Reflexive Monte-Carlo Search

Tristan Cazenave

LIASD
Dept. Informatique
Université Paris 8, 93526, Saint-Denis, France
cazenave@ai.univ-paris8.fr

Abstract. Reflexive Monte-Carlo search uses the Monte-Carlo search of a given level to improve the search of the upper level. We describe the application to Morpion Solitaire. For the non touching version, reflexive Monte-Carlo search breaks the current record and establishes a new record of 78 moves.

1 Introduction

Monte-Carlo methods have been applied with success to many games. In perfect information games, they are quite successful for the game of Go which has a huge search space [1]. The UCT algorithm [9] in combination to the incremental development of a global search tree has enabled Go programs such as CRAZY STONE [2] and MOGO [11] to be the best on 9×9 boards and to become competitive on 19×19 boards.

Morpion Solitaire is a single-player complete-information game with a huge search space. The current best algorithms for solving Morpion Solitaire are based on Monte-Carlo methods. We present in this paper an improvement of the usual Monte-Carlo method based on reflexivity.

In the next section we present Morpion Solitaire. In the third section we detail reflexive Monte-Carlo search. In the fourth section we give experimental results.

2 Morpion Solitaire

Morpion Solitaire was first published in Science & Vie in April 1974. It has also been addressed in Jeux & Stratégie in 1982 and 1983. The current record of 170 moves has been found by hand at this time.

2.1 Rules of the Game

Figure 1 gives the starting position for Morpion Solitaire. A move consists in adding a circle and drawing a continuous line such that the line contains the new circle as well as four other circles. A line can be horizontal, vertical or diagonal. The goal of the game is to play as many moves as possible.

Figure 2 shows a game after five moves. In this figure only horizontal and vertical moves are displayed, diagonal moves are also allowed when possible.

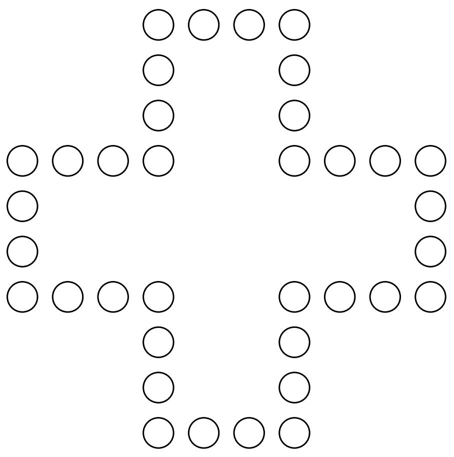

Fig. 1. Initial position.

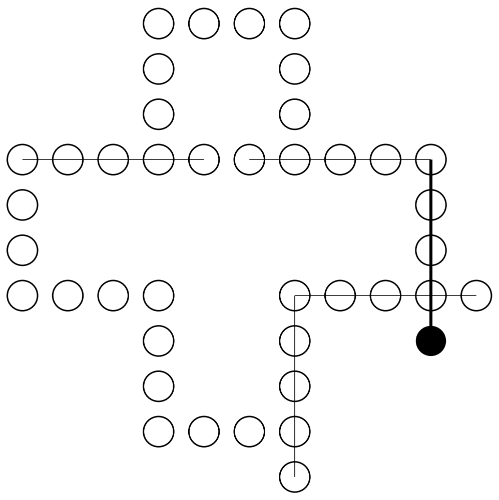

Fig. 2. Position after five moves.

In the original version of the game, two different moves can share a circle at the end of a line even if the lines of the two moves have the same direction. Another version is the non-touching version where two different moves cannot share a circle at the end of a line if their lines have the same direction.

2.2 Previous works on Morpion Solitaire

Hugues Juillé has described an algorithm that finds a 122 moves solution to the touching version [7, 8]. The algorithm was made more incremental by Pascal Zimmer and it reached 147 moves. The 170 moves record has been proved optimal for the last 109 moves [4, 5].

Concerning the non-touching version, the best human record is 68 moves [3]. In his thesis, B. Helmstetter describes a program that finds a 69-moves solution using a retrograde analysis of the 68-moves human record [5]. In December 2006, the best record was 74. It was found with a simulated-annealing algorithm [10, 6]. I found a 76-moves record in December 2006 with Reflexive Monte-Carlo Search, and the 78-moves record in May 2007.

2.3 The Mobility Heuristic

An heuristic for choosing the moves to explore is the *mobility* heuristic. It consists in choosing randomly among the moves that lead to boards that have the maximum number of possible moves. The moves that lead to board that have less possible moves than the boards corresponding to the most mobile moves are discarded.

3 Reflexive Monte-Carlo Search

This section first presents Monte-Carlo search applied to Morpion Solitaire, then it is extended to reflexive Monte-Carlo Search.

3.1 Monte-Carlo search

Monte-Carlo search simply consists in playing random games from the starting position. The code for playing a random game is given in the function $playGame()$:

```
int variation [200];

int playGame () {
  nbMoves = 0;
  for (;;) {
    if (moves.size () == 0)
      break;
    move = chooseRandomMove (moves);
    playMove (move);
    variation [nbMoves] = move;
    updatePossibleMoves (move);
    nbMoves++;
  }
  return nbMoves;
}
```

Moves are coded with an integer, and the $variation$ array contains the sequence of moves of the random game.

The $playGame()$ function is used to find the best game among many random games with the function $findBestMove()$:

```
int nbPrefix = 0;
int prefix [200];
int previousBestScore = 0;
int bestVariation [200];

int findBestMove () {
  int nb = 0, best = previousBestScore - 1;
  for (int i = 0; i < best; i++)
    bestVariation [i] = bestVariation [i + 1];

  initialBoard ();
  for (int i = 0; i < nbPrefix; i++)
     playMove (prefix [i]);
  memorizeBoard ();
  while (nb < nbGames) {
    retrieveBoard ();
    if (moves.size () == 0)
      return -1;
    int nbMoves = playGame ();
    if (nbMoves > best) {
      best = nbMoves;
      for (int i = 0; i < best; i++)
        bestVariation [i] = variation [i];
    }
  }
  previousBestScore = best;
  return bestVariation [0];
}
```

The $prefix$ array contains the moves that have to be played before the random game begins. In the case of a simple Monte-Carlo search, $nbPrefix$ equals zero. In the case of a Meta-Monte-Carlo search the prefix contains the moves of the current meta-game.

The $initialBoard()$ function puts the board at the initial position of Morpion Solitaire. The $memorizeBoard()$ function stores the current board with the list of possible moves, and the $retrieveBoard()$ function restores the board and the list of possible moves that have been stored in the $memorizeBoard()$ function.

3.2 Meta Monte-Carlo Search

Reflexive Monte-Carlo search consists in using the results from Monte-Carlo search to improve Monte-Carlo search. A Meta-Monte-Carlo search uses a Monte-Carlo search to find the move to play at each step of a meta-game. A Meta-Meta-Monte-Carlo search uses a Meta-Monte-Carlo search to find a move at each step of a meta-meta-game, and so on for upper meta levels.

We give below the $playMetaGame()$ function which plays a meta-game:

```
int playMetaGame () {
  previousBestScore = 0;
  for (;;) {
    int move = findBestMove ();
    if (move == -1)
      break;
    prefix [nbPrefix] = move;
    nbPrefix++;
  }
  return nbPrefix;
}
```

This function is used to find the best meta-move in a Meta-Meta-Monte-Carlo search. We give below the code for the $findBestMetaMove()$ function that finds the best meta-move:

```
int nbMetaPrefix = 0;
int metaPrefix [200];
int previousMetaBestScore = 0;
int bestMetaVariation [200];

int findBestMetaMove () {
  int nb = 0, best = previousMetaBestScore - 1;
  for (int i = 0; i < best; i++)
    bestMetaVariation [i] = bestMetaVariation [i + 1];
  while (nb < nbMetaGames) {
    for (int i = 0; i < nbMetaPrefix; i++)
      prefix [i] = metaPrefix [i];
    nbPrefix = nbMetaPrefix;
    int nbMoves = playMetaGame ();
    if (nbMoves - nbMetaPrefix > best) {
      best = nbMoves - nbMetaPrefix;
      for (int i = 0; i < best; i++)
        bestMetaVariation [i] = prefix [nbMetaPrefix + i];
    }
    nb++;
  }
  previousMetaBestScore = best;
  if (best <= 0)
    return -1;
  return bestMetaVariation [0];
}
```

The code for the meta-meta level is similar to the code for the meta-level. In fact all the meta-levels above the ground level use a very similar code. There is no theoretical limitation to the number of levels of reflexive Monte-Carlo search. The only limitation is that each level takes much more time than its predecessor.

4 Experimental Results

For the experiments, the programs run on a Pentium 4 cadenced at 2.8 GHz with 1 GB of RAM. All experiments concern the non-touching version.

Table 1 gives the lengths of the best games obtained with sampling and random moves for different numbers of random games. It corresponds to a simple Monte-Carlo search.

Table 1. Best lengths for different numbers of games with sampling and random moves.

games	10	100	1,000	10,000	100,000	1,000,000	10,000,000
length	55	58	60	63	63	63	64
time	0.008s.	0.12s.	1.0s.	9.8s.	98s.	978s.	17,141s.

Table 2 gives the lengths of the best games obtained with sampling and the *mobility* heuristic for choosing moves in the random games. We can observe that for a similar number of random games, the *mobility* heuristic finds longer games than the completely random choices. However, it also takes more time, and for equivalent search times, the two heuristic find similar lengths.

Table 2. Best lengths for different numbers of games with sampling and mobility.

games	10	100	1,000	10,000	100,000	1,000,000
length	60	61	62	62	63	64
time	0.12s.	1.05s.	10.2s.	101s.	1,005s.	10,063s.

The UCT algorithm uses the formula $\mu_i + \sqrt{\frac{log(t)}{C \times s}}$ to explore moves of the global search tree, μ_i is the average of the games that start with the corresponding move, t is the number of games that have been played at the node, and s is the number of games that have been played below the node starting with the corresponding move. We have tested different values for the C constant, and the best results were obtained with $C = 0.1$.

Table 3 gives the best lengths obtained with UCT for different numbers of simulations. The results are slightly better than with sampling.

Table 3. Best lengths for different numbers of games with UCT and mobility.

games	10	100	1,000	10,000	100,000	1,000,000
length	60	60	61	64	64	65
time	0.16s.	1.3s.	16s.	176s.	2,108s.	18,060s.

Reflexive Monte-Carlo search has been tested for different combinations of games and meta-games, each combination corresponds to a single meta-meta-game. The length of the best sequence for different combinations is given in Table 4. The first line gives the number of meta-games, and the first column gives the number of games. Table 5 gives the time for each combination to complete its search.

Table 4. Best lengths for different numbers of games and meta-games.

	1	10	100	1000
1	63	63	67	67
10	63	67	69	67
100	65	69	72	
1000	71			

Table 5. Times for different numbers of games and meta-games.

	1	10	100	1,000
1	5s.	42s.	417s.	3,248s.
10	23s.	774s.	3,643s.	27,688s.
100	253s.	4,079s.	26,225s.	
1,000	3,188s.			

We have obtained sequences of length 76 with different configurations of the reflexive search. The configuration that found the 78 moves record consists in playing 100 random games at the ground level, 1,000 meta-games at the meta level, and one game at the meta meta level. The search took 393,806 seconds to complete. The sequence obtained is given in figure 3.

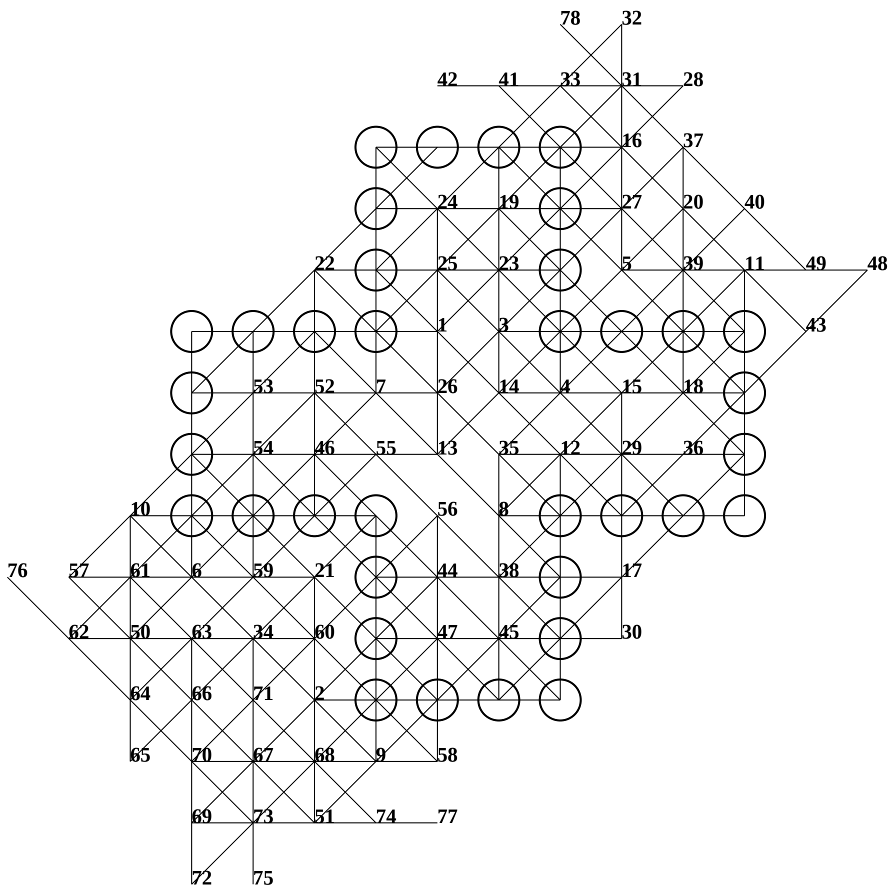

Fig. 3. 78 moves for the non-touching version.

5 Conclusion and Future Research

Reflexive Monte-Carlo search has established a new record of 78 moves for the non-touching version of Morpion Solitaire. This is four moves more than the best attempt made by other programs, and it is ten moves more than the human record for the game. Concerning the touching version the human record of 170 is still much better than what current programs can achieve.

Reflexive Monte-Carlo Search is a general technique that can be applied in other games. Also, future works include the parallelization of the algorithm and its application to other games.

References

1. Bouzy, B., Cazenave, T.: Computer Go: An AI-Oriented Survey. Artificial Intelligence **132**(1) (2001) 39–103
2. Coulom, R.: Efficient selectivity and back-up operators in monte-carlo tree search. In: Proceedings Computers and Games 2006, Torino, Italy (2006)
3. Demaine, E.D., Demaine, M.L., Langerman, A., Langerman, S.: Morpion solitaire. Theory Comput. Syst. **39**(3) (2006) 439–453
4. Helmstetter, B., Cazenave, T.: Incremental transpositions. In H. Jaap Herik, Yngvi Bjornsson, N.S.N., ed.: Computers and Games: 4th International Conference, CG 2004. Volume 3846 of LNCS, Ramat-Gan, Israel, Springer-Verlag (2006) 220–231
5. Helmstetter, B.: Analyses de dépendances et méthodes de Monte-Carlo dans les jeux de réflexion. Phd thesis, Université Paris 8 (2007)
6. Hyyro, H., Poranen, T.: New heuristics for morpion solitaire. Technical report (2007)
7. Juillé, H.: Incremental co-evolution of organisms: A new approach for optimization and discovery of strategies. In: ECAL. Volume 929 of Lecture Notes in Computer Science. (1995) 246–260
8. Juillé, H.: Methods for statistical inference: extending the evolutionary computation paradigm. Phd thesis, Brandeis University (1999)
9. Kocsis, L., Szepesvari, C.: Bandit based monte-carlo planning. In: ECML-06. (2006)
10. Langerman, S.: Morpion solitaire. Web page, http://slef.org/jeu/ (2007)
11. Wang, Y., Gelly, S.: Modifications of UCT and sequence-like simulations for Monte-Carlo go. In: CIG 2007, Honolulu, USA (2007) 175–182