Bandit Based Methods for Solving Ultimate Tic Tac Toe

Jeffrey Zhang

Advisor: James Aspnes

Yale University
Acknowledgements

Many thanks to my advisor, James Aspnes, for his support and guidance throughout the process of this project. I would also like to thank my parents, for always supporting me throughout my life and encouraging me to do my best, and my brother, for being with me every step of the way.
Abstract

A game in economics is an interaction between two or more players, defined by a set of states, transitions, strategies, and rewards. Every player seeks to maximize their own utility, often taking into consideration the possible actions made by other players, often also seeking to maximize their own utilities.

Certain classes of games are such that at least one player can guarantee a win or a draw against the other. The specific class of games is sequential zero sum games of perfect information and recall without nature. These games are theoretically solvable by backwards induction on the game tree, where solving in this context refers to finding a strategy in which one of the two players can force either a win or a draw. Tic Tac Toe for example has been solved to end in a draw. However, many games of this type are complicated enough that such a solution cannot feasibly be found. Most common and studied games of this type include Hex, Go, and Chess.

Solutions to these games are often approximated by randomized approaches. Bandits in the literature have been used to optimize rewards given Markov Decision Problems, a class of problems to which games can be reduced. The algorithm which uses these bandits, Monte Carlo Tree Search, has been successful in defeating top human professionals at Hex and Go.

I implement a bandit based method to Ultimate Tic Tac Toe, and find evidence that such methods are indeed effective for this game as well, though I am unable to achieve an actual solution for the game.
Bandit Based Methods for Solving Ultimate Tic-Tac-Toe

1. Introduction

A game in economics is an interaction between two or more players, defined by a set of rules, actions, and outcomes. Every agent seeks to maximize their own utility, taking into consideration other players seeking to maximize their own utilities as well.

There is a specific class of games for which there exists a definite optimal solution in which one of the two players can either win or force a draw. This class is sequential zero-sum games of perfect information and recall, without nature. Simply, it is a win-or-lose game in which players move one after another.

There are many games which satisfy these conditions, with a trivial example being tic-tac-toe and other notable examples being chess and Go.

A game following these conditions can be modeled as a tree, with nodes being decision points for each player and branches representing moves. Backwards induction will guarantee a solution as above.

Given this fact, in theory games like Chess are always solvable, though in reality this is often infeasible. With backwards induction, the full game tree is made, and optimal decisions are chosen for each node starting from the terminal nodes, up the tree recursively until the root is reached. Due to the large, or in chess’ case, infinite, number of branching possibilities in these games, this is practically impossible.

Because of these constraints, many randomized algorithms have emerged. These methods use a top down approach, approximating the win probability of each option and choosing the best one. This is as opposed to the bottom up approach used in backwards induction. The problem then arises of how to choose options based on incomplete information. In this paper I focus on Monte Carlo tree search (MCTS) using bandits.

Markovian Decision Problems (MDP)

This algorithm was one developed as a solution to Markovian Decision Problems. A MDP is described by a set of states, actions available to states, a transition from states to states given actions, and rewards given actions taken. The goal is to maximize a reward. Given the
similarity in definitions of games and MDPs, it is natural to model the former as one of the latter. MDPs are often modeled and solved as bandit problems.

**Bandit Problems**

A bandit problem is a decision problem in which a player must optimize his reward given a set of $K$ actions, each of which has its own reward distribution. The distributions are unknown, and the only information available is past results. In picking an arm, the player must make a tradeoff between choosing a consistent option ("exploitation") or testing less consistent options to see if they are in fact more profitable ("exploration"). The player must guess which action, or "arm", gives the most consistent optimal payoff, or alternatively, minimize "regret", the expected difference between the realized payoff and the optimal payoff. It is given by:

$$ R = \mu^* n - \sum_{j=1}^{K} \mu_j E[T_j(n)] $$

Where $\mu^*$ is the highest expected reward, and $T_j(n)$ is the number of times arm $j$ is played in the first $n$ trials.

Lai and Robbins\(^1\) showed that no policy exists with regret growing slower than $O(\ln n)$, and Auer et al.\(^2\) proposed a policy which has an expected logarithmic growth of regret. This policy, called the Upper Confidence Bound (UCB), estimates the upper confidence bound that any arm will be optimal, dictates that the player at any turn chooses the arm which maximizes:

$$ UCB1 = \bar{X}_j + \frac{2 \ln n}{n_j} $$

Where $\bar{X}_j$ is the average reward from arm $j$, $n_j$ is the number of times the arm was played, and $n$ is the number of total trials. It can be seen that this policy balances exploitation and exploration: the $\bar{X}_j$ rewards arms which have had consistently good results, while $\sqrt{\frac{2 \ln n}{n_j}}$ is bigger for less explored arms.

---


Upper Confidence Bound on Trees (UCT)

The UCT algorithm was developed by Kocsis and Szepesvari\textsuperscript{3} as a procedure to find near optimal solutions to MDPs. This was done by applying the UCB algorithm to determine the traversal of a tree, where every decision in a Markovian Decision process was modeled as a bandit. As an extension to games, each player’s turn is represented by a bandit, and each possible move is an arm of the bandit. The player’s goal is to pick the arm which maximizes his chances of winning.

The resulting algorithm is as such:

![Figure 1. Visualization of the MCTS procedure. From Browne et al.\textsuperscript{4}](image)

There are four steps taken in each of the iterations of MCTS:

1) Selection. In this step the tree is traversed with each step maximizing UCB1 for the given node. This stops when at least one child node is unexplored.

2) Expansion: The unexplored node is added to the tree.

3) Simulation: A random game is played from the node which was just added to the tree.

4) Backpropagation. In this step all the newly added node and all its parent nodes are updated to incorporate the results of the simulation.

It can also be seen that the general idea is to “think several steps ahead”.


The game of ultimate tic tac toe is an extension of normal tic tac toe. The board large is a tic tac toe board, each containing a smaller board. Winning a smaller game results gives the player the corresponding square in the larger game. The goal is to win the larger game.

The first player may choose any tile to play on, and for succeeding moves the player must play in the large square that corresponds to the position of the previous move. For example, if one player chooses the middle square of a small board, the next player must play in the middle square of the larger board. The exceptions are if that square has already resulted in a win or draw, in which case the next player can choose any square in any unclaimed board.

This game satisfies the conditions for being a sequential zero-sum game of perfect information and recall, without nature, and so it is theoretically solvable.

### Directed Acyclic Graphs

Instead of implementing Monte Carlo Tree Search, I chose to implement Monte Carlo Searching on Directed Acyclic Graphs.

For our purposes, Directed Acyclic Graphs differ from trees in that they can have more than one parent. This would actually be a more logical way to portray games, as for any given board position, there may be multiple ways to achieve it. This reveals some inefficiency in

---

standard MCTS – searches for transpositions must be repeated for every different history of the board configuration. Saffidine et al.\textsuperscript{6} found that their unoptimized implementations of DAG searching beat optimized MCTS methods.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{dag_example.png}
\caption{The example of a DAG is from Saffidine et al. The $n$ represent the number of times a node is reached and $\mu$ represent winrates. As can be seen, node b is preferable to node c. Had this game been modeled as a normal tree, the decision made at a would have been to choose c over b.}
\end{figure}

Instead of having a tree node for every possible history, a DAG will have a node for each transposition, with hash tables to keep track of nodes instead of strict parent-child relationships for each node. The MCTS algorithm remains mostly the same, though in my implementation, every node is updated as it is traversed to in the tree. Each node will compile data from each of its children, and use an updated total in its calculation of each UCB value.

**Results**

To test my program I examined the effects of varying iteration count on the performance of the DAG search.

In the first set of experiments, I had two separate instances of my DAG search, with different numbers of iterations used for each step of the DAG search. The effects were as one may expect: The player with the larger iteration count was favored, with the favored player having approximately a 60% winrate with twice the iterations, and 75% with four times the iterations. The results are shown in the table below. Iterations for player X are on vertical axis,\nn

---

iterations for player O are on the horizontal axis. Values presented are win/loss/draw for player X out of 400 games.

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>4000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>170/184/46</td>
<td>115/222/63</td>
<td>44/299/57</td>
</tr>
<tr>
<td>2000</td>
<td>242/96/62</td>
<td>184/151/65</td>
<td>120/235/45</td>
</tr>
<tr>
<td>4000</td>
<td>324/33/43</td>
<td>241/99/60</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Fig 4. Results of 2 UCTs playing against each other.

I also examined the first moves made by the X player to attempt to see some convergence in the opening strategy, and see whether it depended on the number of iterations per execution of MCTS. There did not seem to be a pattern, and both tests and the combined data failed a chi square goodness of fit test against the uniform distribution (p = .638 for 1000 iterations, p = .624 for 2000 iterations, p = .423 for combined). It is worth noting that while there does exist some strategy to secure a win or draw for one of the players, it may not necessarily be unique. See Fig. 5 for frequencies.

Fig 5. Combined frequency of first moves by square. See Appendix for raw data.
Conclusion

There is evidence that DAG searches are an effective way of playing Ultimate Tic Tac Toe. If there was no effect, we would expect games where one player should have been favored to have approximately a 50% winrate.

However, there is also evidence that DAG searches are not effective for computing a game theoretic solution to Ultimate Tic Tac Toe. Whether this is because DAG is not effective for this purpose or whether it is because the solution is not unique is a topic of further study.

As a final note, I point out that the effectiveness bandit based methods are not necessarily restricted to the class of games I pointed out. As long the game is of perfect information and sequential, therefore reflecting a tree, the algorithm should be able to be modified. Games with randomness simply have different reward distributions on choices, for example.

Games which probably cannot be solved this way include games like poker, where the same actions probably do not have similar priors, and there is a significant psychological component.

References


Fig 6. Frequency distribution of first moves with 1000 iterations of MCTS per player. The legal first moves were restricted to squares 18 to 44, as any other move is equivalent to one of these through reflection or rotation.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>42</td>
<td>42</td>
<td>47</td>
<td>40</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>31</td>
<td>35</td>
<td>23</td>
<td>44</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>31</td>
<td>36</td>
<td>39</td>
<td>31</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

Fig 7. Frequency distribution of first moves with 2000 iterations of MCTS per player.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>41</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>26</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig 8. Combined frequencies for 1000 and 2000 iterations.