VNUHCM - UNIVERSITY OF SCIENCE

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHTN

ĐHQG-HCM

Lab 2: Adversarial Searching Algorithm

Student's name	: Phan Xuân Nam
Student's ID	: 20127247
Instructors	: Phạm Trọng Nghĩa, Nguyễn Thái Vũ
Subject	: Introduction to Artificial Intelligence

Contents

Lab's description

In this second lab, we implement different adversarial searching algorithms for a game called Tic Tac Toe with different board sizes: 3x3 and 5x5

Adversarial Search

"Adversarial" means to fight or oppose each other. In computer science, it's about planning ahead to find the optimal move for the user to win against the opponent and vice versa.

In most games like chess, go, or checkers, the game's characteristic is deterministic, meaning that each move follows a rule, and each player is allowed to see the other's move as well. Applying adversarial algorithms, we ensure to find the best move based on the current state. In some cases, it might give an unfathomable move which requires profound calculations.

Here in this lab, we are going to cover 2 algorithms and how it is integrated into Tic Tac Toe:

Minimax

Minimax is a backtracking algorithm where it assumes both players are playing optimally.

In this algorithm, we have two players: a maximizer (max) and a minimizer (min). Each player will try to move such that benefits them the most. The max one will have a positive infinity while the min one will have a negative infinity. The goal for both players is the same: trying to maximize/minimize the value based on the current state of the game.

Since Minimax is based on DFS to traverse all possible states, we ensure that with a board as big as 9x9, it will find all possible combinations, which are n! with n stands for the size of the board. One downside of this algorithm is the fact that with the size of n x n, the trees can be expanded up to b^d with:

- b: the branching factor (= n)
- d: the deep level of the tree ($d \leq n$)

Therefore, it is impractical to apply Minimax to a 4x4 board or above as the time expands exponentially and the level will be significantly deeper.

As mentioned above, Minimax is a DFS and backtracking algorithm to calculate the score based on the current state it derives from. Unearthing the process is fairly simple: Given a state, fill each empty slot with a sign and:

1. It will the rest of the slots with the remaining sign until it reaches its terminal/base case and returns a score based on the final case
2. Similar to the following slots, we return a score based on the terminal case. It will backtrack to return the max of a subtree, then the min of the subtree and so on. The process will stop when the current state is assessed with a valid value.

In terms of completeness, Minimax is based on DFS and hence definitely finds all solutions. Also, Minimax's only optimal when both players are playing optimally. If either player plays a weird move that the tree's depth hasn't reached, it will take more time to dig; therefore, consumes more time as it processes.

3x3 Board's assessment

A 3x3 board has 9 blanks so there are 9! combinations to fill the board. When implementing the Minimax algorithm, it takes 9^9 times to compute. Since the board is small and when we expand the tree, it's still shallow enough so that when the recursion starts, it doesn't have to store so many states at once.

5x5 Board's assessment

A 5x5 board has 25 blanks so there are 25! combinations. It is also noticeable that when using Minimax, the tree's depth grows exponentially and can not be calculated. The algorithm definitely finds a state but it will take a very long time, which scales n-fold with n being the size of the board. The branching factor is so huge that, after expanding a blank, it cannot return a value immediately but keep on expanding the rest of the blanks. This is a huge downside of Minimax as this is purely DFS's nature that Minimax has to accept.

Alpha-Beta Pruning

Alpha-Beta Pruning is a technique used in Minimax to cut down some unnecessary branches since there are better options than them. Essentially, it is still Minimax but an optimized version. The algorithm cannot guarantee to cut down every redundant branch but it does make the deepening take less time and perform faster to return the evaluation of the current state.

This technique requires 2 new parameters: **alpha** and **beta**. Depends on your choice that **alpha** can have a positive or negative infinity and the same applies to **beta**. But one thing should be taken into consideration: a maximizer can only make changes to the positive infinity and a minimizer can only make changes to the negative infinity.

Alpha-Beta Pruning can be put simply: when the tree reaches the terminal case, it returns a value and assigns to **alpha**, then assigns to **beta** of its parent's node. Then it continues to search for the right sub-tree of the parent. If the value we find when reaching the terminal case is bigger than **alpha** of the parent, the search stops and that branch is completely ignored. The same goes for the right side of the root. After we finish pruning the tree, we then perform the minimax, review and give a score based on the current state.

Should the trees are placed unordered, the search will be much slower due to the nature of DFS: only expand vertically, not horizontally. This is why the order of the state in the game is emphasized.

Another approach is to use heuristics. It's better to use but in Tic Tac Toe, heuristics are quite hard to measure and the game's state can vary with each game. One more thing to keep in mind: if players are making a move that the AI has to dive deeper, chances are it will consume more time to create a better move to counteract.

To conclude, the space and time complexity of this algorithm in the worst case is O(b^m) with b standing for the branching factor and m standing for the maximum depth of the tree. Completeness-wise only does it complete the traversal when the depth limit isn't set. Most of the time, we apply the depth limit property from IDS to ensure that it doesn't recur so many times that it creates a stack overflow.

3x3 Board's assessment

Space-wise, the state that AI has to judge is reduced a lot but in terms of time to process, there's no huge difference since the board itself is quite shallow. Nevertheless, optimization is a huge step up on this board.

5x5 Board's assessment

As I mentioned above, the tree is so big that it takes approximately 25^{25} (this could be more) times to go through every single state. With Alpha-Beta Pruning, the states that have to be judged are reduced by approximately one-fourth or one-fifth of all possible states. Still, it is undoubtedly inefficient since the depth is too big to go through all of them.

References

javatpoint. (n.d.). *Alpha-Beta Pruning*. Retrieved from javatpoint: https://www.javatpoint.com/ai-alpha-beta-pruning

javatpoint. (n.d.). *Mini-Max Algorithm in Artificial Intelligence*. Retrieved from javatpoint: https://www.javatpoint.com/mini-max-algorithm-in-ai

Demo

Link: https://youtu.be/ThBg55XIEuE