Abstract

The most intuitively exhaustive and conventional form of artificial intelligence (AI) for board games is minimax. Minimax involves constructing the complete game tree of possible moves from the current state of a game and choosing the branch that leads to most positive terminal results. Although this may be a very thorough form of AI, for even moderately complex games the game tree is much too immense to be able to iterate through the complete game tree efficiently.

Thus, the solution to this is the Monte Carlo Tree Search Algorithm (MCTS), a heuristic algorithm, which uses only knowledge of the possible moves and the Monte Carlo method to selectively sample branches of the complete game tree with many playouts and reach conclusions based on these samples. The game which MCTS was applied to for this project was Connect Four, because its complete game tree is much too massive for minimax to be a viable option. The goal of this project was to program an efficient artificial intelligence, implementing the Monte Carlo Tree Search Algorithm, to make decisions on a turn-by-turn basis for the game Connect Four (given only the knowledge of the possible moves).

First, using an object-oriented approach in Java, a game framework for Connect Four was created with an established domain of moves. Then, MCTS was implemented in four parts: *selection* of nodes, *expansion* of children, *simulation* of game playouts, and *backpropagation* to update node stats. Repeating these four steps for a predetermined number of times gave rise to knowledge that allowed the AI to make a decision.

Data was collected in order to determine the effectiveness and efficiency of the MCTS implementation, including many trials of AI vs AI playouts, timing the speed of the AI, comparing the different MCTS play-styles, and comparing MCTS to the e-greedy and minimax selection policies.

Ultimately, it was found that the MCTS implementation was indeed efficient and showed a high level of success in both efficiency and effectiveness at different levels of playouts for the game Connect Four. Future implications include giving the AI further domain knowledge to account for sampling error, continuing to increase its efficiency, and fully optimizing the playstyle and number of playouts. Using this knowledge of MCTS and the frameworks of this implementation, it can be applied to a vast array of games as well as more practical applications such as amino and nucleic acid polymerizations in bioinformatics.

Connect Four

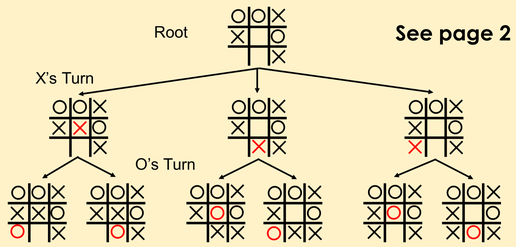
- Classic two-player logic-based game
- Goal = get four of own color in row
- 4,531,985,219,092 possible board permutations
- Currently estimated ~2 billion terminal board permutations
- **Tree too large for computer to find efficiently with minimax**

See page 3

Goal/Objective

Program an efficient artificial intelligence, implementing the Monte Carlo Tree Search Algorithm, to make decisions on a turn-by-turn basis for the game Connect Four

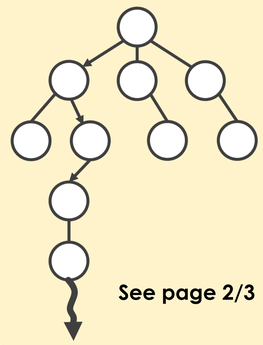
The Minimax Game Tree



See page 2

Minimax creates a game tree for each possible move sequence and chooses the branch that minimizes losses while maximizing wins. However, even moderately complex games yield enormous, inefficient game trees.

Monte Carlo Tree Search



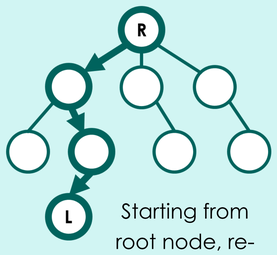
See page 2/3

- Monte Carlo Method makes conclusions based on statistical samples
- MCTS samples game branches asymmetrically to reach conclusions
- Requires much less memory/time to sample incomplete tree

Choosing the Move

- max child: select the root child with the highest reward
- robust child: select the most visited root child
- **max-robust child**: select the root child with both the highest visit count and the highest reward

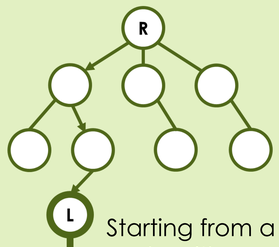
Selection



Starting from root node, recursively select child nodes until a node has no previously visited children

See page 6

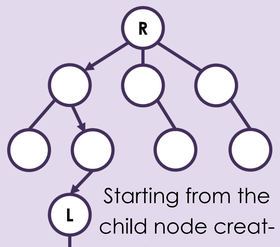
Expansion



Starting from a node with previously unvisited children, create children and choose one

See page 8

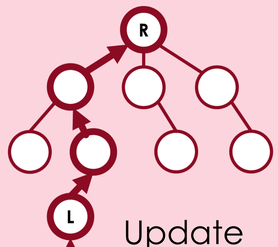
Simulation



Starting from the child node created from expansion, simulate a random game until some end result (win, loss, tie) is reached

See page 9

Backpropagation



Update each visited node with result

See page 10

Selection: Bandit Policy (UCB/UCT)

- Balances **exploration and exploitation**
  - UCB = Upper Confidence Bound (Bandit)
  - UCT = Upper Confidence Trees (MCTS)
- $\bar{X}_j = average\ reward\ from\ arm\ j\ (value \div total\ visits)$
- $C_p = tree\ exploration\ constant\ \left(generally\ \frac{1}{\sqrt{2}}\right)$
- n = number of visits to the parent node
- n_j = number of visits to child node

$$UCT = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

See page 7

Sample Output

- Xs and Os denote the player pieces
- First column → value of particular column branch
- Second column → number of visits to particular branch
- Chosen branch

```
Player O's Turn
0: 76.0 228.0
1: 136.5 341.0
2: 100.5 275.0
3: 90.0 255.0
4: 84.5 244.0
5: 216.5 483.0
6: 49.5 174.0
5
```

```
| - | - | - | - | - | - | - |
-----------------------------
| - | - | - | - | - | - | - |
-----------------------------
| - | - | - | x | - | - | - |
-----------------------------
| - | - | - | o | x | o | - |
-----------------------------
| - | - | x | x | o | o | - |
-----------------------------
| - | x | o | o | o | x | - |
-----------------------------
```

Discussion

First, the MCTS AI vs AI playouts yielded some interesting implications (Figure 1). As expected, those MCTS AIs with a higher number of playout iterations performed much better than those with lesser playout iterations. This is evident because the AI with 5000 iterations received 197 more wins than the 1000 iteration AI and 650 more wins than the 500 iteration AI when played against one another (both matchups had a negligible number of ties). Also, when AIs of the same playout iterations were played, they had equal success. Thus, it is apparent that a greater number of playout iterations means a more successful MCTS AI.

However, it seems that the relationship between AI success and playout iterations is not linear. This is so because although the 5000 playout AI had 4000 more iterations than the 1000 playout AI, it did not do substantially better (enough to propose a linear relationship). Therefore, it may be more appropriate to model this relationship with a logarithmic or inverse tangent model as the initial increase in playout iterations has a drastic effect on MCTS success (as evident between 1000 vs 500), but this increase in success slowly levels off logarithmically or asymptotically as it approaches a large amount of playout iterations.

Furthermore, as the number of playout iterations increases, so does the inefficiency and time required by the AI to make a move (Figure 2). This is illustrated by the amount of time it takes for an initial move to be made given the number of playout iterations of MCTS. As it is apparent in Figure 2, the relationship between playout iterations and the time elapsed for the first move is linear. Moreover, when this information is combined with the previously derived information that higher playouts numbers do not do substantially better than lower ones, it may be a likely conclusion that for the game Connect Four (and perhaps others) there is an optimum number of playout iterations of MCTS. This is so because it may be inefficient to always use the maximum number of playout iterations since the time elapsed increase linearly/indefinitely, yet its success only increases by a slight margin.

The E-greedy data shows that MCTS is indubitably superior to the simplistic e-greedy policy for the game Connect Four (and likely other games as other evidence shows). Even the most limited number of MCTS iterations (500) had no trouble in completely thwarting 5000 iterations of e-greedy at both values of n (See Page 14). This shows that the MCTS UCT Bandit-based selection policy is far superior in its efforts to balance exploration and exploitation. Additionally, although the minimax implementation was far too inefficient to test, extrapolating the results from Figure 2 to account for the 2 billion playouts (where y seconds x/1000 and x = 2 billion playouts) of minimax estimates the initial move would take approximately three weeks (~23 days), much longer than MCTS (See page 13).

Lastly, the backpropagation policy results illustrate various trends as a result of changes in policy (See page 14). For one, the balanced policy seemed to always end in matching the number of wins as the opposing policy, proving it to be consistently robust. To add, when two AIs with the same policy were played against one another, they always resulted in approximately the same number of wins. One unique feature is that when the two defensive strategies were played against one another, it almost always resulted in a tie. This may be useful knowledge given that Connect Four is a game in which when two players play 'perfectly' it is possible to always tie (Kocsis). Thus, it may be that the defensive strategy is the most optimal for 'perfect' play. To add, when the defensive was matched up against offensive it was the only strategy to come out with substantially more wins.

Successes

- Program is functional, AI is difficult to beat
- AI is fast (relative to minimax) and consistent
- Implementation is simple (doesn't need any external information)
- Asymmetric tree is efficient
- Follows expected patterns of MCTS

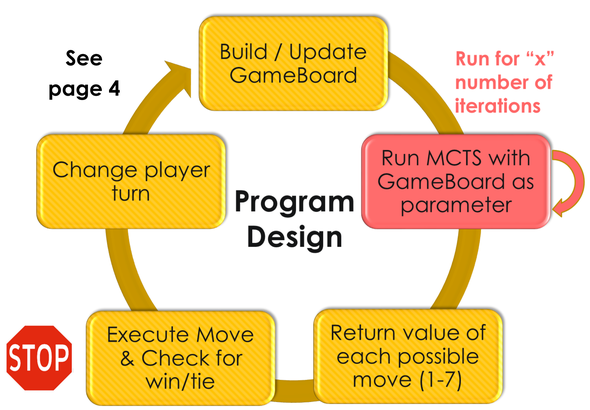
See page 16

Developing

- Can still be sped up/made more efficient
- Some reasonable moves apparent to the human player are overlooked due to magnitude of permutations and natural sampling error
- Domain Knowledge – filter out implausible moves or urge very plausible moves to increase asymmetry
- Optimize playstyle and playout iterations

Practical Applications

- Improve knowledge on artificial intelligence in general
- Improve real-time strategy board or video games' AI
- Simulate complex real life situations with perfect (or near perfect) information
  - Stock simulations
  - Robot decision making
  - Amino and Nucleic Acid polymerization simulation in bioinformatics



See page 4

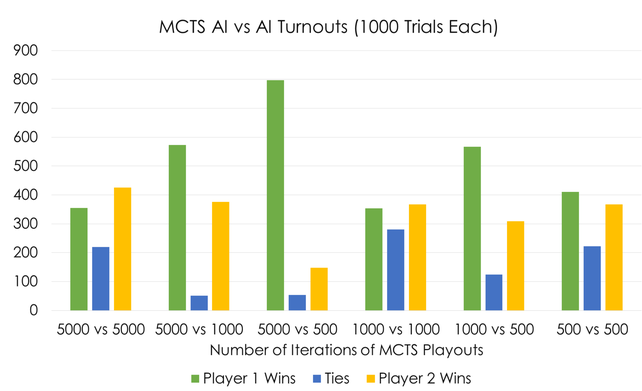


Figure 1: MCTS AI vs AI Turnouts (1000 Trials for Each Matchup) [See page 12 for table and page 15 for discussion]

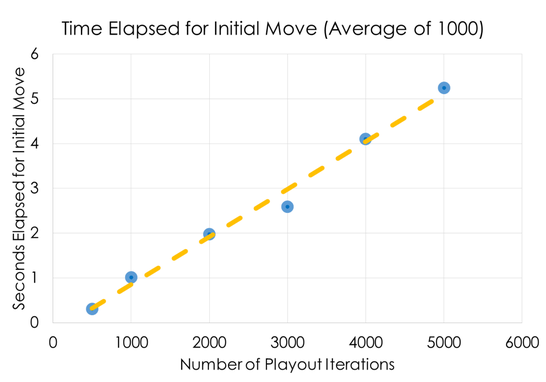


Figure 2: Time Elapsed for Initial Move [See page 13 for table and page 15 for discussion]