Solving Lights Out Using Gaussian Elimination

Rohit Bagda, Chukwubueze Hosea Ogeleka, Katya Kelly

20 December 2017

Introduction

The game of Lights Out begins with an *m* x *m* board of buttons that double as light bulbs, where $m \geq 2$. In our implementation, we begin with all lights on and the objective is to turn them off. The catch is that when a button is toggled, its immediately adjacent neighbors are also toggled:

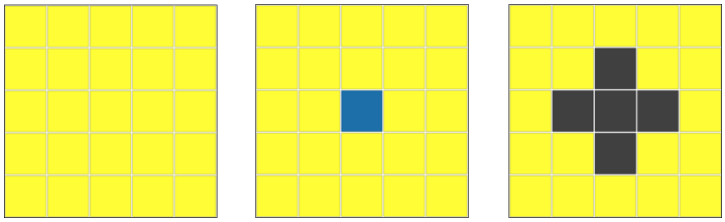


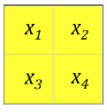
Figure 1: An example of a move in Lights Out on a 5x5 board.

We can use a variation of the Gaussian Elimination algorithm to find a solution to any square board. We focus on square boards in particular because they are guaranteed to have at least one state from which a solution state can be achieved (Caro, 1996 and Sutner, 1988). Additionally, we use Java to code an interactive visualization of the game up to a 99x99 board size.

Gaussian Elimination is interesting and important to consider in the context of this game because it is a systematic way of finding all possible solutions. While the strategy to the game may appear random to a player at first, understanding how the algorithm works can aid in the detection of intriguing patterns and sequences that are used to solve the puzzle.

Methodology

Lights Out can be solved using linear algebra. We can generate all possible solutions to the puzzle using Gaussian Elimination and backward substitution, although our goal in this implementation is to show how to achieve just one solution. A solution to the game is represented as a vector in $\mathbb{Z}_2^n$,[1] where *n* represents the number of bulbs on the board. We will walk through an example of how to obtain a solution for a 2x2 game board.



We begin by assigning a variable x_i to each bulb for $1 \leq i \leq n$ (see below for a diagram of the process). Next, we write a set of equations to represent the state of the board after each possible button toggle. We then rewrite these equations into a matrix, where each

[1] $\mathbb{Z}_2 = \{0, 1\}$, where 0 represents an "off" state and 1 represents an "on" state.

column represents one variable. The Gaussian Elimination algorithm operates on this matrix to transform it into row-echelon form[2]. Finally, we can perform backward substitution to solve for the value of each button and construct the solution vector. This vector summarizes which buttons need to be toggled; they do not need to be toggled in any particular order, so long as each is toggled exactly once.

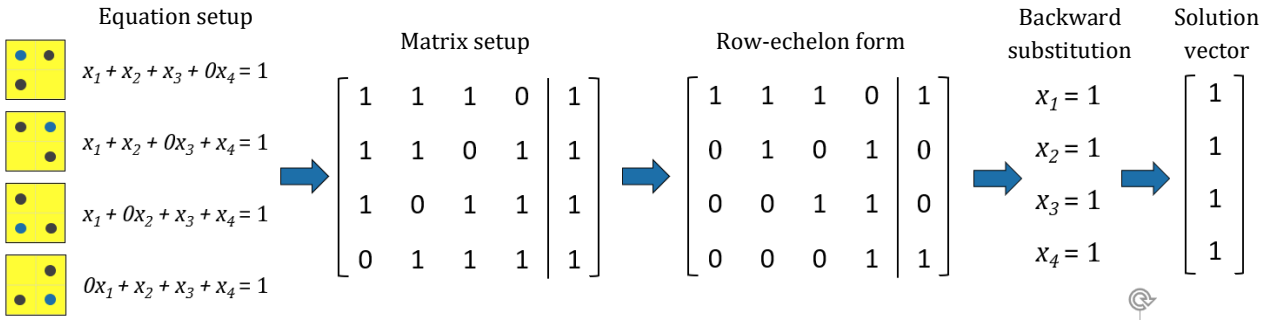


Figure 2: the process of finding a solution vector.

In this vector, a 1 indicates that a bulb should be toggled, while a 0 means it should not. Thus, for the example demonstrated above, the solution that our algorithm finds is to toggle each button exactly once. This is guaranteed to result in an all-lights-off position.

For boards of size 2x2, 3x3, and 4x4, the player is guaranteed to be able to achieve a solution from any state of the board. However, for dimensions of 5x5 and larger, there are states from which it is impossible to solve the game without undoing moves.

Algorithm Pseudocode

At the core of finding the solution is the Gaussian Elimination algorithm. To implement this algorithm, we consult pseudocode from the book *Introduction to the Design & Analysis of Algorithms* (Levitin, 2011). Levitin's algorithm generally operates on matrices of real numbers; however, we have modified it to operate on binary matrices. The following is the pseudocode of our implementation:

```
Algorithm BinaryGaussianElimination(A[0..n-1, 0..n-1])
//Input: Matrix A[0..n-1, 0..n-1]
//Ouput: Augmented matrix C[0..n-1, 0..n] in row-echelon form
        C[0..n-1, 0..n] = createAugmentedMatrix(A)
        for i = 1 to n-2 do
                pivotRow = i
                for j = i+1 to n-1 do
                        if |C[j, i]| > |C[pivotRow, i]|
                                pivotRow = j
                for k = i to n do
                        swap A[i, k] and A[pivotRow, k]
                for j = i+1 to n-1 do
```

[2] Row-echelon form requires 3 properties: (i) all nonzero rows are above any rows of all zeros, (ii) each leading entry of a row is in a column to the right of the leading entry of the row above it, and (iii) all entries in a column below a leading entry are zeros (Lay, 2012).

```
                temp = C[j, i]/C[i, i]
                    for k = i to n do
                        C[j, k] = (|C[j, k] – C[i, k]*temp|) % 2
    return C
```

A cursory analysis of this pseudocode suggests that the algorithm runs in n^3 time, where n represents the number of bulbs on the board. We can perform formal algorithm analysis to verify this intuition.

Analysis & Discussion of Efficiency

If we consider the number of bulbs n to be the input to the Gaussian Elimination algorithm, then the runtime of the algorithm increases in polynomial time as n increases. A graphical representation of the runtime reveals the efficiency class of Gaussian Elimination to be $\Theta(n^3)$:

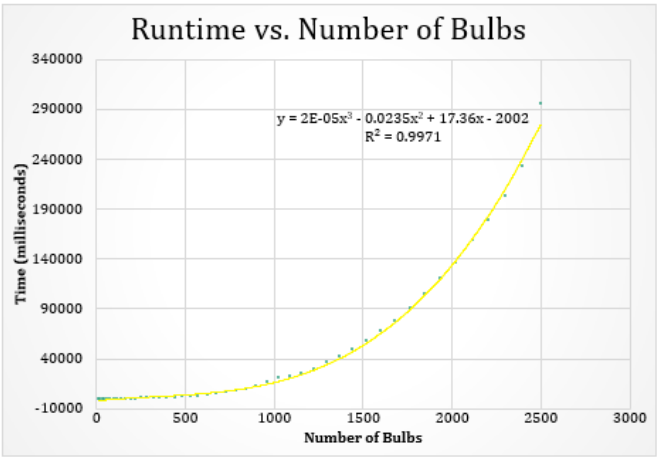
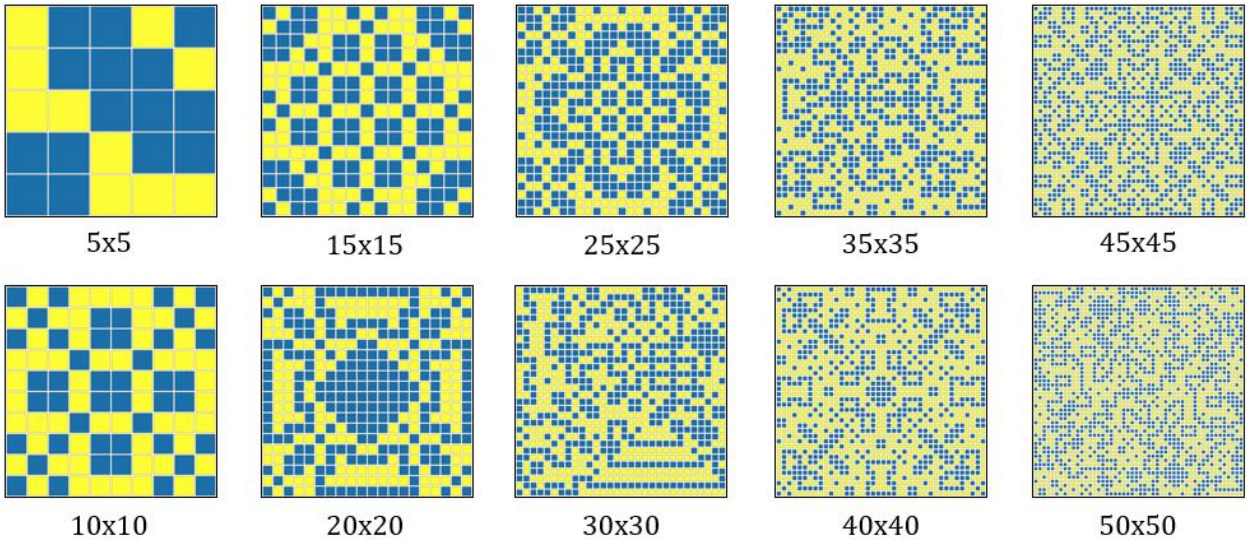


Figure 3: $O(n^3)$ efficiency

This may not seem like a particularly efficient approach to finding a solution. However, this is exponentially more efficient than trying to determine the quickest solution to the game: that is, the solution which requires the fewest buttons to be toggled. To find the quickest solution, we would have to solve for every possible permutation of the free variables in the augmented matrix. This would cause the algorithm to run in $O(2^n)$ time. For all but very small values of n, this is much less efficient than finding the solution obtained by setting all free variables equal to 0.

Implementation & Visualization

To illustrate that Gaussian Elimination works to produce a solution for any square game board, we have coded a visualization of the game. Our program allows the user to enter a dimension up to 99x99 and gives them three options: (1) play the game, (2) reveal a solution, or (3) play an animation that shows the board being solved. We include other customizable features, such as an animation speed slider and the option to play, pause, or reset the board.



References

Caro, Y. (1996). Simple proofs to three parity theorems. *Ars Combinatoria, Volume*(42), pp. 175–180.
This is one of the papers that is referenced by Minevich as having proven that any nxn board that begins with all lights on has at least one solution.

Lay, D. "Row Reduction and Echelon Forms." Linear Algebra and Its Applications, 4th ed., Pearson, 2012, pp. 12-24.
We referenced this chapter of the textbook for the formal definition of a matrix in row-echelon form.

Levitin, A. "Gaussian Elimination." Introduction to the Design & Analysis of Algorithms, 3rd ed., Pearson, 2011, pp. 208–216.
We referenced the pseudocode for the forward Gaussian Elimination algorithm (p. 211) and wrote code for it in our program. The input is a matrix that represents the results of toggling each button/bulb; for example, the first row in the matrix for a 2x2 board would be [1, 1, 1, 0], indicating that if the first (top left) button were toggled, the second (top right) and third (bottom right) buttons would also be toggled. For an nxn game board, this matrix will always be of size n^2xn^2. The output is a matrix in row echelon form which we will eventually use to find a solution to each nxn board with parametric vectors.

Minevich, I. (2012). Symmetric Matrices over F_2 and the Lights Out Problem. *arXiv:1206.2973*.
This is the paper from which we got the references to the Caro and Sutner articles.

Sutner, K. (1988). Additive automata on graphs. *Complex Systems, Volume*(6), pp. 649–661. This is one of the papers that is referenced by Minevich as having proven that any nxn board that begins with all lights on has at least one solution.