Nested Monte-Carlo Search

Tristan Cazenave
LAMSADE
Université Paris-Dauphine
Paris, France
cazenave@lamsade.dauphine.fr

Abstract

Many problems have a huge state space and no good heuristic to order moves so as to guide the search toward the best positions. Random games can be used to score positions and evaluate their interest. Random games can also be improved using random games to choose a move to try at each step of a game. Nested Monte-Carlo Search addresses the problem of guiding the search toward better states when there is no available heuristic. It uses nested levels of random games in order to guide the search. The algorithm is studied theoretically on simple abstract problems and applied successfully to three different games: Morpion Solitaire, SameGame and 16x16 Sudoku.

1 Introduction

When there is no available heuristic, it can be useful to perform random playouts in order to evaluate the interest of developing a position. Moreover, it is important to optimize moves at all stages of a game and not only near the root. Nested Monte-Carlo Search uses random playouts at its base level. A search at a given level uses searches at the lower level to decide which move to play in its game. Since a complete game is performed at each level the moves are optimized at all stages.

The point of this paper is to show that random moves can be successfully used at the base level of a nested search algorithm, and that memorizing the best sequence is very useful in that case. A theoretical analysis of the algorithm as well as its successful application to three different games with huge state spaces are presented.

The outline of this paper is as follows: the next section presents related work, section 3 presents the Nested Monte-Carlo Search algorithm, section 4 analyzes the algorithm on two simple abstract problems, section 5 gives experimental results for three different games.

2 Related work

The simplest Monte-Carlo search algorithm is Iterative Sampling, it consists in playing random games until a solution is found or the search time is elapsed.

Rollouts were successfully used by Tesauro and Galperin to improve their Backgammon program [Tesauro and Galperin, 1996].

Nested rollouts combined with an heuristic to choose the next move at the base level were used by Yan et al. to improve their Klondike solitaire program [Yan et al., 2005]. Nested rollouts have been used with heuristics that change with the stage of the game of Thoughtful Solitaire, a version of Klondike Solitaire in which the locations of all cards is known [Bjarnason et al., 2007]. These algorithms use a base heuristic which is improved with nested rollouts, whereas our algorithm uses random moves at the base level.

A related algorithm is Reflexive Monte-Carlo search [Cazenave, 2007] which has been used to find long sequences at Morpion Solitaire. The idea of Reflexive Monte-Carlo search has some similarity with the nested rollouts idea, it consists in playing random playouts at the base level, and to play a few games at the lower level of a search in order to find the best move at the current level of the search. Games at the meta level give better results than games at the lower level. In Reflexive Monte-Carlo search, there is a fixed number of games played at each level of the search before deciding the move to play. Whereas in Nested Monte-Carlo Search each possible move is tried only once before each lower level search.

The use of Monte-Carlo methods in games has been recently very successful for the game of Go [Gelly and Silver, 2007].

3 The algorithm

Nested Monte-Carlo Search combines nested calls with randomness in the playouts and memorization of the best sequence of moves. In nested rollouts the rollouts are based on a heuristic. It implies that nested rollouts always improves on rollouts and on simply following the heuristic. When the base level does not use a heuristic but random moves, it is possible that a nested search gives worse results than a lower level search. It is then useful to memorize the best sequence found so far in order to follow it when the randomized searches give worse results than the best sequence.

The basic sample function just plays a random game from a given position, we use the function \texttt{play(position, move)} which plays the move in the position and returns the resulting position:
int sample (position)
1 while not end of game
2 position = play (position, random move)
3 return score

The Nested Monte-Carlo Search function plays a game, choosing at each step of the game the move that has the highest score of the lower level Nested Monte-Carlo Search. At each step the algorithm tries all possible moves, plays a nested search at the lower level after each move, and memorizes the move associated to the best score of the lower level searches. As the samples are randomized, it is not guaranteed that a nested search will always improve on previous searches or even lower level searches. In order not to lose the best moves of the best sequence found by a previous search, the algorithm memorizes the best sequence. If none of the moves improve on the best sequence, the move of the best sequence is played, otherwise the best sequence is updated with the newly found sequence and the best move is played:

int nested (position, level)
1 best score = -1
2 while not end of game
3 if level is 1
4 move = argmax_m (sample (play (position, m)))
5 else
6 move = argmax_m (nested (play (position, m), level - 1))
7 if score of move > best score
8 best score = score of move
9 best sequence = seq. after move
10 bestMove = move of best sequence
11 position = play (position,bestMove)
12 return score

The algorithm can be made anytime with iterative calls:

int iterativeNested (position, level)
1 bestScore = -1
2 while time left
3 score = nested (position, level)
4 bestScore = max (bestScore, score)
5 return bestScore

4 Analysis of the algorithm
In order to explain how Nested Monte-Carlo search behaves, we analyze it on two very simple abstract problems. The search tree of both problems can be represented as a binary tree. In each state there are only two possible moves: going to the left or going to the right.

4.1 The leftmost path problem
The scoring function of the first problem consists in counting the number of moves on the leftmost path of the tree. Let’s call this problem the leftmost path problem. The search space of the leftmost path problem for depth 3 is depicted in figure 1. A sample search that chooses moves randomly has a probability $2^{-n}$ of finding the best score of a depth n problem. A depth-first search that order moves randomly has one chance out of two of choosing the wrong move at the root, so the mean complexity of finding the best score with a depth-first search is at least $2^n - 2$. A level 1 Nested Monte-Carlo Search will always find the best score and its complexity is $n \times (n - 1)$. Nested Monte-Carlo Search is appropriate for the leftmost path problem because the scores at the leaves are extremely correlated with the structure of the search tree.

4.2 The left move problem
However, leaf scores of a problem are not usually as correlated to the structure of the tree as in the leftmost path problem. We define the left move problem as the problem where the score of a leaf is the number of moves to the left that have been made during a game. A depth 3 left move problem search tree is depicted in figure 2. Sample search and depth-first search behave the same as in the leftmost path problem. Nested Monte-Carlo Search is less well informed in this problem. At the root of a depth d tree, the number of leaves that have a given score s is \(\binom{s}{d}\), in the left branch this number is \(\binom{s-1}{d-1}\), concerning the right branch this number of leaves is \(\binom{s}{d-1}\). The probability that a sample search starting with a left move finds the score s is therefore $P_{leftscore}(s, d, 0) = \binom{s-1}{d-1}$. The probability that a sample search starting with a right move finds the score s is
A Nested Monte-Carlo Search program for the left move problem. A Nested Monte-Carlo Search that does not memorize the best sequence improves much less with the level than a search that does memorize it. A level 3 search with memorization has some chances of finding the best score, whereas a search without memorization has very little chances of finding it. The experimental distribution of figure 3 has been compared to the theoretical results given by the dynamic programming algorithm. The experimental results are within 1.02% of the theoretical distribution.

### 4.3 Real-time properties

It is clear that the distribution of the scores improves with the level. However a level \( n + 1 \) search takes more time than a level \( n \) search. In order to compare the programs according to the time they take to search, we ran 100 iterative Nested Monte-Carlo Search of 82 seconds for levels 0 to 3. For times starting at 0.01 second and doubling until 81.92 seconds we have the mean score reached with an iterated Nested Monte-Carlo Search.
Monte-Carlo Search. Figure 5 give the mean scores for different times and levels of a search with memorization of the best sequence. The mean score of a search increases almost linearly with the logarithm of the time. A level 1 search is much better than a level 0 search (6 points), a level 2 search is 2 points better than a level 1 search and a level 3 search is one point better than a level 2 search. In a real-time setting increasing the level of the search is beneficial until level 3 for the left move problem. Figure 6 give similar results for searches without memorization of the best sequence. We see that a level 1 search is better than a level 0 search, however level 2 and 3 are worse than level 1. For the left move problem nested calls are not beneficial at level 2 and 3 if the best sequence is not memorized.

5 Experimental results

Nested Monte-Carlo Search was experimented on three quite different games: Morpion Solitaire, SameGame and 16x16 Sudoku.

5.1 Morpion Solitaire

Morpion Solitaire is an NP-hard puzzle and the high score is inapproximable within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless $P = NP$ [Demaine et al., 2006]. A move consists in adding a circle such that a line containing five circles can be drawn. Lines can either be horizontal, vertical or diagonal. The starting position already contains circles disposed as in figure 7. In the disjoint version a circle cannot be a part of two lines that have the same direction. The best human score at Morpion Solitaire disjoint version is 68 moves [Demaine et al., 2006]. We have tested a level 4 Nested Monte-Carlo Search on Morpion solitaire and obtained an 80 moves grid after 5 hours of computation on a cluster of 32 dual core computers. The grid is given in figure 7. On a single machine a run at level 4 takes approximately 10 days.

Figure 7: A world record found by Nested Monte-Carlo Search at Morpion Solitaire disjoint version
In Morpion Solitaire a nested search of level $l$ is 200 times longer than a nested search of level $l - 1$. We can guess from the figure 9 that playing 200 games at level $l - 1$ is likely to give a worse score than playing one game at level $l$ for $l \leq 4$. In order to test this assumption, we computed the mean score of an iterated search for given times and levels. The results are depicted in figure 10. It is clear that a level 2 search is better than a level 1 search which is better than a level 0 search. Similarly to the left move problem, the increase in score is almost linear with the logarithm of the time. So given a time limit it is advisable to choose the highest level $\leq 4$ that can be searched within the time limit, and to use iterative Nested Monte-Carlo Search.

5.2 SameGame

SameGame is an NP-complete puzzle [Kendall et al., 2008]. It consists in a grid composed of cells of different colors. Adjacent cells of the same color can be removed together, scoring $(\text{numberOfCellsRemoved} - 2)^2$. When cells are removed, the upper cells fall down, and when a column is empty the columns to the right of the empty column are moved to the left. There is a bonus of 1,000 points for removing all the cells.

In the simulations we used the TabuColorRandom strategy [Schadd et al., 2008]. It means that at the beginning of each playout the color that has the most cells is set as the tabu color. During the playouts, moves of the tabu color are played only if there are no moves of the others colors.

The previous best algorithm at SameGame used SP-MCTS based on restarts of the UCT algorithm [Schadd et al., 2008], it scored 73,998 on a standard test set. With similar time settings IDA* scored a total of 22,354 and Darse Billings program scored 72,816 [Schadd et al., 2008]. Nested Monte-Carlo search is more simple and gives better results at SameGame since it scores 77,934 with a level 3 search with memorization of the best sequence that corresponds roughly to the previous time settings.

Table 1 gives the scores of SP-MCTS and of a level 3 search for the 20 positions of the test set. In order to evaluate the interest of memorizing the best sequence at SameGame a level 2 search was also performed with memorization of the best sequence (level 2m) and without memorization (level 2). Memorizing the best sequence clearly improves the search since its total score is 65,937 when not memorizing only scores 44,731 at level 2.

5.3 16x16 Sudoku

Sudoku is a popular NP-complete puzzle [Kendall et al., 2008] usually played on a 9x9 grid. Some cells are empty and others are filled with a number. The goal is to fill all the empty cells with numbers between 1 and 9 such that all the numbers in a row are different, all the numbers in a column are also different, and all the numbers in predefined 3x3 squares are also different.

Instead of using the usual 9x9 grid, we have used a 16x16 grid in order to have more difficult problems. The principle is the same except that numbers range from 1 to 16 and that squares have size 4x4. We have modeled Sudoku as a constraint satisfaction problem. Each cell is a variable that may contain the sixteen possible values. Each time a cell is associated to a value, all the variables in the same row, column or square are updated and the value is removed from their domain. As soon as a variable has an empty domain the search
Search to a level 2 search is not beneficial, maybe because Iterative Sampling takes much less time than Forward Checking. If the search time for a problem exceeds 20,000 seconds, Forward Checking is unable to solve 21 problems out of 100. Iterative sampling takes much less time than Forward Checking and solves all the problems. Nested Monte-Carlo Search is clearly much better than Forward Checking, and better than Iterative Sampling. Going from a level 1 Nested Monte-Carlo Search to a level 2 search is not beneficial, maybe because the problems are already easy for a level 1 search. A level 1 search without memorization of the best sequence takes 7.00 seconds instead of 1.34 seconds. A level 2 search without memorization takes 4.87 seconds instead of 1.64 seconds.

### Table 2: Results for 16x16 Sudoku with 66% of empty cells

<table>
<thead>
<tr>
<th>Position</th>
<th>SP-MCTS</th>
<th>level 2</th>
<th>level 2m</th>
<th>level 3m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,557</td>
<td>1,442</td>
<td>1,805</td>
<td>3,121</td>
</tr>
<tr>
<td>2</td>
<td>3,749</td>
<td>1,543</td>
<td>3,151</td>
<td>3,813</td>
</tr>
<tr>
<td>3</td>
<td>3,085</td>
<td>1,817</td>
<td>2,707</td>
<td>3,085</td>
</tr>
<tr>
<td>4</td>
<td>3,641</td>
<td>2,077</td>
<td>3,221</td>
<td>3,697</td>
</tr>
<tr>
<td>5</td>
<td>3,653</td>
<td>2,471</td>
<td>3,084</td>
<td>4,055</td>
</tr>
<tr>
<td>6</td>
<td>3,971</td>
<td>2,394</td>
<td>3,367</td>
<td>4,459</td>
</tr>
<tr>
<td>7</td>
<td>2,797</td>
<td>1,802</td>
<td>2,567</td>
<td>2,949</td>
</tr>
<tr>
<td>8</td>
<td>3,715</td>
<td>2,480</td>
<td>3,785</td>
<td>3,999</td>
</tr>
<tr>
<td>9</td>
<td>4,603</td>
<td>3,598</td>
<td>3,897</td>
<td>4,695</td>
</tr>
<tr>
<td>10</td>
<td>3,213</td>
<td>2,040</td>
<td>3,075</td>
<td>3,223</td>
</tr>
<tr>
<td>11</td>
<td>3,047</td>
<td>1,486</td>
<td>2,687</td>
<td>3,147</td>
</tr>
<tr>
<td>12</td>
<td>3,131</td>
<td>1,995</td>
<td>2,699</td>
<td>3,201</td>
</tr>
<tr>
<td>13</td>
<td>3,097</td>
<td>1,265</td>
<td>2,857</td>
<td>3,197</td>
</tr>
<tr>
<td>14</td>
<td>2,859</td>
<td>1,163</td>
<td>2,413</td>
<td>2,799</td>
</tr>
<tr>
<td>15</td>
<td>3,183</td>
<td>1,871</td>
<td>2,937</td>
<td>3,677</td>
</tr>
<tr>
<td>16</td>
<td>4,879</td>
<td>3,468</td>
<td>4,607</td>
<td>4,979</td>
</tr>
<tr>
<td>17</td>
<td>4,609</td>
<td>3,263</td>
<td>4,397</td>
<td>4,919</td>
</tr>
<tr>
<td>18</td>
<td>4,853</td>
<td>3,015</td>
<td>4,689</td>
<td>5,201</td>
</tr>
<tr>
<td>19</td>
<td>4,503</td>
<td>2,822</td>
<td>4,287</td>
<td>4,883</td>
</tr>
<tr>
<td>20</td>
<td>4,853</td>
<td>2,719</td>
<td>3,705</td>
<td>4,835</td>
</tr>
<tr>
<td>Total</td>
<td>73,998</td>
<td>44,731</td>
<td>65,937</td>
<td>77,934</td>
</tr>
</tbody>
</table>

Table 1: Results for SameGame

### References


