An Algorithm for Flip

Jack Cutler[†] and Jack Liddicoat[†]

[†]Wake Forest University, Winston-Salem, N.C., U.S.A.

Corresponding author: Andy Sphar

Abstract

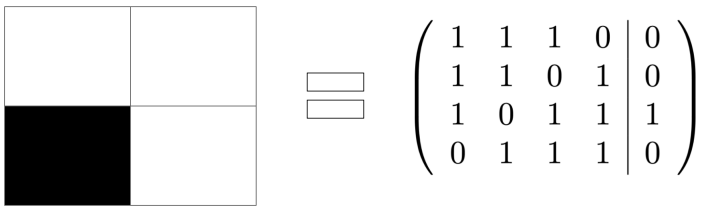
In this short paper, we discuss an algorithm and a conjecture for the popular online game Flip. We discuss how techniques from linear algebra helped us in making such algorithm and form a conjecture about the solvability of given 1 x n boards.

Keywords: Flip; Matrix; Linear Algebra

1 Overview

Flip is a game where a user is given a board with black and white squares, say n x m where n represents the number of rows of the board and m represents the number of columns. Every time a user clicks on a given square, it flips the non-diagonal, adjacent squares to the other color. Through a series of clicks, the goal of the game is to make all of of squares on the board white.

Using techniques from linear algebra, we were able to transform a given board into a matrix. The conversion from a 2 x 2 matrix is shown below in Figure 1.



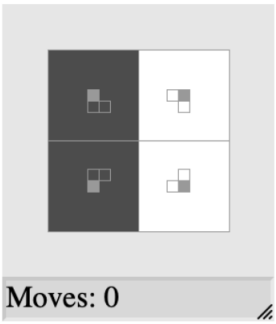
$$\left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right)$$

Figure 1: A 2 x 2 Flip grid to a 4 x 4 matrix

The 1's and 0's in the 4 x 4 matrix represent the affected squares if a given square on the 2 x 2 board is flipped (clicked). So, a_{11}, a_{12}, and a_{13} are given the value of 1 and a_{14}, the fourth tile, is given 0 (it is diagonal to the first tile). We then repeat this process for the rest of the squares, hence why there are four rows in our matrix. The augmented column is coded as the vector $\vec{v} = \langle 0, 0, 1, 0 \rangle$ because the third tile is colored black.

We then performed a Gaussian row reduction process to get the given matrix into reduced row echelon form (RREF). Let $\vec{x}$ denote the given board (the vector in the augmented column). If a given matrix was consistent (i.e., $A\vec{x} = \vec{b}$ for some $\vec{b} \in \mathbb{R}^n$), then the given board has a solution. If not, then there is no possible sequence of clicks that can solve the board. Our main interests were what proportion of given boards were solvable and whether or not there were any patterns associated with if there were any patterns in the size of the boards that were or were not solvable.

Using our algorithm, we are also able to tell the user *how* to solve any given board. For example, if we have the starting position and matrix shown in Figure 2, then we row reduce the matrix and click the squares corresponding to the values in the augmented column.



$$\left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right)$$

Figure 2: Example matrix to solve

In the example above, our matrix in RREF is:

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right)$$

Since the the vector in the augmented column is $\vec{v} = \langle 0, 1, 0, 1 \rangle$, then we should click tile two and then tile four to solve the board.

2 Algorithm Design and Analysis

2.1 Code Design

We used SageMath 10.2 (1) for our computational work.[1]. We started off by defining a function that takes the number of rows and columns of the desired board as inputs. The goal of the function was to make the non-augmented portion of the matrix, which it can do very quickly, even for larger matrices.[2]. Then, we checked every possible board by using a **for** loop to make the augmented column. Then, using some of the built-in linear algebra Sage operations[3], we row reduced the matrix, which threw an error if the matrix was inconsistent and gave us the vector $\vec{x}$ that satisfies the equation $A\vec{x} = \vec{b}$ if the matrix was consistent. We then added a counter variable to count the number of equations that gave us some vector $\vec{x}$.

[1] All code for this project is available at https://github.com/jackliddicoat/Flip

[2] For example, the 5 x 5, the 6 x 6, and the 7 x 7 all take roughly 30-45ms to run.

[3] For reference, see https://doc.sagemath.org/html/en/tutorial/tour_linalg.html

2.2 Computational Problems

A problem that we have run into with our algorithm is, like many algorithms, it lacks efficiency. Figure 3 displays a line plot showing the estimated time it would take to compute a matrix of a given size. As can be seen, the relationship is exponential. A smaller, say 2 x 2, matrix takes a fraction of a second to run, but a larger, say 5 x 5, matrix takes hours. This is because the relationship between the matrix size and the number of possible starting boards. for the 2 x 2 case, the algorithm only has to row reduce 16 4 x 4 matrices. But, in the 5 x 5 case, the algorithm has to row reduce 33,554,432 (2^{25}) 25 x 25 matrices. We hope to find a faster, more efficient algorithm to better understand the nature of solvable starting boards.

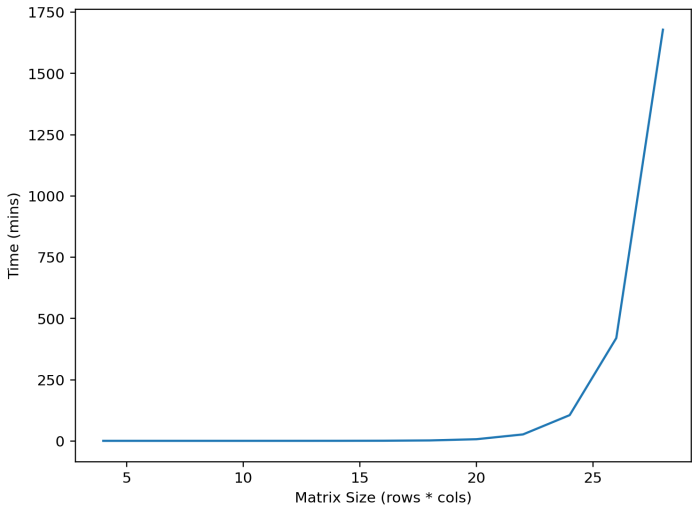


Figure 3: Estimated computation time of matrices

3 Computational Results

The results of our computations for given matrices is shown below.

Table 1: Proportion of boards solvable by size

Rows / Columns	1	2	3	4	5	6	7	9	10
1	1	0.5	1	1	0.5	1	1	0.5	1
2	0.5	1	0.25	0.5	1	0.25	1	0.5	0.25
3	1	0.25	1	1	0.125	1			
4	1	1	1	0.0625	1				
5	0.5	0.5	0.125	1					
6	1	1	1						
7	1	0.25							
8	0.5	1							
9	1	0.5							
10	1	1							

One thing we noticed that was interesting was that, for the n x n case, it is not always solvable (notice how only 6.25% of 4 x 4 boards are solvable, compared to 100% of boards for 1 x 1, 2 x 2, and 3 x 3). One clear pattern in the data is that for the 1 x n case, if $n \equiv 0, 1 \pmod 3$. If $n \equiv 2 \pmod 3$, then half of the starting positions are solvable. This observation serves as the basis for our conjecture.

3.1 Conjecture

All starting positions of 1 x n board are solvable if $n \equiv 0, 1 \pmod 3$. *If* $n \equiv 2 \pmod 3$, *then half of the starting positions are solvable.*

References

The Sage Developers, *SageMath, the Sage Mathematics Software System (Version 10.2)*, 2023. `https://www.sagemath.org`.