Numeric Analysis on Final Tie Positions of k by n Tic-Tac-Toe

Taerim Teddy Kim, Karnaa Mistry, Weij Zheng

December 14, 2020

Introduction

Tic-Tac-Toe, the classic paper-and-pencil game of X's and O's on a 3×3 grid, among many other board games, has extensive potential for combinatorial analysis. Due to Tic-Tac-Toe's composition of X's and O's, it has a natural representation as a $k \times n$ binary matrix.

This project investigated enumeration of final positions of $k \times n$ games of Tic-Tac-Toe, that specifically resulted in a *tie.* In particular, this definition of tie means that there are no horizontal, vertical, nor diagonal streaks of k 1's or 0's in the final matrix. This ensures that all of the binary/(0,1)-matrices have no empty indices, and it makes the ideas behind counting (seemingly) straightforward.

The main approach consisted of constructing initial rows of the final matrix via permutations of 1's and 0's, then building upon those initial rows such that no k-streak is created. This was accomplished by computing a Cartesian product of all possible elements of the initial row $k-2$ times. Doing so would not cause any violation, since there cannot be a vertical nor diagonal k-streak in $k-1$ rows, and it was ensured beforehand that no row would consist of all 0's or 1's. For matrices with an even number of squares, there will be an equal number of 1's and 0's. For those with an odd number of squares, there will be one more 1 than there are 0's.

Example:
Good and bad 3×3 matrices:

$$\text{Good} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} ; \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad \text{Bad} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \text{(violates diagonal)} \tag{1}$$

The Manageable Case: $k = 3$

The 3×3 matrix is the classic Tic-Tac-Toe board, and the investigation of $3 \times n$ proved to be successful, thanks to Dr. Z's Maple package [1]. The Maple procedure **Alph3**() successfully generated the 36-element set of "letters", which are just all possible values of the matrix consisting of row 1 and row 2. Using this, **Followers3**() takes any such letter in **Alph3**() and finds all possible arrangements allowed to be the third row. (See (2).)

Sample step:
Building a final position by taking one "letter" in **Alph3**(), and finding its follower(s) and third row(s) (this only has one, [0 1 0]):

$$M = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ ? & ? & ? \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2}$$

Using $\mathbf{TicB}(3, n, 3)$ for values of n starting at 1, it was possible to generate the matrix representations of final $3 \times n$ Tic-Tac-Toe positions (of games that resulted in a tie) for up to a relatively small value of n. Taking the number of elements of these results, the first enumerated sequence was found. For any n beyond, however, the time to compute gets extremely long.

This led to the importance of using generating functions. By using $\mathbf{GF3tx}(t, x)$, which uses weight-enumeration techniques, a generating function for $3 \times n$ Tic-Tac-Toe was found. This computed the same results as $\mathbf{nops}(\mathbf{TicB}(3, n, 3))$, but, much FASTER, after using taylor expansions for even n, for odd n, and merging the two sequences to get one for all n, namely in $\mathbf{AllTTT3}(N)$. This allowed very convenient recurrences to be found; however, extending onward to larger values of k, the task would prove to be extremely challenging, and essentially not possible.

The Problems of Larger k

For $k = 4$, **Alph4**() and **Followers4**() take some time to compute, but still work similarly, by initializing all possible "letters", and finding what followers (final rows) would be allowed. Yet, now, there are 14 initial permutations versus 6 for $k = 3$, since there are 14 numbers expressed with 4 bits, from 0 to 2^4, after removing 0 and 16. Doing a Cartesian product $k - 1 = 3$ times means that there are $14^3 = 2744$ elements in **Alph4**(). While this is relatively manageable, it is significantly greater than the 36 elements for $k = 3$. Expanding onward, $k = 5$ would involve $30^4 = 810000$ elements.

Conceptualizing growth in difficulty:
Seeing how fast $|\mathbf{Alphk}(k)|$ grows: (3)

- $|\textbf{initial set } S| = 2^k - 2$;

  Take the Cartesian product $k-1$ times:

- $|\mathbf{Alphk}(k)| = (2^k - 2)^{k-1}$;

For $k = 3, 4, 5, 6, 7$:
$[3 \to 36;\ 4 \to 2744;\ 5 \to 810000;\ 6 \to 916132832;\ 7 \to 4001504141376]$;

There were attempts to create **GF4tx**(t, x), and even **GFktx**(k, t, x), although computation proved to be essentially impossible. Since these procedures were based on **GF3tx**(t, x), they involved solving a system of equations with $|\mathbf{Alphk}(k)|$ equations with $|\mathbf{Alphk}(k)|$ variables. Maple can handle 36, but 2744? What about 81000? It became increasingly more clear that finding generating functions of such complicated cases would not be possible due to physical of computation. Instead, some smaller sequences were generated via **TicB** (direct computation and violation-checking), even though many of them took quite a long time for relatively short sequences.

Conclusion & Potential Future Projects

Overall, it was possible to successfully enumerate the sequence of final tie positions for $3 \times n$ Tic-Tac-Toe games, but there was extreme computational difficulty for arbitrary $k > 3$. Generating the initial three rows for $k = 4$ was manageable, but due to the significant increase in the number of final rows, computing a generating function would take far too long, and be nearly impossible, as would be the case for $k > 4$. Enumeration of some small sequences and boards without generating functions was still interesting. As a whole, these methods certainly led to a clearer understanding of how immense the complexity of the topic truly is.

Upon generalizing k, some noticeable relationships did arise despite the technical limitations. Basing off of Dr. Z's advice and methodology behind the efficient procedure **GF3tx**(t, x), creating a concept of one for a general k led to some observations. One was that when k is an even number, the number of 1's in $k \times n$ Tic-Tac-Toe has only even number cases for 1's. This led to the procedure **AllTTTkEven**. When k is odd, the number of 1's in the final Tic-Tac-Toe has both even and odd cases, and so the ideas of **OddTTTkOdd** and **EvenTTTkOdd** led to conjecture **AllTTTkOdd**. From these observations, an integrated procedure/formula (**TTT**) was created, and successfully matches what is expected from **nops**(**TicB**$(3, i, 3)$), just at a much faster rate. For other k's, the main methods would be relatively similar, but physical computation is the number one barrier. For more details on this theoretical insight, see **PaperTTTk**() from the final Maple package.

Looking forward, there are limitless possibilities for different approaches and related experiments. For example, in order to reduce the amount of computation for larger k values, one might restrict the center square(s) and count along the remaining positions, or alter win/lose/draw conditions that ease the growth of complexity as k increases. In fact, for practical 4×4 Tic-Tac-Toe games, most rules state that a player could win via a 4-streak vertically, horizontally, or diagonally, *as well as* 4 in a 2×2 square, and control of the 4 corners.

The possibilities are endless, and depending on what limits or conditions are chosen, future studies could perform extensive analyses while creating manageable databases, computing workable generating functions, and so much more, whether it be for specific values, or for all cases onward, too.

References

[1] Dr. Doron Zeilberger,
Math 454, Section 02 (Combinatorics), Rutgers University, Fall 2020.
`https://sites.math.rutgers.edu/~zeilberg/math454_20.html`.