Numerical Tic-Tac-Toe on the 4×4 Board

Bryce Sandlund[1], Kerrick Staley[2], Michael Dixon[2], and Steve Butler[2]

[1] University of Wisconsin–Madison, Madison, WI 53706, USA
bcsandlund@gmail.com
[2] Iowa State University, Ames, IA 50011, USA
kerrick@kerrickstaley.com,
{medixon,butler}@iastate.edu

Abstract. Numerical Tic-Tac-Toe on the $n \times n$ board is a two player game where the numbers $\{1, 2, \ldots, n^2\}$ are divided between the two players (usually as odds and evens) and then players alternately play by placing one of their numbers on the board. The first player to complete a line of n numbers (played by either player) that add up to $n(n^2 + 1)/2$ is the winner. The original 3×3 game was created and analyzed by Ron Graham nearly fifty years ago and it has been shown that the first player has a winning strategy. In this paper we consider the 4×4 game and determine that in fact the second player has a winning strategy.

Keywords: Tic-Tac-Toe, games, symmetry, backtracking, pruning.

1 Introduction

Tic-Tac-Toe is a classic game that is familiar to people of all ages. Because of its familiarity this game is often used as a starting example of how to mathematically analyze a game, and it is well known that in optimal play by both players the game will always end in a tie.[1] One of the best known Tic-Tac-Toe strategy guides was created by Randall Munroe in an XKCD posting [6].

Nearly fifty years ago Ron Graham created a variation of Tic-Tac-Toe which has come to be known as "Numerical Tic-Tac-Toe". The game is still played on the same 3×3 board but now instead of using $\times$'s and $\circ$'s the two players are given the numbers $\{1, 3, 5, 7, 9\}$ and $\{2, 4, 6, 8\}$, respectively. The players take turns (with the odd player going first) and at each round the players put one of their unused numbers on an open square on the board. The first player to create any three numbers in a line that sum to 15 wins the game.

Numerical Tic-Tac-Toe is easily played by two players and the reader is encouraged to give it a try to help get a sense of the game. It should be noted that there are many implementations of this game online as well as apps for portable devices.

[1] Even modest training can result in a "good" player. At one time people would compete against chickens playing Tic-Tac-Toe in casinos in Atlantic City (though the chickens themselves responded more to lighting cues than strategy).

By extensive case analysis carried out by hand, Graham [1] determined that remarkably the *first* player has a strategy that can guarantee a win. An exhaustive computer analysis by Markowsky [4,5] thirty years later verified the result of Graham, while Orr and Cooper [7] subsequently gave a compact strategy for the game.

This game can be generalized to be played on any size board. Usually, on the $n \times n$ board the numbers $\{1, 2, \ldots, n^2\}$ are divided into the odd's and even's. The players take turns, starting with the odd player, and at each round the players put one of their unused numbers on the board. The first player to complete a line of any n numbers that sum to $n(n^2+1)/2$ wins the game. (The value $n(n^2+1)/2$ comes from the average value of $1, 2, \ldots, n^2$ being $(n^2+1)/2$ and then having n of them.)

In this paper we will outline the computation that was done to carry out the analysis for $n = 4$. Our exhaustive computation shows that in optimal play the *second* player has a winning strategy (though we do not have a compact description of such a strategy).

We mention in passing that there are other variations that could be considered for this game. One variation that we also explored was altering the initial division of $1, 2, \ldots, 9$ between the two players for the 3×3 game. It turns out that regardless of which five numbers the first player has they will always be able to win. This was independently confirmed by Bennett Hansen [2]. For the 4×4 case one could also create variations by changing the initial division; or giving each player the numbers $\{1, 2, \ldots, 8\}$ and then the winner would be the first person to get four numbers in a line that total to 18. We have not considered these variations here but the technique we will give can be used to analyze these and other situations as well.

One popular approach to solve perfect information games that has proven effective is using retrograde analysis, i.e., starting at the finishing positions and then working backwards. This works well when there are relatively few finishing positions, e.g., chess endgames [9]. This technique was also used to solve end game for checkers when there were relatively few pieces (later Schaeffer et al. [8] gave a complete solution of checkers combining various ideas). While this technique could work in this setting we will opt instead to use symmetry and efficient pruning of the game tree to determine the result. A survey of games that have been solved, including a discussion of various techniques to solve them, is given by van den Herik et al. [3].

2 Symmetries of the 4×4 Game

One naïve approach to this problem is to determine every possible state of the board and then form a graded poset with the unique maximal entry corresponding to an empty board and then as we go down we look at all possible ways to legally insert one number until either there is a win or the game results in a tie. Given such a poset we could then easily determine the winner of the game by working from the bottom to the top. However an approximation for the number of boards at depth k is

$$\binom{16}{k}\binom{8}{\lfloor k/2\rfloor}\binom{8}{\lceil k/2\rceil}k!.$$

That is, choose k positions out of 16 possible positions, then choose which odds are to be played, which evens are to be played and put them on the board in all possible ways. Summing up over all possible k then gives us 2.7×10^{15} boards. For comparison the 3×3 version of Numerical Tic-Tac-Toe has 9.3×10^{6} and classical Tic-Tac-Toe has $6,046$. (This count gives all possible boards, but some of these boards would not occur in gameplay as they contain within them two or more disjoint winning lines which would indicate play has already stopped.)

Even with advances in computing power and memory storage this is still prohibitive to approach an analysis of the 4×4 board. We will employ several techniques to reduce the size of this problem to a point where the computation can be carried out efficiently.

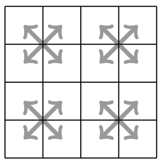
One of the most important tools that we have is to use the symmetry of the board. That is a bijection from the board to itself which preserves lines. To be more precise when we have the following board:

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

then the lines are:

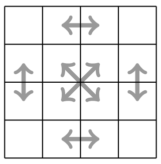
$$\{A,B,C,D\},\{E,F,G,H\},\{I,J,K,L\},\{M,N,O,P\},\{A,E,I,M\},$$
$$\{B,F,J,N\},\{C,G,K,O\},\{D,H,L,P\},\{A,F,K,P\},\{D,G,J,M\}.$$

Proposition 1. *The bijections of the board consist of compositions of the following maps: rotations, reflections, and the two maps shown below.*



=

F	E	H	G
B	A	D	C
N	M	P	O
J	I	L	K



=

A	C	B	D
I	K	J	L
E	G	F	H
M	O	N	P

For the two maps given above we will call the one on the left the "cross-symmetry" and the one on the right the "X-symmetry". When all of these are combined we end up with 32 different bijections that preserve the lines of the 4×4 board.

Proof. A simple check will verify that each one of those maps will preserve lines, so it suffices to show that if we have preserved lines that we must be a composition of these maps.

Next we note that each map we have outlined is reversible (i.e., for rotation we reverse the direction and for the other maps we simply apply the map a second time). Therefore to show that we have all possible bijections by combining these maps, it suffices to show how we can apply these maps to get back to the starting board.

So suppose we have a board that has preserved lines. Then by applying rotations we can place the A in the upper left corner. Note that A, D, M and P will always have to form the corners of a square and so if needed we can apply reflection to place D in the upper right corner. Reflection will achieve this because A and D cannot be on opposite corners, since that would force B and C to be one of the four center squares. The center squares are involved in three lines, but B and C are only involved in two, meaning they can never be center tiles. Therefore we are in one of the following two situations:

A			D
M			P

	A	D	
	M	P	

Similarly, F, G, J, and K will form another square and their positioning must agree with the above boards in preserving diagonal lines. So in the first case we now have the following four possibilities:

A			D
	F	G	
	J	K	
M			P

A			D
	F	J	
	G	K	
M			P

A			D
	K	G	
	J	F	
M			P

A			D
	K	J	
	G	F	
M			P

Every remaining unfilled square is contained in two lines and we can determine what value (if possible) must go in the square by taking the intersection of the two lines. The first possibility reduces to the identity, the fourth gives the X-symmetry, and the other two are impossible.

For the second case we have the following four possibilities:

F			G
	A	D	
	M	P	
J			K

F			J
	A	D	
	M	P	
G			K

K			G
	A	D	
	M	P	
J			F

K			J
	A	D	
	M	P	
G			F

Again proceeding as before we can fill in any remaining squares by looking at the intersection of the lines. The first possibility reduces to cross-symmetry, the fourth possibility is the result of composition of X-symmetry and cross-symmetry maps, and the other two are impossible.

Thus using only our given maps we have accounted for each valid bijection of the board. ■

There is one other natural candidate for symmetry involving manipulation of the numbers themselves rather than the location of each entry. The symmetry was used by Markowsky [4] in the 3 × 3 version and was found due to the fact:

$$a + b + c = 15 \leftrightarrow (10 - a) + (10 - b) + (10 - c) = 15.$$

Simple algebra takes the original sum $n(n^2 + 1)/2$ and finds that in the general case, subtracting each filled entry from $n^2 + 1$ presents a possible symmetry. For the 4×4 version, each filled cell q would then be replaced by $17 - q$. An example is given below:

$$\begin{array}{|c|c|c|c|}\hline 3 & & & \\ \hline & 16 & & \\ \hline & 10 & 7 & \\ \hline & & & \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline 14 & & & \\ \hline & 1 & & \\ \hline & 7 & 10 & \\ \hline & & & \\ \hline \end{array}$$

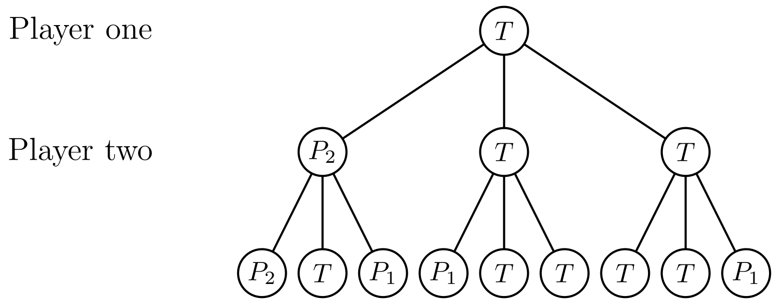
However, due to the fact replacing q by $17 - q$ changes the parity of an entry, this symmetry is invalid for two reasons. First, if we apply this after an odd number of plays then we will be in an impossible board, i.e., one with more even numbers than odd numbers. Second, if we apply this after an even number of plays then situations can change dramatically. As an example of this latter case, consider the above scenario where it is now the first player's turn. In the original board the first player can block but cannot win with the next move, while in the second board the first player can place 9 in the lower right corner and win.

We note that this last symmetry does work for the 3 × 3 board and more generally any $n \times n$ board when n is odd, due to the fact $n^2 + 1$ is even whenever n is odd.

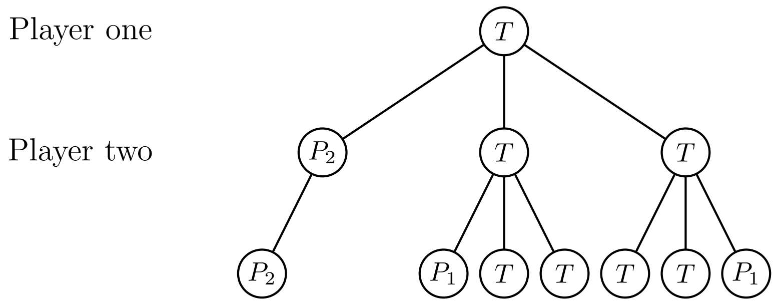
3 Splitting the Computation

As already mentioned, we can form a graded poset consisting of all boards where connections go between consecutive levels between boards that differ by a legal move. In this poset we then identify each "winning move" and break all connections below such boards and work from the bottom up. Each board can be labeled with one of three possibilities $P1$, $P2$, or T for "player one wins", "player two wins" and "neither player can guarantee a win". Working from the leaf nodes upwards we visit each board and identify the labels of all boards below it which it connects to and then determine the label of the board by the following rule: If it is player one's turn then $P1 \succ T \succ P2$ (i.e., $P1$ is preferred to T which is preferred to $P2$ and the player takes the best labeling of all boards immediately below); while if it is player two's turn then $P2 \succ T \succ P1$. Finally, the label of the empty board indicates the outcome of optimal play on the part of both players.

Instead we will opt for starting at the root and working our way down the tree and filling in this information as we go along. The implemented program is a version of the minimax algorithm, working in a depth-first, backtracking manner. One advantage of this is that we can use alpha-beta pruning to eliminate the need to compute significant parts of the tree (i.e., avoid having to consider some board configurations). Suppose that below is part of the tree for us to consider (where on the left we have indicated which player is playing):



Given that at each board we will scan its children from left to right, we can then trim off parts of the tree that we guarantee are not necessary to visit, giving us the following:

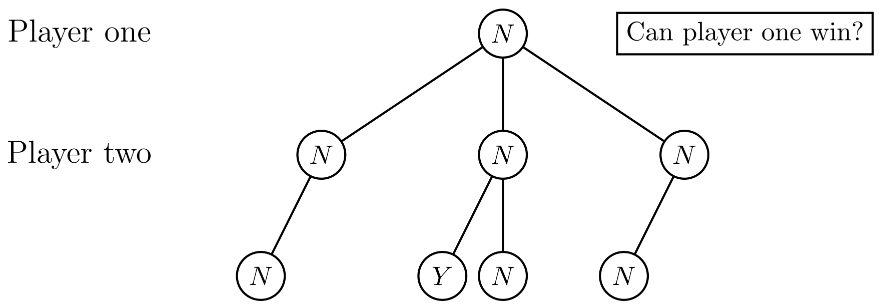


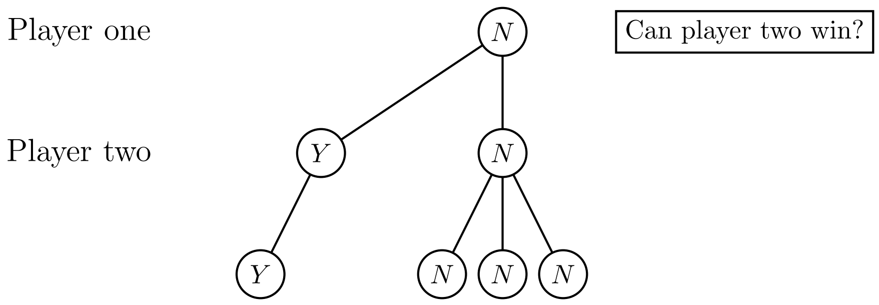
Since this technique is applied at each level of the tree, this pruning has a dramatic effect on execution time. But we can prune even more if we run the computation twice and instead of looking at $P1$, $P2$ and T we ask the following two questions:

- Can player one force a win?
- Can player two force a win?

While forming the tree for each of these two situations we will use Y and N for "Yes" and "No" respectively. For the first question we note that $Y = \{P_1\}$ and $N = \{P_2, T\}$, further when it is player one's turn $Y \succ N$ and for player two $N \succ Y$. For the second question we note that $Y = \{P_2\}$ and $N = \{P_1, T\}$, further when it is player one's turn $N \succ Y$ and for player two $Y \succ N$.

Applying this to the above tree and trimming as we are wont will result in the following two trees:





In this particular problem space, we found that examining only two utility values instead of three sped up computation exponentially due to the fact many tie boards exist in the game space. With this technique, only one player must search to find a winning configuration, but the other can return as soon as a tie board is found. In theory, this could reduce runtime from $O(b^d)$ to $O(b^{d/2})$ (where b is the branching factor and d is the depth), due to the repeating pattern of $1 * b * 1 * b$... corresponding to an immediate return vs. searching every child. In practice, we found the strategy more effective for player one's computation than player two's computation (sensible considering our result) and encourage the reader to look at the number of boards visited at each depth in the attached appendix, which is explained in more detail in the following section.

The authors comment that this general technique is probably known in AI literature and can be seen as similar to other techniques such as proof-number search [10]. For the record, however, we note that the above strategy could be easily translated to any general problem space with k utility values to turn the original question into $O(log(k))$ separate questions by binary searching for the best achievable utility value.

4 Results

In the previous two sections we have outlined our basic approach. First we run two computations to ask starting from an empty board whether each player can win. To further help prune we store all boards up through some preset depth and the corresponding answers for those boards. Symmetries were only used in the first five depths, a cutoff found by trial and error to avoid the expensive cost of calculating symmetries but keep the gain of avoiding duplicate computation. Boards were still stored after this point since it is quite possible to arrive at the same board from two or more different paths. Finally, due to memory constraints, to answer player two's computation, periodically we removed configurations from memory at the deeper depths of the memory storage.

In the appendix is the information regarding the outcome when we ran the program for the question: Can player one force a win? For each depth (i.e., number of rounds played in the game) it records the number of boards that were visited, how many were determined because they were already calculated in memory (stored up through depth 9), and how many times the answer was "Yes". Since our program looked one move ahead to determine if the opponent can win, all recorded "Yes's" represent a win at least one depth away from the winning board. Similarly, in the appendix is the same information when we ran the program for the question: Can player two force a win? (Where we stored boards up through depth 10.)

In particular, we see that for the 4×4 Numerical Tic-Tac-Toe game that player two can force a win. Comparing the amount of computation required to determine the answers to our original questions, it appears that in random play that it is much easier to stop player one from winning than it is to make player two win. This is easy to convince yourself of in the 3×3 case where it is not so hard to stop player two from winning but far from obvious how to have player one win.

By using an appropriately pruned tree for the question "Can player two force a win?" we would have a perfect strategy for the game. Unfortunately this requires immense amounts of storage and so is likely impractical to program. Therefore we expect that for most players (even most low-powered computer players with limited time to act in each move), games will likely end in a tie.

There are of course many interesting questions left to ask for numerical tic-tac-toe, including whether or not one of the players always has a winning strategy. One can imagine that we divide the numbers up arbitrarily or play in larger boards. If one player did always have a winning strategy is there an easy explanation for which player it is and what strategy they should pursue? Another question that we did not answer here is whether player two could force an early win, i.e., is it possible to finish the game in the fourteenth round? We do not yet have the answers to these questions, but look forward to the next move in this area.

Implementation

The program for carrying out the computation was written in Java, and is available online: `https://github.com/brycesandlund/4x4TicTacToe/`

References

1. Graham, R.: Personal communication
2. Hansen, B.: Personal communication
3. can den Herik, H.J., Uiterwijk, J.W.H.M., van Rijswijck, J.: Games solved: Now and in the future. Artificial Intelligence 134, 277–311 (2002)
4. Markowsky, G.: Numerical tic-tac-toe–I. J. of Recreational Math. 22, 114–123 (1990)
5. Markowsky, G.: Numerical tic-tac-toe–II. J. of Recreational Math. 22, 192–200 (1990)
6. Munroe, R.: xkcd: Tic-Tac-Toe, `http://xkcd.com/832/`
7. Orr, S., Cooper, C.: A compact strategy for numerical tic-tac-toe. J. of Recreational Math. 27, 161–171 (1995)
8. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is solved. Science 317, 1518–1522 (2007)
9. Thompson, K.: Retrograde analysis of certain endgames. ICCA Journal 9, 131–139 (1986)
10. Allis, L.V., van der Meulen, M., van den Herik, H.J.: Proof-number search. Artificial Intelligence 66(1), 91–124 (1994)

Appendix

For the question “Can player one force a win?” we have the following information in regards to the computation:

Depth	Boards Visited	Found in Memory	How many “Yes”’s
0	1	0	0
1	128	112	0
2	16	0	0
3	1,440	136	0
4	1,304	0	0
5	88,430	33,554	0
6	57,295	0	2,419
7	2,158,685	1,131,526	2,419
8	1,334,445	6,328	309,061
9	23,654,937	9,012,830	306,283
10	18,237,546	0	3,896,978
11	215,581,273	0	3,896,978
12	221,312,077	0	9,627,782
13	1,462,159,978	0	9,627,782
14	1,452,532,196	0	0
15	2,818,792,528	0	0
16	2,818,792,528	0	0

For the question “Can player two force a win?” we have the following information in regards to the computation:

Depth	Boards Visited	Found in Memory	How many “Yes”’s
0	1	0	1
1	128	112	128
2	72	0	16
3	1,467	19	1,411
4	17,358	329	1,396
5	155,748	5,215	140,006
6	2,822,029	150,020	135,188
7	14,434,502	833,734	11,892,556
8	185,959,992	16,012,248	11,122,103
9	572,733,507	45,970,492	413,409,269
10	5,686,646,311	699,206,459	374,906,758
11	10,231,334,279	0	5,593,772,898
12	82,553,255,357	0	5,593,772,898
13	109,945,222,392	0	32,985,739,933
14	521,913,263,546	0	32,985,739,933
15	488,927,523,613	0	0
16	488,927,523,613	0	0