Multiplayer Chopsticks Game using Max^N Algorithm

Rajat Sharma (B18CSE043)

October 2020

Abstract

The project aims to formulate and solve the chopsticks game as an adversarial search problem. We used Max^N algorithm to perform adversarial search on the game tree in order to calculate the maximum utility. We also tried to optimize the vanilla Max^N algorithm by using evaluation function for limited nodes, and graph search and dynamic programming optimizations to speed up search and reuse calculated values.

1 Introduction

Chopsticks is a simple game played using fingers, usually amongst two players. The players try to "kill" both the opponents hands, which is done by transferring fingers on their hands to that of the opponents till it becomes 5 in a hand, at which the count of fingers rounds off to 0, and the hand is said to be "dead". Chopsticks is an example of a combinatorial game [Wik20b], and hence has a perfect play with optimal strategy from any point being known. Hence it can be formulated as an adversarial search problem, and solved using minimax algorithm [Wik20c] for two players, and extended using Max^N algorithm [LI86]for n players, which is what we seek to accomplish in this project.

1.1 Rules for Chopsticks

The rules, taken ad verbatim from Wikipedia [Wik20a] are:

1. Each player begins with one finger raised on each hand. After the first player, turns proceed clockwise.
2. On a player's turn, they must either attack or split, but not both.
3. To attack, a player uses one of their live hands to strike an opponent's live hand. The number of fingers on the opponent's struck hand will increase by the number of fingers on the hand used to strike.
4. To split, a player strikes their own two hands together, and transfers raised fingers from one hand to the other as desired. A move is not allowed to simply reverse one's own hands. You can not spilt 3 with a dead hand. If any hand of any player reaches exactly five fingers, then the hand is killed, and this is indicated by raising zero fingers (i.e. a closed fist).
5. A player may revive their own dead hand using a split, as long as they abide by the rules for splitting. However, players may not revive opponents' hands using an attack. Therefore, a player with two dead hands can no longer play and is eliminated from the game.

6. If any hand of any player reaches more than five fingers, then five fingers are subtracted from that hand. For instance, if a 4-finger hand strikes a 2-finger hand, for a total of 6 fingers, then 5 fingers are automatically subtracted, leaving 1 finger. Under alternate rules, when a hand reaches 5 fingers and above it is considered a "dead hand".

7. A player wins once all opponents are eliminated (by each having two dead hands at once).

1.2 Novelty and Constraints

The main novelty of the problem statement lies in the fact that since there doesn't exists any programmed version for tackling it as an n player game, hence there doesn't also exist any AI for the same. Thus our use of Max^N algorithm to solve it for n players is a novelty in this direction. Also we have formulated a new evaluation function for scoring the intermediate states, besides using dynamic programming optimizations to increase the depth of the search.

The main constraints that we faced in solving this problem is to increase the reachable depth. Since the time complexity of Max^N is B^M where B is the branching factor for the search state, and M is the depth, therefore allowable depth is limited since larger depths take a lot of time. Resolving this was an important issue that we faced.

2 Problem Definition

We intend to formulate our problem as an adversarial search problem, and we will be assuming that the opponents are rational and hence play optimally. The various elements for the search problem are as follows:

2.1 State

The state of the game at any point is defined by an $(N+1)$ tuple, where the first the N elements denote the state of the hands for each player, while the $(N+1)th$ element is the index of the current player. For each player, the state of hands is denoted by a pair of integers $< h_1, h_2 >$, where h_1 and h_2 are the number of fingers on each hand, and $0 \leq h_1, h_2 \leq 4$. Thus the state at any point is defined as

$$S = \{< h_{1,1}, h_{2,1} >, < h_{1,2}, h_{2,2} >, ..., < h_{1,n}, h_{2,n} >, p_{curr}\} \tag{1}$$

Any player is said to be alive if $h_1 + h_2 \neq 0$, else he is said to be dead. If all but one players are dead, then that state is said to be the goal state. A goal test for the same is defined by the evaluation function. The initial state of the game is defined as:

$$S_1 = \{< 1, 1 >_1, < 1, 1 >_2, ..., < 1, 1 >_n\} \tag{2}$$

2.2 Transitions

There are two kinds of state transitions for the game:

1. Attack: Every player can tap the opponent's live hand to increase the number of fingers on that hand by the number of fingers on the hand used to strike. The conditions for attack are as follows:

- The attacking player must be alive, i.e. at least one of his hands must contain a positive number of fingers.
- The hand used to attack must also be alive, i.e. it should have a non-zero number of fingers stretched out.
- The opponent must also be alive, and so should be the attacked hand.

Suppose player p_i using hand $h_{j,i}$ attacks player p_k on hand $h_{l,k}$, $1 \leq i, k \leq N$ and $1 \leq j, k \leq 2$, then after the attack, $h_{l,k} = (h_{l,k} + h_{j,i}) \mod 5$. Thus, taking $l = 1$ without loss of generality, the attack transition can be represented as:

$$T(S, \mathit{Attack}) = \{< h_{1,1}, h_{2,1} >, ..., < (h_{1,k} + h_{j,i}) \mod 5, h_{2,k} >, ..., < h_{1,n}, h_{2,n} >\} \tag{3}$$

2. Split: The player can transfer fingers from one of his hands to the other as desired by tapping them together. The split operation can be used to reduce the number of fingers on a hand, revive a hand, or to remove the total number of fingers by increasing the number of fingers on a hand beyond 5. The split operations has some constraints, which are listed as follows:

   - The splitting hand cannot transfer fingers greater than what it is holding, and the hand cannot be killed off.
   - The receiving hand cannot be killed off, although its count may increase beyond 5, after which its remainder is taken with 5.
   - The split cannot reverse the number of fingers on each hand, since that is considered as a wasteful move.

   Suppose player p_i splits $h_{1,i}$, transferring k fingers to hand $h_{2,i}$, $1 \leq k < h_{1,i}$. Then after the split, the status of the hands is $h'_{1,i} = h_{1,i} - k$, $h'_{2,i} = (h_{2,i} + k) \mod 5$ with $h'_{1,i} \neq h_{2,i}, h'_{2,i} \neq h_{1,i}$ and $h'_{1,i}, h'_{2,i} \neq 0$.

$$T(S, \mathit{Split}) = \{< h_{1,1}, h_{2,1} >, ..., < (h_{i,1} - k), (h_{2,i} + k) \mod 5 >, ..., < h_{1,n}, h_{2,n} >\} \tag{4}$$

2.3 Evaluation Function

Our evaluation function is a mixed one, since it provides expected utility for both intermediate states and goal states, and hence also functions as a goal test. The utility for a state is represented as a N tuple, where the i^{th} element is the sum of fingers on both hands. The basic intuition behind this formulation is that each player must try to extend his lifetime in the game, and this can be done by always having a non-zero number of fingers on his hands. Thus by maximising the number of fingers, we can maximize the players duration of play. If at any point, all but one players are dead, i.e. they have no fingers outstretched, then that state is said to be a goal state. Suppose only i^{th} player is alive, then his utility is modified to be 10, with all others having utility as 0. Thus the evaluation function is represented as:

$$E(S) = \{(h_{1,1} + h_{2,1}), ..., (h_{1,i} + h_{2,i}), ..., (h_{1,n} + h_{2,n})\} \tag{5}$$

If $(h_{1,i} + h_{2,i}) \neq 0$ and $(h_{1,j} + h_{2,j}) = 0 \forall j \neq i$, then $e_i = 10$ and $e_j = 0 \forall j \neq i$. Thus the utility for a goal state is given by:

$$E(S) = \{e_1 = 0, ..., e_i = 10, ..., e_n = 0\} \tag{6}$$

3 Background Survey

Our background survey revealed that there exist very few implementations of the game for the variant that we are working on, and all of them are for two players only, there being no multiplayer implementations for the same. Implementations of relaxed variants also exist, but they are also for two player mode only. Thus our implementation is novel in its scope, since it targets a more generalized game-play. Also, our evaluation functions and optimizations developed for this project are also original, to the best knowledge of the author.

4 Discussion

4.1 Non-Cooperative Games [Wik16]

In the most basic terms, a non-cooperative game is a game where all players are independent of each other, and their decisions are based on maximising their utility by competition, instead of coalition. Players in a non-cooperative game cannot form alliances, where two or more players work together to maximise payoff, and instead have to work independently. Alliances may appear and disappear throughout the game play, but these are based more on maximising individual payoff, instead of any enforced rules based alliances.
Chopsticks qualifies all the requirements to be classified as a non-cooperative game, and hence we can classify it as such and move on towards the next section.

4.2 A Perfect Information Game

A sequential game is said to have perfect information if the following conditions are satisfied [Wik20d]:

1. All players know the game structure.
2. Each player, when making any decision, is perfectly informed of all the events that have previously occurred.

For chopsticks, the current game state, the transition function, and the evaluation function is known to each player, hence we can say that the game structure is known to each player. In case of the second condition, we can modify the state representation to contain the collection of moves previously played also. Suppose there have been N moves of the game till now, then we can redefine the state as:

$$H_n = H_{n-1} \cup S_n \tag{7}$$

$$S'_n = \{S_n, H_n\} \tag{8}$$

where S_n is the state of the game in the N^{th} move, as defined in (1), H_n denotes the history of the game after n moves, and S'_n is the modified state representation, with $H_0 = \phi$. Thus the game state now also contains information about the previous events, and hence the second condition is also satisfied, and therefore chopsticks is a non-cooperative, perfect information game. By the following theorem, non-cooperative, perfect information games are solvable, and hence we can move forward to finding one.

Theorem 1 *A finite n person non-cooperative game which has perfect information possesses an equilibrium point in pure strategies (proof in [Hud80], page 63).*

Without going into much mathematical detail, the equilibrium point is a point where no player can increase his her utility by changing his strategy, where a strategy is a function that maps a move (to be taken by the respective player), to a given state. Thus for each player, his choice of move is fixed for a given game state, and hence that is considered as a solution, i.e. suppose a player has to make a move and the current game state is S, then only by transitioning to state S', can the player maximise his utility. Any other successor state cannot result in an increase in utility provided the other players don't change their respective strategies.
For each state, we can apply the transition function to consider all possible transitions and hence produce a game tree for the chopsticks game. Thus we can use the following theorem to find a solution for the game:

Theorem 2 *Given an n person, non-cooperative, perfect information game in tree form, max^n finds an equilibrium point for the game (proof in [LI86] page 159).*

Thus now we can use the Max^n algorithm to find a solution for the game.

4.3 Max^N Algorithm

The Max^N algorithm procedure is quite simple. At every turn, each player tries to maximise his own utility, with the utility being calculated by the evaluation function. Thus by theorem 2, we can find a solution (equilibrium point) for the game.

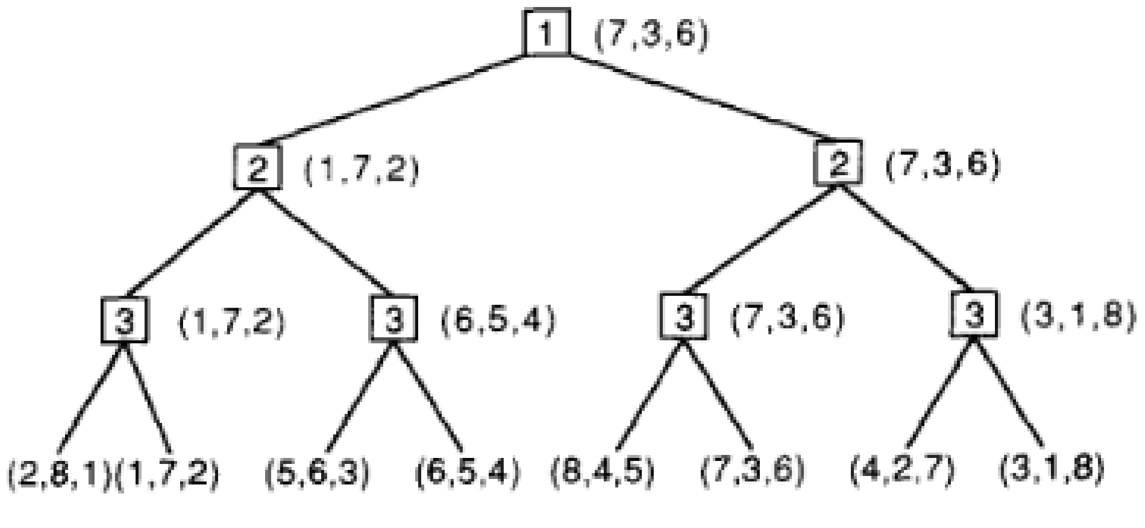

Figure 1: A sample 3 player Max^N game tree [Kor91]

Procedure: MaxN(State)

```
1. If $State$ is Terminal:
2:     return $Eval(S)$
3. endif
4. Initialize $curr \leftarrow State_{n+1}$
5. Initialize $successor \leftarrow T(State, Attack) \cup T(State, Split)$
6: Initialize $mx \leftarrow -\infty$
7. Initialize $util \leftarrow NULL$
```

```
8: For $v$ in $successor$, do:
9:        $move_{curr} \leftarrow MaxN(v)$
10:        If $move_{curr} > mx$
11:              $mx \leftarrow move_{curr}$
12:              $util \leftarrow move$
13.        endif
14: endfor
15. return $util$
```

The *State* used in the procedure is defined analogously to the one defined in (1), while $Eval()$ function used is defined as given in (5) and (6). The transitions function $T()$ returns a set of all possible and valid transitions for the given state. The procedure returns a utility tuple, which is then passed to the calling function. Thus the game tree is traversed recursively, and a move is fixed for the initial state by examining the maximum utility.

4.4 Implementation Details

The adversarial search program based on the Max^n algorithm was implemented in MS C++ 2017. For the implementation we created a class based design for the game which has allowed for a great deal of flexibility in the code for future modifications. This has been done in order to provide scope for scaling up the implementation to tackle greater challenges.
Talking about the implementation of the algorithm, instead of searching the entire game tree, we created a depth limited search, with the maximum depth being determined by considering an allowable response time for the program.
Also, in order to increase the efficiency of the search, we traverse each node only once along a path, and we also store the result for each state once it has been processed. This gives us an expected time complexity of $O(N^2)$, where N is the total number of states reachable from the initial state, which is examined in detail in the next section.

4.5 Time Complexity Analysis

The time complexity for a naive implementation of the Max^N algorithm is dependent on the branching factor for each state, and the search depth. This is similar to that of the naive implementation of minimax algorithm, and both of these give an asymptotic complexity of $O(B^M)$ where B is the branching factor, and M is the depth to which we search. This is bad in case of games which have a large number of branches from a single state, since the depth to which we can look forward is limited by time constraints.
The naive method would prove to be quite slow and can limit the scope of the game. Thus we propose

No. of Players (N)	Branching Factor (B)
2	5
3	9
4	14
5	18

Table 1: Estimated branching factors for different no. of players

No. of Players (N)	$D = 2$	$D = 3$	$D = 5$	$D = 8$
2	0.000s	0.000s	0.011s	0.659s
3	0.000s	0.002s	0.230s	$>$ 1min
4	0.000s	0.008s	1.265s	$>$ 1min
5	0.001s	0.012s	4.726s	$>$ 1min

Table 2: Time taken by vanilla Max^N for a single move

some optimizations which can bring down the cost of searching the entire tree to $O(|S|^2)$, where $|S|$ is the cardinality of the set of all the valid states. The two optimizations proposed are:

1. In a single search, we never visit a node twice. Instead we simply call the $Eval()$ function on it to calculate its expected utility.
2. Once we have searched a given state (taken as initial state), we never search it again. If encountered in the utility search of some other state, we simply use it's calculated utility.

Thus in a single search we expand only $|S|$ states, and the search is performed for a maximum of $|S|$ times. Thus the overall complexity comes out to be $O(|S|^2)$.
Now since each state is represented as a $N + 1$ size tuple, and the first N elements are pairs of integers $0 \leq h_1, h_2 \leq 4$, and the last element $1 \leq p_i \leq N$, therefore a theoretical upper bound on the number of states is $N * 5^{2N}$. Thus $|S| \leq N * 5^{2N}$ and hence a theoretical upper bound on the time complexity is $O(N^2 * 5^{4N})$, which is the complexity for a full tree search.
However, not all states are valid due to which the number of states to be searched reduces, and hence the complexity is somewhat lower. Plus, this complexity results from assuming a complete tree search at once. Instead we can use a dynamic method, where we search only if required, and store the results every time we search a node. This can bring down the number of searches to the number of moves in the game, which is much less than the number of states in the game. Suppose the game lasts for K moves, then the time complexity would be $O(K * |S|)$, with a maximum theoretical time complexity of $O(K * N * 5^{2N})$.

No. of Players (N)	$D = 2$	$D = 3$	$D = 5$	$D = 8$
2	0.001s	0.001s	0.007s	0.010s
3	0.001s	0.002s	0.048s	0.218s
4	0.001s	0.003s	0.217s	3.596s
5	0.001s	0.003s	0.579s	8.853s

Table 3: Time taken by Max^N after optimizations for a single move

As we can see the optimizations drastically improve the speed of the search, and hence a greater number of depths can be searched. The slight decrease in speed in the shallower depths is due to the overhead encountered in storing states, which is more than compensated by the reduction in number of states expanded.

4.6 Worked Out Example

The given example showcases the Max^N algorithm being applied in practice for $N = 2$. For space constraints, the value of N has been kept low, and the entire search tree hasn't been shown. The evaluation function is used to calculate expected utilities at the leaf nodes, which are then propagated

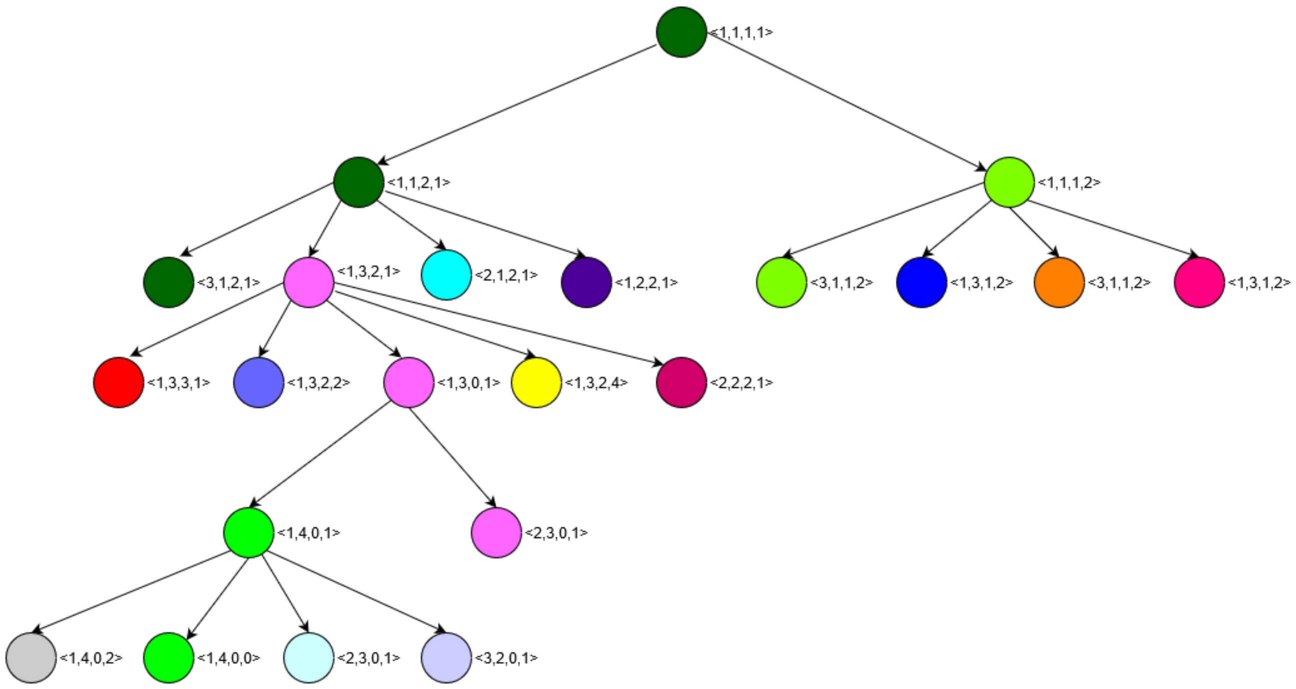

Figure 2: A worked out example showcasing the use of Max^N for $N = 2$

upwards by the Max^N algorithm. The colors represent different utilities, and same colors in a parent node indicate that the expected utility for the parent is same as that of the child. In case of a tie, the leftmost node is chosen. At each level, the player whose turn it is tries to maximise his own utility, which guides the working of the algorithm.

5 Summary and Conclusions

In this report we tried to create an N player AI variant for the game Chopsticks using adversarial search. We used Max^N [Kor91] algorithm to solve the problem as an N player, non-cooperative, perfect information game. We created our own evaluation function to score the game states, while also making several optimizations in the naive Max^N algorithm. By doing this, we gained an insight into game theory, and also cleared our concepts about using artificial intelligence to solve games. The project also helped us to refine our algorithmic knowledge while optimizing the Max^N algorithm.
The main limitations that we faced while doing this project were:

1. While the Max^N algorithm is optimal for finding the equilibrium point, it still doesn't guarantee that the play of the players will be optimal, since the equilibrium is dependent on the choice of the evaluation. Hence there is a need for judging the performance of the game using some metrics, which can be looked further into. The author's personal views on this topic are that the game played quite like a human, and was able to carry the game forward for a long time, while also managing to defeat the human player several times, without suffering any in return.

2. The author of [Stu03] mentions that while pruning is always inefficient for multi-player games, other algorithms such as Paranoid algorithm [Stu03] can perform better than Max^N and hence they can also be looked into.

6 Acknowledgements

I would like to acknowledge my parents, my sister and my grandfather for their constant support and encouragement during these time. I would also like to thank my friend and colleague Nivedit Jain (B18CSE039) for his valuable suggestions and review.

References

[Hud80] Paul Hudson. "Game theory: mathematical models of conflict, by A. J. Jones. Pp 309. £15. 1980. SBN 0 85312 147 8 (Horwood/Wiley)". In: *The Mathematical Gazette* 64.430 (1980), pages 299–300. `https://doi.org/10.2307/3616751`.

[Kor91] R. Korf. "Multi-Player Alpha-Beta Pruning". In: *Artif. Intell.* 48 (1991), pages 99–111.

[LI86] Carol Luckhart and K. Irani. "An Algorithmic Solution of N-Person Games". In: *AAAI*. 1986.

[Stu03] Nathan Sturtevant. "A Comparison of Algorithms for Multi-player Games". In: *Computers and Games*. Edited by Jonathan Schaeffer, Martin Müller, and Yngvi Björnsson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pages 108–122. ISBN: 978-3-540-40031-8.

[Wik16] Wikipedia contributors. *Non-Cooperative Games — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Non-Cooperative_Games&oldid=739858873`. [Online; accessed 26-October-2020]. 2016.

[Wik20a] Wikipedia contributors. *Chopsticks (hand game) — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Chopsticks_(hand_game)&oldid=982996841`. [Online; accessed 25-October-2020]. 2020.

[Wik20b] Wikipedia contributors. *Combinatorial game theory — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Combinatorial_game_theory&oldid=969579580`. [Online; accessed 25-October-2020]. 2020.

[Wik20c] Wikipedia contributors. *Minimax — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Minimax&oldid=984478181`. [Online; accessed 25-October-2020]. 2020.

[Wik20d] Wikipedia contributors. *Perfect information — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Perfect_information&oldid=969838373`. [Online; accessed 26-October-2020]. 2020.