Monte Carlo Tree Search Based AI Bot For Board Games

Sabau Andrei-Mircea

Abstract

General video game playing is a challenging research area in which the goal is to find one algorithm that can play many games successfully. Classic approaches to game AI require either a high quality of domain knowledge, or a long time to generate effective AI behavior. These two characteristics hamper the goal of establishing challenging game AI. “Monte Carlo Tree Search” (MCTS) is a popular algorithm that has often been used for this purpose. It incrementally builds a search tree based on observed states after applying actions. However, the MCTS algorithm always plans over actions and does not incorporate any higher level planning, as one would expect from a human player.

Content

1.Introduction

Recent game programming research focusses on algorithms capable of solving several games with different types of objectives. A common approach is to use a tree search in order to select the best action for any given game state. In every new game state, the tree search is restarted until the game ends.

When implementing AI for computer games, the most important factor is the evaluation function that estimates the quality of a game state. The classic approach is to use heuristic domain knowledge to establish such estimates. However, building an adequate evaluation function based on heuristic knowledge for a non-terminal game state is a domain dependent and complex task. In the last few years, several Monte-Carlo based techniques emerged in the field of computer games. They have already been applied successfully to many games, including POKER and SCRABBLE. Monte-Carlo Tree Search (MCTS), a Monte-Carlo based technique that was first established in 2006, is implemented in top-rated GO programs. These programs defeated for the first time professional GO players on the 9×9 board. However, the technique is not specific to GO or classical board games, but can be generalized easily to modern board games or video games.

The purpose of this article is to show and demonstrate the applicability of MCTS in the board games domain as a powerful AI.

2. Monte Carlo Tree Search Algorithm

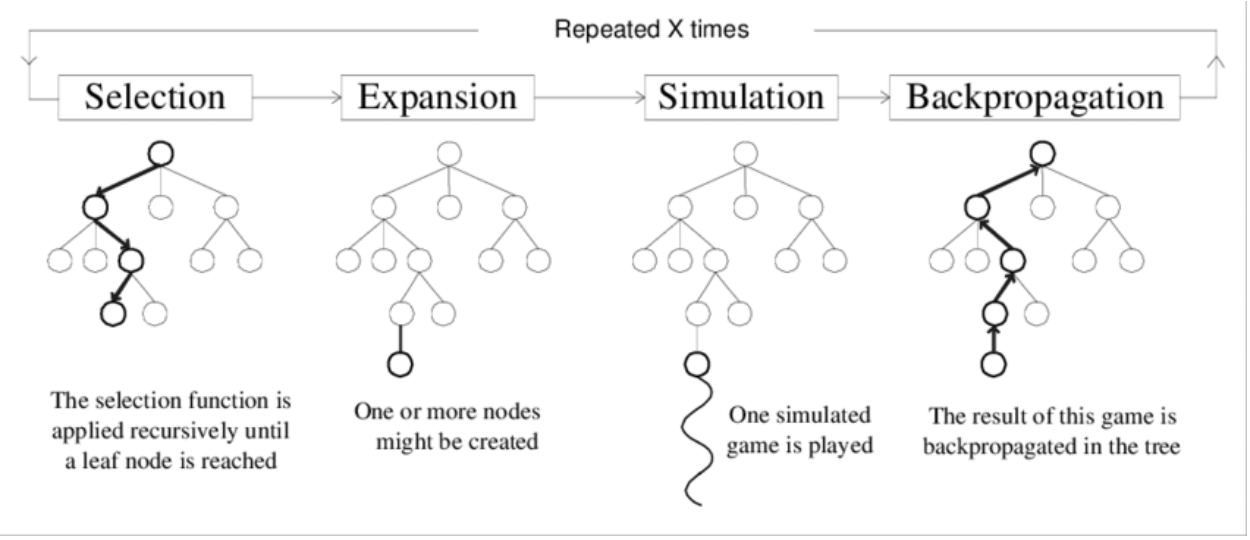

Figure 1 (the four steps of the MCTS algorithm)

Monte-Carlo Tree Search (MCTS), illustrated in Figure 1, is a best-first search technique. MCTS can be applied to any game of finite length.

The thing AI needs to do is to look at the current game and play simulations through all the potential next moves which could be played. AI does this by pretending that it played a given move and then continue playing the game until the end, alternating between the possible actions. While doing that, the AI is building up a game tree of all the possible moves the AI and the opponent (player in this case) would play. The algorithm builds and uses a tree of possible future game states, according to the following mechanism:

2.1 Selection

While the state is found in the tree, the next action is chosen according to the statistics stored, in a way that balances between exploitation and exploration. On the one hand, the task is often to select the game action that leads to the best results so far (exploitation). On the other hand, less promising actions still have to be explored, due to the uncertainty of the evaluation (exploration).

Generally speaking, there are 2 modes the AI can operate in when determining the next move to play:

Aggressive:

1. Play a move which will cause an immediate win (if possible)
2. Play a move which sets up a future winning situation

Defensive:

1. Play a move which prevents your opponent from winning in the next round (if possible)
2. Play a move which prevents your opponent from setting up a future winning situation in the next round

2.2 Expansion

When the game reaches the first state that cannot be found in the tree, the state is added as a new node. This way, the tree is expanded by one node for each simulated game.

2.3 Simulation

For the rest of the game, actions are selected at random until the end of the game. Naturally, the adequate weighting of action selection probabilities has a significant effect on the level of play. If all legal actions are selected with equal probability, then the strategy played is often weak, and the level of the Monte-Carlo program is suboptimal. We can use heuristic knowledge to give larger weights to actions that look more promising.

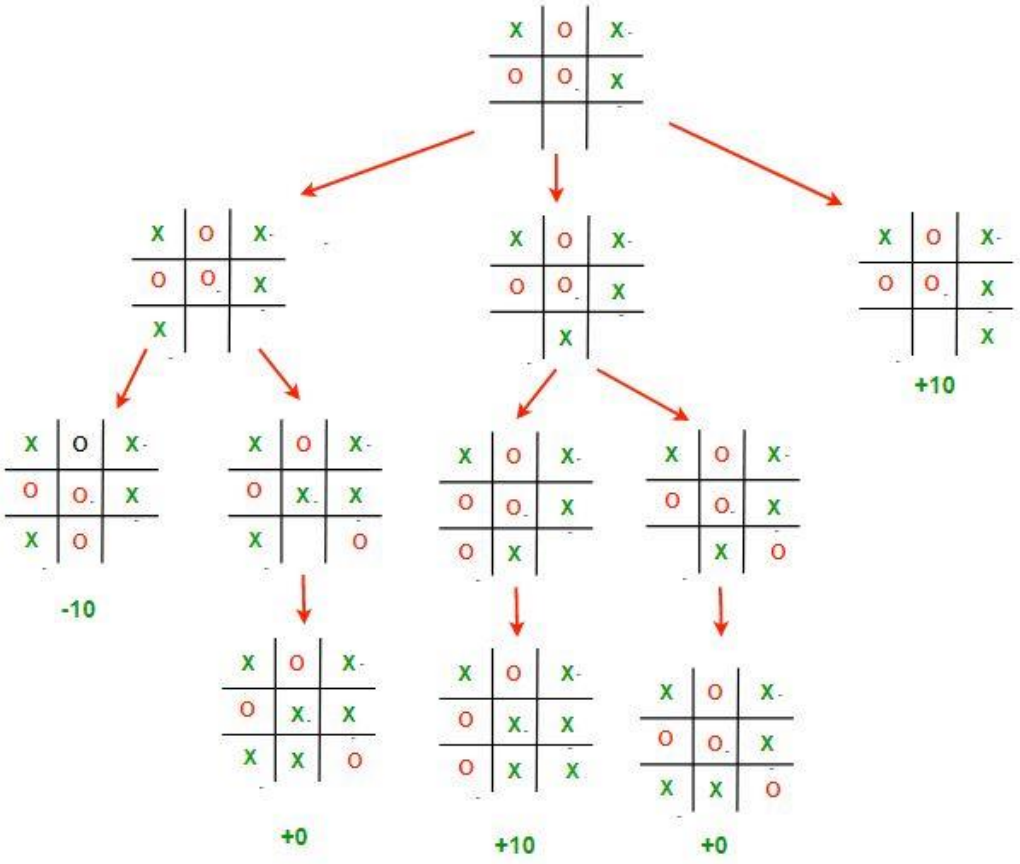

Figure 2. (MCTS applied on tic tac toe)

Above, there is an example of tic tac toe simulation. Each leaf of the tree is either a win, a loss or a draw. If the leaf has a win configuration, the fitness function awards the player with +10 points, if there is a loss, the player gets -10 points and in the case of draw, there are no points to be received.

2.4 Backpropagation

After reaching the end of the simulated game, we update each tree node that was traversed during that game. The visit counts are increased and the win/loss ratio is modified according to the outcome. The game action finally executed by the program in the actual game, is the one corresponding to the child which was explored the most.

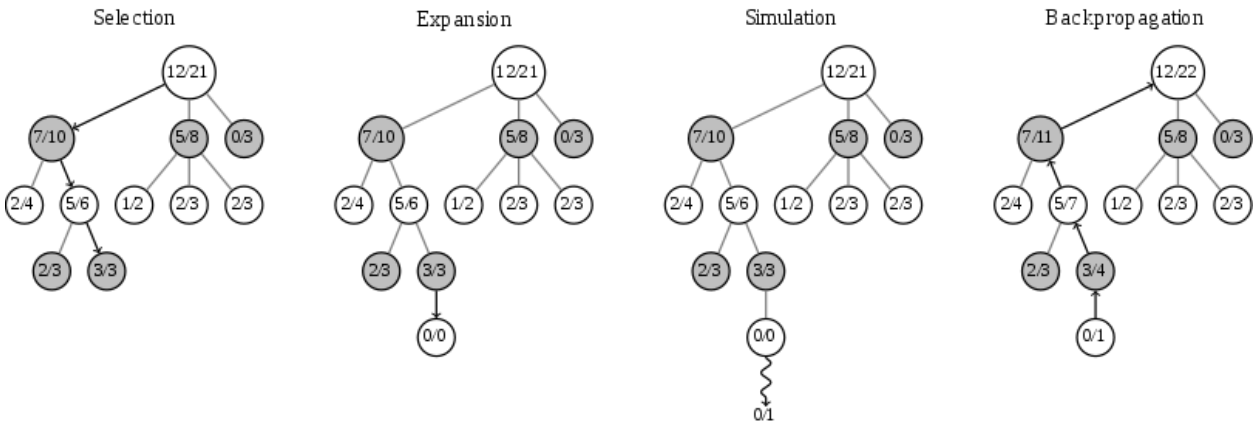

Figure 3. (An example of complete MCTS flow containing each of the four operations)

2.5 Pseudocode

What do we mean by a rollout? Until we reach the leaf node, we randomly choose an action at each step and simulate this action to receive an average reward when the game is over.

```
Loop Forever:

if Si is a terminal state:

   return Value(Si)

Ai = random(available_actions(Si))

Si = Simulate(Si, Ai)

This loop will run forever until you reach a terminal state.
```

Figure 4. (MCTS algorithm pseudocode)

2.6 Search space reduction with pruning

MCTS is a simple and elegant tree search algorithm. Given enough compute resources it will always find the optimal next move to play.
But there's a problem. While this algorithm works flawlessly with simplistic games such as Tic-Tac-Toe, it's computationally hard to implement it for strategically more involved games such as Chess. The reason for this is the so-called tree branching factor.

The difference of the complexity between Tic-Tac-Toe and Chess is huge. While the first game is played on a 3x3 table and the players have 2 different type of moves, the Chess is played on an 8x8 table with a lots of different moves.

Considering that we simulate a dumb move for X on the Tic-Tac-Toe table, in the worst case, at the first level of the tree, we will simulate 9 moves. On the other hand, if we take each piece of the Chess table and simulate the first level for each of them, we will use a lot of memory and computation time to simulate that moves. Due to the limitations of the computer memory and speed, the algorithm must reduce the stupid moves, the moves that are unnecessary. This is what pruning does. A simple example: consider a Tic-Tac-Toe game. It would be worthless to continue to simulate another move if at the current level in the tree, the game is already won by X or 0. Due to this fact, we will firstly check if at the current level the game can continue on the current node and only after that will expand the node in different new simulations and will consider the next move.

2.7 Advantages and disadvantages

Advantages

Aheuristic - MCTS does not require any knowledge about the given domain. Apart from the game rules and conditions the does not need to know any strategy or tactics, which makes it reusable for other games with small modifications.

Asymmetric - MCTS visits more interesting nodes more often, and focusses its search time in more relevant parts of the tree.

Anytime - It is possible to stop the algorithm at any time to return the current best estimate.

Elegant - The algorithm is not complex and is easy to implement.

Disadvantages

Playing Strength - MCTS, in its basic form, can fail to find reasonable moves for games even of medium complexity within a reasonable amount of time.

Speed - MCTS search can take many iterations to converge to a good solution, which can be an issue for more general applications that are difficult to optimize.

3. Applications of Monte Carlo Tree Search

Classic board-games, i.e., two-player deterministic games with perfect information, have been submitted to intensive AI researched. Excellent results have been achieved in the game of CHESS and CHECKERS. Initially, the use of randomized simulations in classic board games was criticized by researchers. However, it was later shown that MCTS is able to use highly randomized and weakly simulated games in order to build the most powerful GO-programs to date. It is noticeable that the best programs were built by people who only knew the rules of GO, and were not able to play the game at a strong level themselves.

Another well-known nowadays board game that shows incredible results when implemented with MCTS is the game SETTLERS OF CATAN. Modern board-games are of particular interest to AI researchers because they provide a direct link between classic (two-player, perfect information) board-games and video games.

4. Conclusions

The Monte Carlo tree search is a powerful tool in order to create a strong AI. Being able to search the tree for different types of strategies and to reduce the search domain with pruning, the only remaining problem is to find a good fitness function to help in creating a reliable strategy. I consider this a dependent algorithm due to that fitness function which in some cases is hard to implement.

Finally, if the tree search operates on a finite game environment and there is well defined method to win (fitness function), Monte Carlo tree search can lead to amazing results.

5. Bibliography

https://www.chessprogramming.org/Monte-Carlo_Tree_Search

https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/

https://www.researchgate.net/publication/220978338_Monte-Carlo_Tree_Search_A_New_Framework_for_Game_AI

https://towardsdatascience.com/game-ais-with-minimax-and-monte-carlo-tree-search-af2a177361b0

http://sander.landofsand.com/publications/bakkes_cig2016_paper_10.pdf

https://towardsdatascience.com/monte-carlo-tree-search-in-reinforcement-learning-b97d3e743d0f