Comparative Study of Monte-Carlo Tree Search and Alpha-Beta Pruning in Amazons

Hikari Kato(✉), Szilárd Zsolt Fazekas, Mayumi Takaya, and Akihiro Yamamura

Department of Computer Science and Engineering, Akita University, 1-1, Tegata Gakuen-machi, Akita 010-8502, Japan
{m9014076,szilard.fazekas,msato,yamamura}@ie.akita-u.ac.jp

Abstract. The game of Amazons is a combinatorial game sharing some properties of both chess and Go. We study programs which play Amazons with strategies based on Monte-Carlo Tree Search and a classical search algorithm, Alpha-Beta pruning. We execute several experiments to investigate the effect of increasing the number of searches in a Monte-Carlo Tree Search program. We show that increasing the number of searches is not an efficient method to strengthen the program for Amazons. On the other hand, augmenting the algorithms with a choice of several evaluation functions fulfills has great influence on playing strength.

Keywords: Amazons · Two player games · Monte-Carlo tree search · UCT · Game programs · Playouts · Alpha-Beta pruning · Evaluation function

1 Introduction

Artificial intelligence is an important technology in our digitalized society. There are many applications of artificial intelligence to industry, e.g., data mining in big data processing, natural language processing, robotics, intelligent agents and machine learning, etc. One of the most important applications, as well as proving grounds for artificial intelligence methods is to create game playing programs for board games like chess or Go. Monte-Carlo Tree Search is one of the simple, yet often efficient approaches along this line. We shall study the performance of a simple Monte-Carlo Tree Search program playing Amazons compared with traditional artificial intelligence methods like Alpha-Beta pruning.

Alpha-Beta pruning is a search algorithm that applies an evaluation function to each leaf node in the game tree and selects the node with the highest evaluation based on the Mini-Max principle. It has been widely studied for a long time as a search program for two player games such as Shogi and Reversi. When applying this method, it is important to use strong evaluation functions [11] and enhanced pruning techniques of the game tree [10]. *Monte-Carlo Tree Search* (MCTS for short) is a search algorithm based on probability statistics,

I. Khalil et al. (Eds.): ICT-EurAsia 2015 and CONFENIS 2015, LNCS 9357, pp. 139–148, 2015.
DOI: 10.1007/978-3-319-24315-3_14

and it can create a strategy without using an evaluation function which is first implemented by Coulom [6]. This property made it a prime candidate for games for which it is difficult to create an evaluation function such as Go [8] and Arimaa [20]. For instance, it was considered difficult to write a strong program for Go using conventional Mini-Max search technique. However, programs such as CrazyStone [7], based on MCTS, were able to win computer Go tournaments, proving the validity of the approach.

The game of Amazons (Amazons for short) is a two player game [24] sharing some attributes with both chess and Go, but also being different from them in crucial ways. There are more legal moves in each turn in the game of Amazons than in chess. A strong game playing program must explore the game tree to great depth. However, searching deeply in Amazons with a simple Alpha-Beta pruning is ineffective, because the state space is huge; the number of legal moves is so great that doing a full width search is impractical throughout at least the first two thirds of an Amazons game [3]. Therefore, creating a strong player using only Alpha-Beta pruning is impossible.

Amazons has been extensively studied, see, for instance, the analysis of $2 \times n$ Amazons [4], the analysis of endgames [5,13,14], the Amazons opening book [16], and a study of creating strong programs by combining an evaluation function and MCTS [12]. Nobody succeeded in creating a strong game playing program of Amazons based on simple MCTS. Kloetzer et al. [15,17] gave much stronger approaches by combining MCTS and an evaluation function in the search process. They also showed that the strength of MCTS combined with an evaluation function for Amazons can be enhanced by increasing the number of simulations. However, we are not aware of any previous analyses of direct play between simple MCTS not using an evaluation function and Alpha-Beta pruning. As such a study would emphasize the gain brought by combining evaluation functions with MCTS, we conducted experiments in this direction.

We will carried out experiments in which a simple MCTS program not using an evaluation function plays against an Alpha-Beta pruning program using the classical evaluation function. We recorded how many times the MCTS program won against the Alpha-Beta pruning program and the average time that each program took to output a move. Our experiment showed that Alpha-Beta pruning is stronger than the simple MCTS program and increasing the number of simulations in a simple MCTS is inefficient for strengthening the strategy for Amazons.

2 The Game of Amazons

The game of Amazons is a combinatorial two-player game invented in 1988 by Zamkauskas [24] and first published (in Spanish) in issue 4 of the puzzle magazine El Acertijo in 1992. Amazons is played on a 10×10 chess-style board. The game starts by placing four black and white queens on the specified cells on the board. The first player (P_W) selects and moves one of the white queens according to the movement of a queen in chess (vertical, horizontal and diagonal straight lines on

the board). Then, P_W chooses any empty cell in the range of the queen moved and thwarts it. No piece can be placed on the thwarted cell nor pass through it thereafter. Similarly, the second player (P_B) selects and moves one of the black queens and chooses any empty cell in the range of the queen moved and thwarts it. The players P_W and P_B move alternately and the player who can no longer complete their moves (both moving a queen and thwarting a cell) loses the game. Amazons resembles Go in that it is considered good strategy for one to create their own territory, while the movement of the pieces on the board is borrowed from chess. It should be noted, that while Amazons originally uses a 10×10 board, the game can also be considered in a more general manner on an $n \times n$ board as a variant. Figure 1 shows the initial setting of an Amazons game and the board after the first player made their first move.

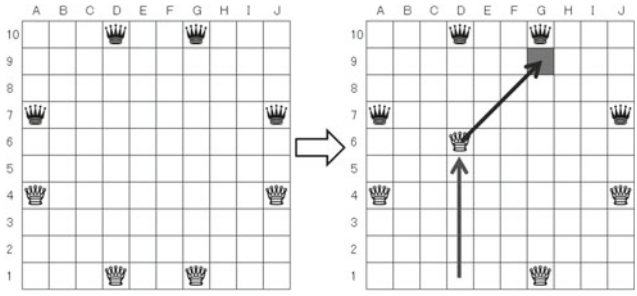

Fig. 1. From initial placement to the first movement of P_W

Amazons is known to have a very large number of legal moves in a given turn compared with other board games such as chess, Shogi or Go. For example, the number of legal moves of a player in their turn in chess is about 35 on average (see [1]), in Shogi it is 80 (see [22]) and in Go it is 361 on a 19×19 board. In contrast, in the first turn of Amazons the starting player has 2176 legal moves, and each player has 400 legal moves per turn on average even during the game. Therefore, the evaluation of the game tree involves many more states than in the case of previously mentioned board games and creating a strong computer player for Amazons is considered difficult [3].

3 Alpha-Beta Pruning

The game tree for a two-player game is a directed graph whose nodes are states in the game and whose edges are legitimate moves. Alpha-Beta pruning is an algorithm to find the best move from a state according to an evaluation function and the Min-Max principle (see [18]) by analyzing part of the game tree. The algorithm has been studied since J. McCarthy showed an idea in 1956. It is a widely used algorithm in the field of two player games; notable examples includes

chess and Reversi. First, Alpha-Beta pruning expands the game tree until a specified search depth. After that, it applies the evaluation function to the child nodes of the portion of the game tree expanded up to then. The evaluation of the nodes higher up in the tree is done by the Mini-Max principle. After obtaining the evaluation of all the nodes, the algorithm selects the move leading to the child node with the highest value.

Alpha-Beta pruning can lead to a stronger strategy if one increases the allowed search depth. However, since Amazons has a very large number of legal moves on average, it is difficult to explore the game tree to a large depth because of time and memory limitations. The Alpha-Beta pruning program in our experiment used depth-first search for evaluating the nodes to a depth of 2. Several different heuristics for Amazons have already been studied [9]. For the Alpha-Beta pruning programs we used the three evaluation functions given in [21] and described below. In what follows, by *turn player* in a state we mean the player whose turn it is to move in that state and by *opponent* we mean the other. When not specified explicitly, by evaluating a state we mean evaluation from the point of view of the turn player.

3.1 Mobility Evaluation

In Amazons, it is advantageous to have more legal moves available in one's turn because players who cannot move, lose the game. Consequently, if the number of legal moves is small in a state, it is considered to be an unfavorable game-state for the player. In other words, reducing the number of legal moves of the opponent is considered as an effective strategy. Let *mobility evaluation* (ME) of a state be the value obtained by subtracting the number of legal moves of the opponent from the number of legal moves of the turn player in a given game state. For example, if P_W has 161 legal moves in a certain game state and P_B has 166 moves, then ME of the state X from P_W's point of view is $\mathrm{ME}_W(X) = 161 - 166 = -5$.

3.2 Territory Evaluation

The concept of territory is important in Amazons. A *territory* of a player is a cell which is reachable by the queens of only that player, so advantageous game states should have many of these. The player who has access to more empty cells is in advantage because the playing area is divided in several separated subareas in the end-game of Amazons.

Figure 2 shows the minimum number of moves required to reach each cell on the game board by any of the pieces of the players. The value in the upper left corner of each blank cell is the minimum number of moves needed for P_W to reach it, whereas the value in the lower right corner represents the corresponding number required for P_B. For example, to reach cell $C6$, player P_W needs at least four moves, e.g., $B8 \rightarrow A7 \rightarrow A6 \rightarrow B5 \rightarrow C6$. In contrast, P_B requires only two moves, $B3 \rightarrow A4 \rightarrow C6$. Therefore, P_B has a faster access to $C6$ than P_W so according to territory evaluation, cell $C6$ belongs to the territory of P_B. Let

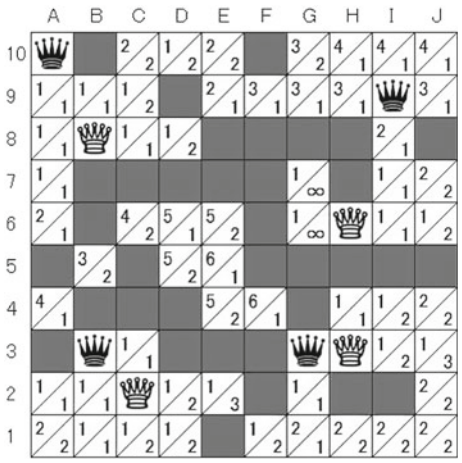

Fig. 2. Territory evaluation

$D_X(A)$ denote the minimum number of moves needed for player X to reach cell A. In the example above, $D_W(C6) = 4$ and $D_B(C6) = 2$.

We compute the minimum number of moves needed for each player in this manner for each blank cell on the board and take the sum over all blank cells to obtain the evaluation of the game-state based on the territories. We define territory evaluation of a state X for P_W as follows.

$$\mathrm{T}_W(X) = \sum_{\text{empty cells } A} \Delta_1(D_W(A), D_B(A)),$$

where

$$\Delta_1(n,m) = \begin{cases} 0 & (n = m = \infty) \\ \frac{1}{5} & (n = m < \infty) \\ 1 & (n < m) \\ -1 & (n > m) \end{cases}$$

The evaluation value of the cells that both players reach in the same number of moves may be set to an arbitrary value; in setting it to $\frac{1}{5}$ we followed [21].

3.3 Relative Territory Evaluation

The basic idea of *relative territory evaluation* is similar to that of territory evaluation. In essence, territory evaluation counts the number of cells that can be reached by a player in less moves than by the other player, disregarding the actual difference in the number of moves. In contrast, relative territory evaluation assesses the difference in the number of moves needed by the two players to reach each blank cell. Let us define relative territory evaluation of state X for P_W as follows.

$$\mathrm{RT}_W(X) = \sum_{\text{empty cells } A} \Delta_2(D_W(A), D_B(A))$$

where

$$\Delta_2(n,m) = \begin{cases} 5 & (m=\infty, n<\infty) \\ -5 & (n=\infty, m<\infty) \\ 0 & (n=\infty, m=\infty) \\ m-n & (\text{otherwise}) \end{cases}$$

For a cell which can be reached by only one of the players, the difference in number of moves is by default infinite. However, for implementation purposes it is convenient to avoid treating infinity and so we set the difference to 5 in those cases.

4 Monte-Carlo Tree Search

A Monte-Carlo algorithm [23] is a randomized algorithm whose output is allowed to be incorrect with a certain probability. Even though the answer may be incorrect, in some cases this approach can be much more efficient than using deterministic algorithms. A Monte-Carlo tree search (MCTS) [6] is a Monte-Carlo algorithm suitable for certain decision processes, most notably employed in game playing. Random simulations in a game tree called *playouts* are employed to select the next move by game playing programs. MCTS has received considerable interest due to its great success in playing *Go* [7].

MCTS employs playouts, which are simulations to determine the outcome of a game played by two players who choose their moves randomly until the game ends. As a refinement of MCTS, the method of *upper confidence bounds applied to trees (UCT)* was introduced by Kocsis and Szepesvári [19] based on the *UCB1 algorithm* proposed by Auer et al. [2]. In a game state G with child states $G_1, \ldots, G_k$, a UCT algorithm selects a child state for which the UCB1 value is maximal among the ones computed for each G_i. The UCB1 value of each child state G_i is defined by the following equation:

$$\mathrm{UCB1}(G_i) = \overline{x_i} + \sqrt{\frac{\log n}{n_i} \min(\frac{1}{4}, \overline{x_i} - \overline{x_i}^2 + \sqrt{\frac{2\log n}{n_i}})}$$

where

- n is the number of playouts executed from game state G,
- n_i is the number of playouts executed from child state G_i,
- $\overline{x_i} = \frac{x_i}{n_i}$ is the win-loss ratio of G_i, where x_i is the number of wins among playouts from G_i.

A UCT program does not necessarily have an evaluation function and its move selection depends only on the result of playouts.

Through selecting a child node, executing the playout, and repeatedly updating winning percentages, it is possible to find the most selected child node and

recommend it as the final move. UCT explores the game tree to a great depth by repeating playouts, therefore it can be configured to be a strong player by increasing the number of searches (or the allowed search time).

5 Experiment

5.1 UCT vs. Alpha-Beta

We compared a UCT algorithm and Alpha-Beta pruning by letting these two programs play Amazons against each other. This experiment not only compared the relative strength of UCT and Alpha-Beta pruning in Amazons but also aimed to evaluate the improvement of the UCT program when increasing the number of allowed searches.

We employed a simple UCT program not using heuristic techniques such as pruning. The number of playouts performed was 10000, 30000, 50000, 100000, 200000. The Alpha-Beta pruning programs had maximum search depth 2 and used one of the three types of evaluation functions described in Sect. 4, respectively.

We executed the experiment on a 10×10 board. For each match-up, we performed 50-50 simulations with the UCT program being the first player and the second player, respectively, and we recorded the number of times the UCT program won. In addition, we recorded the average time taken by UCT and Alpha-Beta pruning to make one move and the average time it took to play one game. This experiment was performed by using an Amazons match simulator written in C#, developed by us. To be able to measure the times correctly, while one game playing program is searching for a move, the other does not compute anything. Our experiments were run on a computer with Windows 7 Professional(64bit), having an Intel(R) Xeon(R) CPU E31245(3.30 GHz) and memory of 16 GB.

5.2 Experimental Results

Table 1 shows the number of times that the UCT programs (with different number of playouts) won against the three Alpha-Beta pruning players out of 100 games (50-50 as first and second player, respectively). Figure 3 shows the average time taken for a move by the programs. The vertical axis of the graph is average search time for one move; the horizontal axis represents the number of playouts performed by the UCT. The time taken by the Alpha-Beta pruning algorithms to compute a move only depends on the depth (2) and on the individual states being evaluated. As the depth was fixed, the slight variations in average computing time for the Alpha-Beta pruning programs is probably due to having to evaluate different game states reached by the changing strategy of the UCT programs and to the relatively low number of matches between the programs. We expect that increasing the number of head-to-head matches would drive the averages closer to each other approximating a horizontal line.

Table 1. UCT vs Alpha-Beta: number of wins for the UCT.

		Mobility	Territory	Relative territory
UCT	10000	71	1	9
	30000	89	8	40
	50000	90	11	43
	100000	97	24	57
	200000	100	34	66

With respect to the number of wins of UCT against Alpha-Beta pruning, the winning percentages of UCT were very different depending on the evaluation function of the opponent. territory evaluation is strongest when comparing the three types of evaluation functions. Against TE, even the UCT program with 200000 playouts won only 34 out of 100. In contrast, when playing against mobility evaluation, the UCT gained significant strength by increasing the number of playouts, the strongest one (200000 playouts) winning all 100 matches.

There are clear differences in the number of wins against the three evaluation functions, but it can be clearly seen that even when the UCT performed poorly (vs. TE) its number of wins was much higher with 200000 playouts than with 10000.

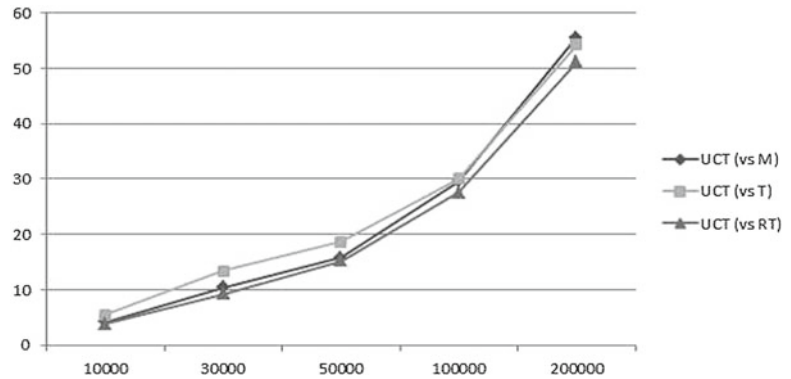

Fig. 3. UCT vs Alpha-Beta: the average time taken by the programs to make a move (sec).

Now let us see the computation time required to make one move by the programs. With 10000 playouts, the UCT needed on average 4 s to decide on a move and this is almost the same as for Alpha-Beta pruning. However, the required time increased in accordance with the increase in the number of playouts. The UCT program with 100000 playouts needed on average 30 s, while the UCT with 200000 playouts took on average 55 s. The UCT program with 200000

playouts had less wins than losses against Alpha-Beta pruning using territory evaluation, even though it required more than 30 times the computation time of the Alpha-Beta pruning program.

Meanwhile, among the three versions of Alpha-Beta pruning there was only a small difference in computing time due to the difference in the evaluation function. Alpha-Beta pruning using mobility evaluation did not record a single win against the UCT with 200000 playouts. However, against relative territory evaluation under the same conditions the UCT won only 34 times. This means that while increasing the number of playouts improved the UCT program, the gain depended heavily on the evaluation function of the opponent, while Alpha-Beta pruning was greatly enhanced by changing the heuristic. Moreover, the increase in playouts caused a significant increase in computing time for UCT, whereas the computing time for Alpha-Beta pruning was not greatly influenced by the change in the evaluation function.

6 Conclusions

We conducted an experiment in which we set UCT based strategies against Alpha-Beta pruning ones in Amazons matches. We showed that even Alpha-Beta pruning with territory evaluation is faster than the simple UCT. Increasing the number of playouts (and thus computing time, too) led to improvements in the UCT. However, with 200000 playouts allowed, the UCT program consumed 30 times more computation time and still only won 34 out of 100 games against Alpha-Beta pruning using territory evaluation. In conclusion, it looks like there is more to gain in playing strength for Amazons programs by improving the evaluation function and using classical methods like Alpha-Beta pruning than by increasing the number of playouts using the MCTS strategy.

References

1. Allis, L.V.: Searching for Solutions in games and Artificial Intelligence. Ph.D. thesis, Maastricht, Rrijksuniversiteit Limburg (1995)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learning **47**(2–3), 235–256 (2002)
3. Avetisyan, H., Lorentz, R.J.: Selective search in an Amazons program. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) CG 2002. LNCS, vol. 2883, pp. 123–141. Springer, Heidelberg (2003)
4. Berlekamp, E.R.: Sums of $N \times 2$ Amazons. Institute of Mathematical Statistics Lecture Notes - Monograph Series, vol. 35 (2000)
5. Buro, M.: Simple Amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs. In: Marsland, T., Frank, I. (eds.) CG 2001. LNCS, vol. 2063, pp. 250–261. Springer, Heidelberg (2002)
6. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)

7. Coulom, R.: Computing Elo ratings of move patterns in the game of Go. ICGA J. **30**, 198–208 (2007)
8. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo Go. Technical Report PR-6062, INRIA (2006)
9. Hensgens, P.: A Knowledge-based Approach of the Game of Amazons. Master's thesis, Maastricht University (2001)
10. Hoki, K., Muramatsu, M.: Efficiency of three forward-pruning techniques in Shogi: futility pruning, null-move pruning, and late move reduction (LMR). Entertainment Comput. **3**, 51–57 (2012)
11. Kaneko, T.: Evaluation functions of computer Shogi programs and supervised learning using game records. J. Jpn. Soc. Artif. Intell. **27**, 75–82 (2012)
12. Kloetzer, J., Iida, H., Bouzy, B.: The Monte-Carlo approach in Amazons. In: Computer Games Workshop, pp. 185–192, Amsterdam, The Netherlands (2007)
13. Kloetzer, J., Iida, H., Bouzy, B.: A comparative study of solvers in Amazons endgames. In: IEEE 2008 Symposium on Computational Intelligence in Games, Perth, Australia (2008)
14. Kloetzer, J., Iida, H., Bouzy, B.: Playing Amazons endgames. ICGA J. **32**, 140–148 (2009)
15. Kloetzer, J.: Experiments in Monte-Carlo Amazons. IPSJ SIG Technical Report, vol. 2010-GI-24 No. 6, pp. 1–4 (2010)
16. Kloetzer, J.: Monte-Carlo opening books for Amazons. In: 7th Conference on Computer and Games (2010)
17. Kloetzer, J.: Monte-Carlo techniques: application to Monte Carlo tree search and Amazon (2011). http://hdl.handle.net/10119/8867
18. Knuth, D., Moore, R.: An analysis of alpha-beta pruning. Artif. Intell. **6**, 293–326 (1975)
19. Kocsis, L., Szepesvári, C.: Bandit based Monte Carlo planning. In: 17th European Conference on Machine Learning (ECML 2006), pp. 282–293 (2006)
20. Kozelek, T.: Methods of MCTS and the game Arimaa. Master's thesis, Charles University in Prague (2009)
21. Lieberum, J.: An evaluation function for the game of Amazons. Theoret. Comput. Sci. **349**, 230–244 (2005)
22. Matsubara, H., Iida, H., Grimbergen, R.: Chess, Shogi, Go, natural developments in game research. ICCA J. **19**, 103–112 (1996)
23. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
24. Zamkauskas, W.: Amazons. http://www.chessvariants.org/other.dir/amazons.html