Vol.2010-GI-24 No.6
2010/6/25

Regular Paper

Experiments in Monte-Carlo Amazons

Julien Kloetzer[†1]

Amazons is a new abstract board game for two players which has recently attracted attention due to its high branching factor. Recent Monte-Carlo Tree-Search methods have been applied with great success to build strong programs. We show in this paper how such an implementation benefits from a good evaluation function as well as from a good optimization between a high quantity of low quality information and a low quantity of high quality information. We also show that an MCTS implementation for Amazons greatly benefits from processing power, especially given a good evaluation.

1. Introduction

Amazons is a two-player deterministic game with perfect information with very simple rules. Although it possess elements both from the territory games world and from the piece moving games world, it has its own set of strategies and features which make it an interesting game to study. The main challenge when programming an Amazons computer program is the branching factor of the game of several hundreds[4], way superior to those of games such as Chess(35), Shogi(80) or even Go(250)[3]. The limit of 92 moves per game, however, makes it a game simpler than Go on a game-tree complexity basis.

Due to the fact that it is not that difficult to design a good evaluation function[8], several strong minimax-based programs emerged. Recent focus however has shifted towards Monte-Carlo Tree-Search based algorithms[6),9] which gave birth to some new strong programs able to compete with traditional minimax programs[10]. The best-first aspect of Monte-Carlo Tree-Search (MCTS) which allows programs to bye-pass the problem of the high branching factor of the game is obviously a factor for this success, although surely not the only one.

†1 Research Unit for Computers and Games, Japan Advanced Institute of Science and Technology; j.kloetzer@jaist.ac.jp

In this paper, we will present several experiments about MCTS Amazons. In Section 2 we present the main basis of our work on MCTS Amazons. In Sections 3 and 4 we present the result of our experiments concerning the evaluation function and the processing power. The conclusion follows in Section 5.

2. Past works

2.1 Monte-Carlo Tree-Search

Monte-Carlo for game programming has been proposed by Abramson in 1990[2]. His main idea was to replace the traditional evaluation function used in conjunction with minimax in game-playing programs with a much simpler heuristic: the winning rate. This is achieved by playing random games (also called simulations) and averaging their results. The technique however was not much used for deterministic games even though it is as much applicable as for indeterministic games and/or games with hidden information. The latter, given the stochastic aspect of the Monte-Carlo evaluation, were a more natural target for this technique.

The recent addition of Tree-Search to simple Monte-Carlo programs however has given birth to a complete new framework for game-programming called Monte-Carlo Tree-Search (MCTS)[5),7]. MCTS is much more than just a classical search using a Monte-Carlo evaluation: the search in an MCTS program launches simulations the same way as a simple Monte-Carlo program but also uses informations from past simulations to drive the search using techniques borrowed from the stochastic problems world, especially the n-armed bandit problem. A simple pseudo-code of MCTS is given in Figure 1.

This algorithm grows a tree in an asymmetric manner as it explores the game state space. The policy used in the selection part can vary, although the UCB heuristic (Upper Confidence Bound) is often used, leading to an algorithm called UCT (UCB for Trees)[7]. As for traditional Monte-Carlo the evaluation is traditionally the result of en ending position - win or lose - although, as we will see in Section 2.2, it could be otherwise.

2.2 The program Campya

Campya is the author's Amazons program. It is based on an MCTS core as described in Section 2.1. It uses the UCB heuristic as its selection policy, making it effectively a UCT program. At any position P during the search the next

```
1  function getBestMove(Position, Endingcondition)
2    while ( Endingcondition not satisfied )
3      endingnode = tree.root
4      Pos = copy(Position)
5      while ( endingnode is in the tree ) \\ selection
6        endingnode = chooseChildOf(endingnode)
7        play(Pos, move leading to endingnode)
8      end while
9      tree.add(endingnode) \\ expansion
10     while ( Pos is not ended position )
11       play(Pos, random move playable from Pos)
12     end while
13     V = evaluation(Pos) \\ evaluation
14     while( endingnode =/= tree.root ) \\ back-propagation
15       update(endingnode, V)
16       endingnode = endingnode.fathernode
17     end while
18   end while
19   return(move m with highest value)
20 end function
```

Fig. 1 Pseudocode for a basic Monte-Carlo Tree-Search playing engine

move is selected according to the UCB policy: if at least one move leads to a position which has not been explored yet by the algorithm, one of these moves is chosen at random; otherwise, the move M maximizing $Value(P,M) + C * \sqrt{\frac{ln(\sum_{Pchildren\ c} V(P,c))}{V(P,M)}}$ is chosen, where $Value(P,M)$ is the computed value for the position reached from P by playing M, and $V(p,m)$ is the number of times that move m was chosen from position p at this time of the computation. In the random games, the moves are chosen completely randomly.

Apart from that, two main features distinguish it from other MCTS program. First, since Amazons move are made of two steps, the game-tree is double-level. Second, the evaluation is different, which will be presented in Section 3.

3. The MCTS Amazons evaluation

As it has already been shown by Kloetzer *et. al*[6] and Lorentz[9], the traditional Monte-Carlo evaluation does not work for the game of the Amazons. Instead of playing complete random games and evaluate them by their end-state, MCTS Amazons programs play short random games which ending position is evaluated using an evaluation function. In the next two sections, we will present our experiments to study the effect of the evaluation function as well as of the length of the random games.

3.1 Parameters of the evaluation

Lieberum describes in[8] three important components for an Amazons evaluation function:

- The Queen Distance (QD) which gives an estimation of the territory
- The King Distance (KD) which rewards a good distribution of the Amazons on the board
- The Mobility (Mob) which punishes Amazons enclosed in narrow spaces

Since these three components do not bear the same importance at any moment of the game (QD is more important near the end of the game, while KD and Mob are more important at the beginning), he also suggests a way to measure the progress in the game.

We will study in this section the effect of three evaluation functions:

- A simple evaluation based on QD alone (simple)
- A more complex combination of the three features described above (complex)
- The result of the complex evaluation to which is applied a step function, giving a result of 0 to negative evaluations and of 1 to positive ones; this evaluation should be the closest to the traditional Monte-Carlo evaluation (ratio)

We will also study the effect of the length of the random games according to three implementations:

- All random games played stop at the same depth from the root
- All random games have the same length
- All random games have the same length except if they end up at an odd depth, in which case one more move is played to end them at an even depth

The Amazons evaluation function has a very strong odd-even effect, so the third implementation should lower this effect compared to the second one.

3.2 Experiments

All experiments were run on Athlon 2.20Ghz or similar machines with at least 1Gb of memory, allowing Campya to play around 10000 simulations per second

for the simple evaluation, and around 6000 for the complex and ratio evaluations. Each version of CAMPYA tested played several hundreds of games against other versions of the program using various settings or against alpha-beta programs, after what Bayeselo was used to compute ELO ratings. Each version of the program was evaluated using either 40000 simulations per move or 5 minutes per game and with an optimized version of the C constant in the UCB formula (presented in Section 2.2).

We first measured the performance of the various evaluations, which are summarized in Table 1. For completeness sake, we also included the performance of the classical Monte-Carlo evaluation, that is the percentage of win. It appears quite clearly that the change of the evaluation is necessary, but also that, despite it requiring more computation time, the use of a more complex evaluation (complex or ratio) gives a great boost to the performance of the program. It also appears that the ratio evaluation is slightly superior to the complex one, although not by much.

Setting	Endgame evaluation	Classic evaluation	Complex evaluation	Ratio evaluation
40.000 samples	780	1500	1741	1779
5 minutes	686	1541	1809	1835

Table 1 ELO rating for each of the three evaluations as well as for the full random game evaluation

We provide next in Figures 2, 3 and 4 the ratings obtained by varying the length and implementation of the random games as presented in Section 3.1.

The first thing we notice is that for any of the implementation as well as for any of the evaluation there exists an optimum value for the random games (sometimes two), which is surprisingly very low (3, 5 or 8 depending on the implementation). This means that, although the random games are very short in the evaluation of an MCTS Amazons program, the information they provide is necessary, but at the same time they should not be too long. To sum up, there exist an optimum between lots of information of lower quality (long random games) and few information of higher quality (short random games) and the exploration provided by the random games is necessary. This last points is confirmed by the bad performance obtained by our program if we evaluate nodes directly, that is if we cut completely the random games (experiments not included here).

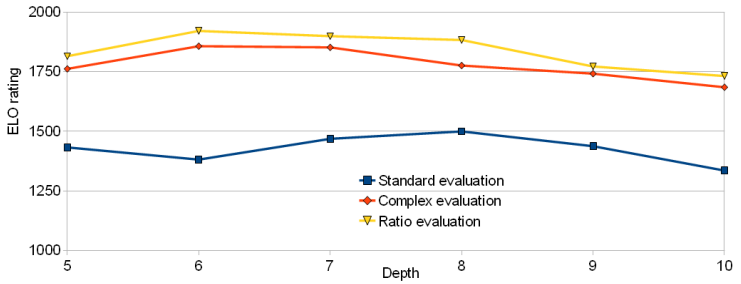

Fig. 2 ELO ratings for various depth of cut and evaluations for the random games

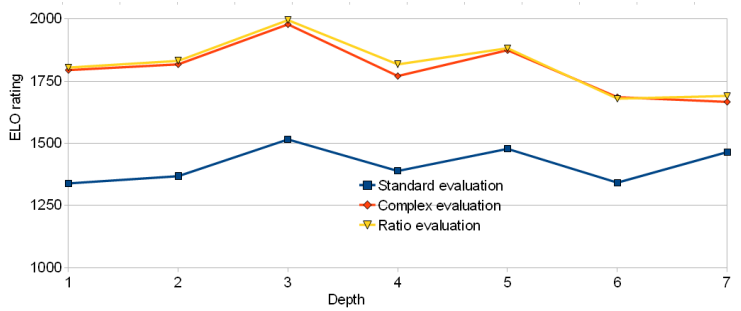

Fig. 3 ELO ratings for various random games length and evaluations

It is also quite clear that the implementation consisting in giving the same length to all the random games is superior to the one consisting in evaluating all the nodes at the same depth, and that for every evaluation. However, it is not clear whether modifying this implementation so that every random game ends at an even depth really helps, since the performance of the best versions for each evaluation are pretty similar with or without this correction.

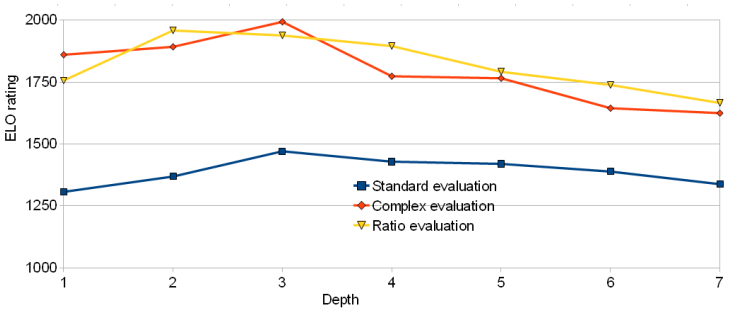

Fig. 4 ELO ratings for various random games length and evaluations, the games always ending at an even depth

4. Effect of the processing power

We present in this section our experiments concerning the effect of the processing power to the performance of our MCTS program. The experiments are performed in a similar way as presented in Section 3.2 and the results are summarized in Table 2.

Beside a horizon effect for a few number of simulations per move, the results for the game of the Amazons follow those presented for the game of Go in the Computer-Go mailing-list[1)], that is that each doubling of power gives a stable gain in ELO, between 100 and 150 in the present case. It seems however that the performance of the simple evaluation quickly reach a plateau, while the ratio evaluation benefits more from a higher processing power than the complex evaluation.

Simulations/move	2500	5000	10000	20000	40000	80000	160000
simple evaluation	1139	986	1137	1365	1500	1641	1698
complex evaluation	1404	1219	1443	1568	1737	1873	1959
ratio evaluation	1351	1136	1365	1624	1745	1937	2035

Table 2 ELO ratings for each of the three evaluations for various numbers of random games

5. Conclusion

We presented in this article results of experiments geared towards the creation of a strong MCTS Amazons playing program. We showed how the use of an evaluation function based on several features combined gives much greater performance than a simple one based on the score. It is not clear however if an implementation based on a ratio of good positions close to the traditional Monte-Carlo evaluation is much better, although we showed that it benefits more from more computation time. Finally, we also showed that the optimal length of the random games in an MCTS Amazons program exists and is very short, whatever the implementation and the evaluation used. On the future of this topic and to broader the experiments, we would like to verify if such a drastic change in the evaluation of an MCTS program could give birth to strong programs for other kind of games for which it is possible to design an evaluation function.

References

1) Computer go mailing list. `http://computer-go.org/pipermail/computer-go/`.
2) B. Abramson. Expected-outcome: a general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):182–193, 1990.
3) L.V. Allis and U.R. Limburg. *Searching for solutions in games and artificial intelligence*. PhD thesis, Maastricht: Rrijksuniversiteit Limburg, 1995.
4) H.Avetisyan and R.J. Lorentz. Selective Search in an Amazons Program. *Lecture Notes in Computer Science*, pages 123–141, 2003.
5) R.Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Lecture Notes in Computer Science*, 4630:72–83, 2007.
6) J.Kloetzer, H.Iida, and B.Bouzy. The Monte-Carlo Approach in Amazons. In *Computer Games Workshop, Amsterdam, The Netherlands*, pages 113–124, 2007.
7) L.Kocsis and C.Szepesvari. Bandit Based Monte-Carlo Planning. *Lecture Notes in Computer Science*, 4212:282–293, 2006.
8) J.Lieberum. An evaluation function for the game of amazons. *Theoretical Computer Science*, 349(2):230–244, 2005.
9) R.Lorentz. Amazons discover Monte-Carlo. In *Computers and Games, Beijing, China, September/October 2008*, pages 13–24, 2008.
10) R.Lorentz. Invader wins Amazons Event. *ICGA Journal*, 2009. to be published.