Monte-Carlo tree search with Epsilon-Greedy for game of amazons

Chuan Tian

Faculty of Engineering, Architecture & Information Technology, University of Queensland, St. Lucia, Brisbane, 4072, Australia

c.tian1@uqconnect.edu.au

Abstract. The Game of the Amazons is an abstract strategy board game. It has a high computational complexity similar to the game of Go. Due to its NP-complete nature and large branching factor of game tree, finding the optimal move given a specific game state is infeasible and it is not trivial to design a computer algorithm that is competitive to an expert in the game of amazons. One way to tackle this problem is to leverage the Monte-Carlo Tree Search by using random simulations. In this article, a computationally cheap heuristic function is proposed and use together with Monte-Carlo Tree Search algorithm with Epsilon-Greedy policy aiming to design a competitive AI for the Game of the Amazon. The effectiveness of the ϵ-greedy based Monte-Carlo algorithm is compared to the widely used MCTS with Upper Confidence Bound and other classical tree search method such as breadth-first search, depth-first search, minmax search and alpha-beta pruning.

Keywords: Game of the Amazons, Monte-Carlo Method, Machine Learning, ϵ-greedy.

1. Introduction

The Game of the Amazons requires two players. It is a popular abstract strategy board game. The game is played by moving pieces and placing arrows to block the opponent, and the winner would be the last player that could be able to move a piece. The rules of the Amazon game are quite simple. The board has 10-by-10 squares. Each player starts with 4 queens, placed on the board, as shown in the left part of Figure 1. The player's action is made up of two parts: first, moving of one of his queens in straight line in any directions similar to the queen in chess. Then, after moving a queen, the player could initiate another orthogonal or diagonal move, just like the movement of a queen, and shoot an arrow from its landing square to another one. A square is then blocked until the end of the game if it is shoot by the aforementioned arrow. Any subsequent movements or arrow shots cannot pass or land on it. The two players move their queens alternatively. If no queen could be moved by a player, the he or she loses this game.

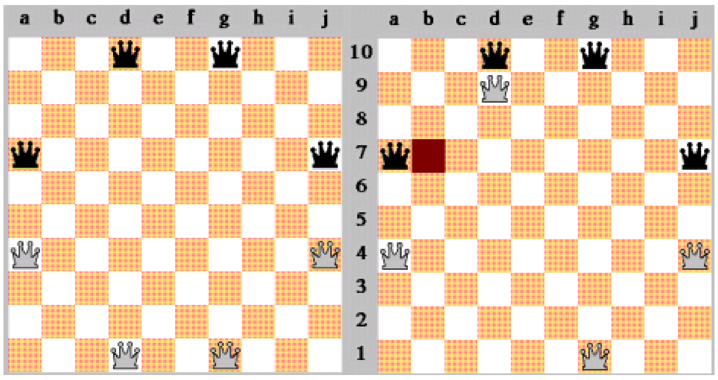

Figure 1. Starting position of queens in Amazons (left) and the board after one move (right). Figure from: http://www.solitairelaboratory.com/amazons.html.

The Game of Amazons is a relatively complex game: Buro shows that solving the game of the amazon is an NP-complete task [1]. NP-completeness of Amazon means solving the best action for a given amazon puzzle requires more than polynomial time under the assumption that $P \neq NP$. Another way to look at the complexity of amazon is by looking at the branching factor of amazons' game tree: the average possible game states per turn are around 2000 and there will be more than 4 million possible states just after two turns. Therefore, it is not practical to search the game tree by brute force.

UCT tackles this large branching factor problem and is commonly used in the AI of game of Amazon. However, other kinds of AI, by using carefully designed heuristic function can also perform well in game. For example, Region based heuristic function together with alpha-beta search has been proposed Hongxia et al [2].

Monte Carlo tree search (MCTS) belong to the family of heuristic search algorithms, which is widely applied for trees searching in board games like chess. MCTS has recently proven to be competitive when combining with neural networks [3]. AlphaGo combines neural network with MCTS. It successfully defeats the world Go human champion Lee Sedol in 2015 [4].

The Upper Confidence bounds applied to Trees (UCT) algorithm is a special MCTS, which was introduced in 2006 [5]. In UCT, upper confidence bound (UCB1) guide the node selection, transferring the process of selection into the problem of multi-armed bandit [6]. In UCT method, the important moves are more likely to be searched than the moves that tend to behave badly. Given infinite time and memory, UCT theoretically converges to Minimax [7].

MCTS is most implemented using UCT. However, other tree policy can also be used. In this article, MCTS is implemented together with ϵ-Greedy. One of the most important goals of UCT is to make trade-offs between exploration and exploitation and explore more on more valuable nodes. This same goal can also be achieved using ϵ-Greedy [8].

The overview of UCT will be introduced in Section 2, and the implementation of ϵ-greedy will be demonstrated in Section 3. The MCTS with UCB approach and classical method like minmax and alpha-beta pruning are compared in Section 4. At last, Section 5 summarizes the conclusions.

2. Monte-Carlo tree search method

MCTS focuses on analysing the most valuable method to expand the search tree on search space using random sampling. Rollout could be regarded as the key for applying the Monte Carlo tree search during the game. In each rollout, a virtual player proceeds the game until the end of the game based on randomly selected moves. The final results will be used to guide the future selection, where, for each nodes within the game tree, a promising node selection that tend to induce a successful result will be selected with higher probability.

2.1. Phases of Monte-Carlo tree search

There are four steps in each iteration of Monte Carlo tree search: Selection, Expansion, Simulation and Backpropagation [9].

2.1.1. Selection. In MCTS algorithm with UCB, the algorithm determines the procedures from current game state (denoted as root) to an unvisited or terminal state (denoted as leaf). The path is determined iteratively based on upper confidence boundary (UCB) algorithm [10]. Given a root state called s_0. The determination of the subsequent states $\{s_0, s_1, ..., s_l\}$ of the UCB is denoted as

$$UCB1(S_i) = \overline{V_i} + C\sqrt{\frac{ln(t)}{n_i}} \tag{1}$$

Where $\overline{V_i}$ is the mean value of node i, C is a constant, t is the total number of simulation, and n_i stands for number of visits of node i.

2.1.2. Expansion. Once an unvisited state s_l is achieved, it will be expanded. During the subsequent iterative selection all the child nodes will be regarded as leaf nodes.

2.1.3. Simulation. For each node in the simulation pool, a simulation is conducted by selecting moves until the terminal state is reached.

2.1.4. Backpropagation. Once values in a recently added node is determined, the remaining tree also needs to be renewed. Therefore, backpropagation is performed, which propagates backwards from the newly added node to the root node. In backpropagation, the number of simulations saved in each node is increased. In addition, if the simulation of the new node ends in a win as a terminal state, the number of wins for its parent nodes is also increased.

Algorithm 1 MCTS

```
Input: state $s_0$
$s_{current} \leftarrow s_0$
repeat
  if $s_{current}$ is a leaf node then
    if $n_i$ $equals\ to$ 0 then
      rollout( $s_{current}$ )
    end if
    Add available actions to $s_{current}$ as leaf node
    $s_{current} \leftarrow$ first child node of $s_{current}$
    rollout( $s_{current}$ )
  end if
  $s_{current} \leftarrow argmax_{S_i}(UCB1(S_i))$
until $s_{current}$ is a terminal state
```

Algorithm 2 Rollout

```
Input: state $s_i$
repeat
  if $s_i$ is terminal state then
    return Q($s_i$)
  end if
  $A_i \leftarrow random(available\ Actions(s_i))$
```

```
        $s_i \leftarrow simulate(A_i, s_i)$
    **until** $s_i$ is a terminal state
```

3. Implementation of Monte-Carlo tree search

Monte Carlo Tree Search is implemented with ϵ-greedy policy, in this paper. The algorithm follows the structure of MCTS: which is Selection, Expansion, Simulation, Backpropagation. Unlike UCT (Upper Confidence Bound applied to trees), the selection strategy for select the action state and the selection in the simulation phase is different. The UCT method always select highest value node, this value is calculated by UCB. However, since there is a large number of potential moves in the following games, the search range need to be reduced before using MCTS. This is achieved by using heuristic function. For improving the performance of the heuristic function. A computationally cheap greedy heuristic is proposed. The heuristic function is:

$$\sum_{i=1}^{4} p(q_i) - \sum_{i=1}^{4} p(o_i) \tag{2}$$

This function calculates the number of legal moves of a queen, qi stands for player's queen, and oi stands for opponent's queen. For a given game state. All possible child states of a player from a given state is calculated using the heuristic function 1. For each queen of a player, queen's possible state is sorted by the value of greedy heuristic function. The highest 12 states of each queen (increase as game goes on) is then selected, result in 48 states of possible chooses. Those 48 states are then added to a simulation pool. Which state is chosen as the final action state is based on their number of winning in the simulation phase in MCTS algorithm.

3.1. Epsilon-Greedy method

ϵ-greed is a commonly used strategy when weighing between exploitation and exploration. A small positive number $\epsilon(< 1)$ representing the probability is used to select unknown action in a random manner, leaving 1 - ϵ probability of selecting the action with highest value, which is defined as a greedy action. Assume that $s_t \in S$ is the current state and A is action sets that could be selected. After conducting an action $a_t \in A$ the agent will meet the next states s_{t+1} and the corresponding gain will be r_t. In the decision process, there is ϵ probability of choosing any non-greedy action, each action is then chosen with probability p_j which will be defined later in equation (4); that is, each action has the p_j probability of being chosen non greedily. There is also $1 - \epsilon$ probability that a greedy action could be selected, so the $1 - \epsilon + \epsilon \cdot p$ represents the likelihood that a greedy action could be selected. Among all the sets of actions A, there is always one action that is considered optimal by the agent at a certain moment. Let A^* be *arg max*$_a Q(S,a)$ and $\pi(a|s) = Pr\{a_t = a|s_t = s\}$, then

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon + \epsilon \cdot p, & \text{if a } = A^* \\ \epsilon \cdot p, & \text{otherwise} \end{cases} \tag{3}$$

3.2. MCTS with ϵ-Greedy

3.2.1. Selection. The node with the largest Q value will be first selected by the greedy policy with probability of ϵ. If the node with highest Q value is not selected, then the selection will be based on the q value sorted in the node. The probability in nodes follow a uniform distribution at the beginning, with each node having identical q-vaule Q(s_i) = 1, Which is then updated through backpropagation. The following probability is used to select node in non-greedy policy

$$p(s_j) = Q(s_j)\left(\sum_{i=1}^{n} Q(s_i)(1 - \delta_{ij})\right)^{-1} \tag{4}$$

where δ stands for the Kronecker delta function. However, as simulation goes on. probability of choosing a node is updated by Backpropagation according to the result of terminal state.

3.2.2. Expansion. Once an unvisited state s_i is achieved, it will be expanded. The value of heuristic function 2 will first be calculated for all child states of s_i and then they will be sorted. Only the largest 48 states will be added as child states of s_i.

3.2.3. Simulation. For each node in the simulation pool, a simulation is conducted by choosing moves based on its probability by formula 4 until a terminal state is met.

3.2.4. Backpropagation. In one iteration of simulation, once a terminal state is met, the parent nodes in the tree need to be updated. Backpropagation process is then begun, which propagates backwards from the new node to the root node. If win is the terminal state of a new node, the probability of choosing that node will increase in further simulation. The new action value is updated by using the following formula:

$$Q'(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\gamma^d \tag{5}$$

where Q is the new action value, is a number in range (0,1] and d is the "reverse depth" of a node counting from 0 to the root node (terminal state's depth is 0). γ is a decaying factor that controls how further the game result propagate. The final implemented algorithm is summarized in Algorithm 3.

Algorithm 3 MCTS2

```
Input: state $s_i$
$s_{current} \leftarrow s_0$
repeat
  if $s_{current}$ is a leaf node then
    Sort the value of formula 2 for all possible actions
    Add the highest 48 actions to $s_{current}$ as its leaf nodes
    $s_{current} \leftarrow$ first child node of $s_{current}$
  end if
  Generate a random number $r$ between 0 and 1
  if $\epsilon > r$ then
    $s_{current} \leftarrow argmax_{s_i}(p(s_i))$
  else
    $s_{current} \leftarrow$ Sample 1 state by probability defined in formula 4
  end if
  Backpropagate Q by formula 5
until $s_{current}$ is a terminal state
```

4. Comparison to classical search algorithm

The MCTS algorithm that have defined (as shown in Algorithm 3) is tested against several agencies, namely MCTS with UCB, alpha-beta pruning, minmax, depth first search and breadth first search. Because of the lack of the Amazons game protocol, the validation of the effectiveness of the improvements are conducted through the self-play games of the standard program. The MCTS with UCT is implemented as shown in algorithm 1. Each agency is put against each other for 100 rounds. Each agency is given identical calculation time of 30 seconds per turn. After the terminal state of game is reached, the result of win/lose is than recorded. The results are demonstrated in Table 1. The experiments are conducted on a i7-1280P CPU with 8G RAM.

Table 1. Results comparison.

	Win(MCTS)	Lose
MinMax	92	8
DFS	100	0
BFS	100	0
MCTS(UCT)	62	38
α-β pruning	82	18

Noted that to reduce the branching factor, for each of 4 algorithms only top 48 nodes ranked by formula 2 is pre-selected as possible action.

5. Conclusion

In this paper Monte Carlo Tree search with greedy is implemented. In this work the same heuristic function is used. The MCTS algorithm overall demonstrates its advantage in tree search with large branching. From the play out of games, MCTS obtained a higher win rate when competing with classical tree search method. The method implemented in this paper also has a slightly better performance than commonly used MCTS+UCB method, with 62%-win rate. Heuristic function is also an important factor of the algorithms' performance. In game of amazons, it is often hard to find a good move in early game, special crafted Heuristic functions or may be neutral network can be used together with MCTS to produce a stronger AI, and this method has been proven successful in the game of go.

References

[1] Buro, M. (2000). Simple Amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs. In International Conference on Computers and Games, 250-261.

[2] Hongxia, X., Yuan, X., & Guoyu, Z. (2013). A region-divided search algorithms for game Amazons. In 2013 25th Chinese Control and Decision Conference (CCDC), 933-938.

[3] Silver, D., Schrittwieser, J., Simonyan, K., et al. (2017). Mastering the game of go without human knowledge. nature, 550(7676), 354-359.

[4] Silver, D., Huang, A., Maddison, C. J., Guez, A., et al. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

[5] Kocsis, L., & Szepesvári, C. (2006). Discounted UCB. In 2nd PASCAL Challenges Workshop 2, 51-134.

[6] Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. Machine learning, 47(2), 235-256.

[7] Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. In European conference on machine learning, 282-293.

[8] Thrun, S., & Littman, M. L. (2000). Reinforcement learning: an introduction. AI Magazine, 21(1), 103-103.

[9] Chaslot, G. M. J., Winands, M. H., Herik, H. J. V. D., Uiterwijk, J. W., & Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. New Mathematics and Natural Computation, 4(03), 343-357.

[10] Rosin, C. D. (2011). Multi-armed bandits with episode context. Annals of Mathematics and Artificial Intelligence, 61(3), 203-230.