step ends when a leaf node is selected.

Based on the selection strategy UCT (Upper Confidence bounds applied to Trees) proposed in [4], Schadd [1] proposes a modified UCT version as the selection strategy. As described in [1], at the node p, it selects the child node i with the maximum value of the following formula:

$$v_i + C * \sqrt{\frac{\ln n_p}{n_i}} + \sqrt{\frac{\sum r^2 - n_i * v_i^2 + D}{n_i}} \tag{1}$$

where v_i is the average game score achieved at the child node i, n_p and n_i are the numbers of visits of the parent node p and the child node i respectively. These two terms are the same as the original UCT formula, where the first term controls the exploitation by choosing the child node with a higher average game score, and the second term controls the exploration by choosing the child node with a smaller number of visits. The modified UCT adds a third term, where $\sum r^2$ is the sum of the squared game scores achieved at the child node i so far, $n_i * v_i^2$ is the expected value of the squared game scores, and D is a constant corresponding to the exploration to ensure rarely visited nodes are selected.

As mentioned in [1], since the puzzle SameGame is a single-player game that does not have any uncertainty caused by the opponent player, Schadd [1] adds a variant to the modified UCT formula:

$$0.98 * v_i + 0.02 * top\_score + C * \sqrt{\frac{\ln n_p}{n_i}} + \sqrt{\frac{\sum r^2 - n_i * v_i^2 + D}{n_i}} \tag{2}$$

where top_score is the highest game score achieved at the child node i so far, and this encourages the child node with a higher top score to be selected.

Another variant of the selection strategy is similar to the selection mechanism in [1] and [2], the selection strategy is used only when the number of visits of a node exceeds a threshold T. Otherwise, a selection strategy based on the prior knowledge of the game is used to select the child node at the early stage of SP-MCTS. At the node p whose the number of visits is less than a threshold T, the child node i with the maximum heuristic game score is selected, where the heuristic game score is computed by the game board status as the following formula:

$$current\_game\_score + (the\_number\_of\_pieces\_of\_the\_most\_frequent\_color - 2)^2 \tag{3}$$

Since a good strategy of solving SameGame is to form a large block of the same color, this heuristic encourages the child with a higher number of pieces of a single color to be selected. In this way, a larger block of the same color can be formed during the simulation step with the TabuColorRandom strategy discussed in the later section.

Besides, a variant of the selection strategy inspired by Crazy Stone's selectivity algorithm proposed in [2] is implemented in this project. At the node p, the child nodes are sorted by their top game scores μ_i so that $\mu_0 > \mu_1 > ... > \mu_N$, and the corresponding variance of a child node i is computed as

$$\sigma^2 = \frac{\sum r^2 - n_i * v_i^2 + P}{n_i + 1} \tag{4}$$

Where P is the possible maximum score of the current game board by assuming colored pieces are all connected for each color. Then, similar to [2], each child node is selected with a probability proportional to

$$u_i = \exp(-10.0 * \frac{\mu_0 - \mu_i}{\sqrt{2 * (\sigma_0^2 + \sigma_i^2)}}) + \frac{0.1 + 2^{-i}}{n_i} \tag{5}$$

As discussed in [2], this selection strategy selects each child node according to its probability of being better than the current best child node.

Moreover, Klein [3] proposes another variant of the selection strategy of SP-MCTS. At the node p, it selects the child node i with the maximum value of the following formula:

$$\frac{v_i}{5000} + C * \sqrt{\frac{\ln n_p}{n_i}} \tag{6}$$

This formula is called the static normalization, which normalizes the average game score achieved at the child node i by a constant value of 5000 for SameGame. This normalization method encourages the exploration at the early stage of the SP-MCTS, and it encourages the exploitation at the late stage of the SP-MCTS.

2.2 Expansion

Same as the expansion strategy used in [1] and [2], one non-expanded child node of the selected node is added to the search tree in each expansion step.

2.3 Simulation

Given the newly expanded node, the simulation strategy is applied until the game ends. The Random, TabuRandom, and TabuColorRandom strategies proposed in [1] are implemented. The Random strategy randomly selects a child move of the current game board until the game ends. The TabuRandom strategy randomly selects a color at the beginning of the simulation. Then it randomly selects a child move of non-selected colors, and it only makes the child move of the selected color when there are no other moves available. This strategy forms large blocks of the selected color during the simulation. Since forming a large block of the same color is a good strategy for solving SameGame, similar to the TabuRandom strategy, the TabuColorRandom strategy selects the color that appears most frequently at the beginning of the simulation. This strategy may increase the size of the block with the selected color formed during the simulation. And same as [1], the ϵ-greedy policy, which randomly chooses a child move with the probability ϵ, is used to deviate the TabuRandom and TabuColorRandom simulation strategies.

2.4 Backpropagation

After the simulation is done, the backpropagation step updates the results of the simulation to the nodes on the path from the newly expanded node to the root. Same as [1], during the backpropagation step, the average game score, the number of visits, and the top game score achieved so far are updated.

3 Implementation

3.1 SameGame

The BoardState class, defined in BoardState.h and BoardState.cpp files, represents the SameGame game state. The following subsections describe the main features of the BoardState class.

3.1.1 Board Representation

The board is represented by a 15 x 15 2D vector of integers (1 to 5), where each integer represents a color. When a game board is generated initially, an integer from 1 to 5 is selected randomly for each position on the board. And the number of pieces for each color is recorded accordingly.

3.1.2 The Generation of Child Moves Functionality

To generate child moves for a given board state, the generateChildMoves method checks every colored position on the board, and a bitset that records whether a colored position is checked or not is kept. For a given colored position, the find_connectedBlock method is called to find all connected positions with the same color. It uses a queue data structure to store the same colored positions that are connected to the given position and are not haven their neighbors checked. While the queue is not empty, a position with the same color is popped from the queue, and four orthogonally connected neighbors of the popped position are checked. For each orthogonally connected neighbor, if it has the same color and is not checked, it is pushed into the queue, and the corresponding position of this neighbor in the bitset is set to true. When the queue is empty, the given position is a valid child move only when the size of the block of the given position is greater than 1. That means each valid child move, which is a block with the same color, is represented by one of the positions in this block.

3.1.3 The Making Move Functionality

The makeMove method takes a valid child move, which is one of the positions of a colored block, as the input parameter. Same as the find_connectedBlock method, a queue data structure is used to find all connected positions with the same color of the given position. And all connected positions with the same color are set to 0 (which indicates the empty position on the board). The number of colored pieces of the removed block is deducted accordingly, and the score of the game board, computed as $(removed\_block\_size - 2)^2$, is updated as well. After this, the compactBoard method is called to move all colored pieces above the removed block down. And if there are any empty columns, all positions on the right of the empty columns are moved left to fill the empty columns.

3.2 Single-Player Monte-Carlo Tree Search

Several variants of SP-MCTS introduced in Section 2 are implemented. The following sections describe the implementations of the main features of the node of the search tree, SP-MCTS, and the variants of the selection, expansion, simulation, and backpropagation strategies.

3.2.1 The Node of the Monte-Carlo Tree

The node of the Monte-Carlo Tree is represented by the MCTS_Node class. Each node in the search tree contains the following elements: the board state, a vector of pointers to child nodes, a vector of untried child moves, a pointer to the parent node, the number of visits, the total game scores achieved so far, the total squared game scores achieved so far, the top game score. The vector of pointers to child nodes is a vector of MCTS_Node pointers, where each pointer points to a child node of this node. The vector of untried child moves is a vector of child moves of the board state of this node, and such moves have not been made and added as the child nodes of this node. The pointer to the parent node is a MCTS_Node pointer that points to the parent node of this node. And the number of visits is the number of times that this node is on the path from the selected node that performs the simulation step to the root node.

The computeUCT method returns the UCT value of the node, which is computed by one of the formulas ((1), (2), (6)), each one corresponds to a variant of SP-MCTS. And the appendChildNode method takes a pointer to the child node as the input parameter and appends this pointer to the vector of child node pointers.

3.2.2 SP-MCTS

The functionalities of SP-MCTS are implemented in the MCTS class. The search method is used to start a new search. The search method keeps a vector of MCTS_Node instances in the memory as the nodes of the Monte-Carlo Tree, and the search ends when the number of nodes in the memory exceeds a threshold. When the number of nodes in the search tree is less than the threshold, a MCTS_Node pointer (selected_node_ptr: the pointer that points to the selected node) points to the root node initially. When the selected node has no untried child moves and has child nodes, the selection step is performed to set the pointer of the selected node to one of the child nodes following the given selection strategy. Then, the expansion step is performed to add a child node to the search tree, and the pointer to the selected node points to the newly added child node. After this, the simulation step is performed from the selected node until the game ends. And the game score result from the simulation step is passed into the backpropagation step to update the nodes on the path from the selected node to the root. When the search ends, similar to [1], the final move is selected according to the maximum top score, so the top score achieved by the root node is the maximum game score that can be achieved by the root node, which is the final result of the search.

3.2.3 Selection

Several variants of the selection strategy are implemented in this project. And the input parameter of the selection method is the pointer of the selected node. As described in Section 2.1, three variants of the selection strategy are to set the pointer of the selected node to the child node of the previously selected node, which has the maximum UCT value computed by the formulas (1), (2), (6) respectively (implemented in the selection method). Another variant of the selection strategy is to select the child node with the maximum heuristic game score, computed by the formula (3), when the number of visits of the previously selected node is less than a threshold $T = 10$ (implemented in the selection_heuristic method). Moreover, the selection_ProgressivePruning method implements the selection strategy similar to Crazy Stone's selectivity algorithm [2]. It sorts the child nodes based on their top game scores and computes the corresponding variance using the formula (4). Then the non-normalized probability for each child node is computed using the formula (5). And a randomly generated probability value is used to select one of the child nodes based on the normalized probabilities.

3.2.4 Expansion

The expansion method implements the expansion step. Given the reference of the pointer to the selected node and the reference of the search tree, it pops the last untried child move, and a copied board state of the parent node makes this child move. Then, an instance of MCTS_Node is created using the child board state and added to the search tree. And the parent node pointer of this child node is pointed to the given parent node, the pointer to this child node is added to the vector of child node pointers of the parent node as well. Lastly, the pointer of the currently selected node points to this newly added child node.

3.2.5 Simulation

Three variants of the simulation strategies proposed in [1] are implemented in this project. The input parameter of the simulation method is the pointer to the selected node. The rollout_Random method randomly selects one of the child moves until the game ends. The rollout_TabuRandom method selects a color randomly. And until the game ends, it selects a child move of a different color, if no child moves of a different color are available, a child move of the selected color is selected randomly. The rollout_TabuColorRandom method selects the color with the maximum number of pieces using the getMaxColor method. And the child move selection mechanism is same as the rollout_TabuRandom method. Moreover, for each child move selection, a probability value is sampled from a uniform distribution. And a child move is selected randomly if this probability is less than $\epsilon = 0.03$.

3.2.6 Backpropagation

The backpropagation method takes the pointer of the selected node and the game score from the simulation as the input parameters. Until the pointer of the selected node has no parent node, which means it points to the root node, the number of visits, the total game scores, and the top game scores of the selected node are updated. And the pointer of the currently selected node is updated to point to the parent node.

4 Experiment Results

Experiments are performed on each variant of SP-MCTS introduced in the previous sections. For each variant of SP-MCTS, the settings of the experiments are the same, where 250 boards of SameGame are tested, and the number of nodes in the Monte-Carlo Tree is 10^6. Each board is created by passing the integer from 0 to 249 as the random seed, so each variant of SP-MCTS experimented on the same set of game boards. And the evaluation metrics are the sum of average game scores, the sum of top game scores, and the average time consumed for a game board.

Table 1 compares the performances of the selection strategies using three different UCT values. The selection strategy used by all three variants is the plain selection strategy (the selection method) that maximizes the UCT value. And The simulation strategy used by all three variants is the rollout_TabuColorRandom method.

Table 1

	Total Average Scores	Total Top Scores	Average Time (seconds)
Modified UCT (formula 1)	**518516**	548253	121.7
Modified UCT with the Weighted Top Score (formula 2)	516688	546063	**119.9**
Static Normalization (formula 6)	485780.7	**574212**	393.6

From Table 1, the Static Normalization method achieves the highest top scores, but it is approximately one-third as fast as the other two methods. The modified UCT method and the modified UCT with the weighted top score method have similar total top scores and average times.

Table 2 compares the performance of three simulation strategies: Random, TabuRandom, and TabuColorRandom. And the selection strategy that maximizes the modified UCT with the weighted top score is used by all three variants of the simulation strategies.

Table 2

	Total Average Scores	Total Top Scores	Average Time (seconds)
Random	342601	351681	**62.1**
TabuRandom	442054	528816	118.4
TabuColorRandom	**517560**	**546586**	114.2

From Table 2, although the TabuColorRandom strategy is approximately one-half as fast as the Random strategy, the total average scores and the total top scores achieved by the TabuColorRandom strategy are noticeably higher than the other two strategies.

Table 3 compares the performance of three selection strategies. And the TabuColorRandom simulation strategy is used by all three variants of the selection strategies. The first selection strategy is the plain selection method that maximizes the modified UCT with the weighted top score. The second selection strategy is the one similar to Crazy Stone's selectivity algorithm [2]. And the third selection strategy uses the heuristic score to select a child node when the number of visits is less than a threshold $T = 10$; otherwise, the plain selection method is used to select a child node.

Table 3

	Total Average Scores	Total Top Scores	Average Time (seconds)
Plain Selection	521297	548893	118.5
Selection that is similar to Crazy Stone's Selectivity Algorithm	334782.6	**611263**	1003.4
Selection with the Heuristic Score ($T < 10$) and Plain Selection ($T >= 10$)	**526138**	547749	**107.2**

From Table 3, the selection strategy that is similar to Crazy Stone's selectivity algorithm [2] achieves the highest total top scores, and its average top score is approximately 250 points higher than the other two selection strategies. But the time consumed by the second selection strategy is approximately 10 times of the time consumed by the other two selection strategies.

In conclusion, the TabuColorRandom strategy is the best simulation strategy among all tested simulation strategies. And for the selection strategy, the selection method that is similar to Crazy Stone's selectivity algorithm [2] increases the top game score noticeably, but it is more computationally expensive. One possible improvement for this selection strategy could be using a priority queue, whose key is the top score, to sort the child nodes during the construction of the search tree, and this might save the time consumed by sorting child nodes in every selection step.

References

[1] Schadd, M.P., Winands, M.H., Tak, M.J. and Uiterwijk, J.W., 2012. Single-player Monte-Carlo tree search for SameGame. Knowledge-Based Systems, 34, pp.3-11.
[2] Coulom, R., 2006, May. Efficient selectivity and backup operators in Monte-Carlo tree search. In International conference on computers and games (pp. 72-83). Springer, Berlin, Heidelberg.
[3] Klein, S., 2015. Attacking SameGame using Monte-Carlo tree search: using randomness as guidance in puzzles.
[4] Kocsis, L. and Szepesvári, C., 2006, September. Bandit based monte-carlo planning. In European conference on machine learning (pp. 282-293). Springer, Berlin, Heidelberg.