Lights Out With a Focus on Linear Algebra

Mahati Gorthy
Math 401: Applications of Linear Algebra
Professor Dimakis
Final Project

6 December 2024

Abstract

In this paper, I explore the game *Lights Out*, an electronic game that was first released by Tiger Electronics in 1995. I examined different aspects of Linear Algebra that can be used to solve the game, as well as further observations and ideas that I found interesting. With Linear Algebra, I determine conditions for solvability and define the linear algebraic ideas involved.

Lights Out: Introduction and Rules

Lights Out consists of 25 buttons, which are each arranged in a 5 by 5 array. Each button can be in one of two states: **on** or **off**. The game begins with a random arrangement of on or off buttons, as seen in a hypothetical starting configuration below:

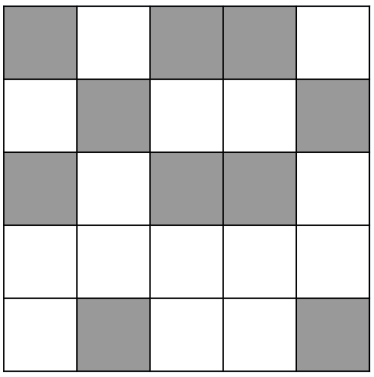

Figure 1: This picture above shows a randomly generated game from `https://www.logicgamesonline.com/lightsout/`

From this state, a move consists of pressing a button, which toggles the state of the button and all its immediate vertical and horizontal neighbors (it's important to note that we aren't changing the state diagonally). Ultimately, the goal is to turn all the lights off.

Initial Insights

1. First, we can see that pressing any button twice returns the button to its original state, meaning that each button must only be pressed once. This simplifies the problem to binary operations: either a button is pressed (we can represent this as 1) or it is not (we can represent this as 0).

2. Next, we can observe that the final configuration depends only on the total number of presses for each button, rather than the specific sequence in which the buttons are pressed. Therefore, we can conclude that the order of moves does not affect the final outcome.

Mathematical Representation

In the classic Lights Out game, the 5 by 5 grid of lights can be represented as a matrix where each entry corresponds to the state of a specific light:

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{pmatrix}$$

Each light state can be in one of two states which is represented by: 1 for on and 0 for off.

To use Linear Algebra ideas for this problem, we can convert this 5 by 5 grid into a single-column vector b. This process involves "flattening" the matrix by stacking each row on top of the next, resulting in a 25-element vector:

$$b = \begin{pmatrix} b_{1,1} \\ b_{1,2} \\ b_{1,3} \\ b_{1,4} \\ b_{1,5} \\ b_{2,1} \\ b_{2,2} \\ \vdots \\ b_{5,4} \\ b_{5,5} \end{pmatrix}$$

Now, let's let x be a 25-element column vector representing the buttons

pressed:

$$x = \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{5,5} \end{pmatrix}$$

where each $x_{i,j}$ is either 0 or 1.

The system's behavior is modeled using a 25×25 matrix A, where each entry a_{ij} represents the effect of pressing button j on the configuration of the i-th component of the system. This matrix structure arises because the system has 25 components, and each button press can affect all components, resulting in a total of 25×25 interactions.

Let x denote a 25×1 column vector specifying the number of times each button is pressed (representing the chosen strategy), and let b represent a 25×1 column vector describing the resulting configuration of the system. The relationship between these elements is expressed as:

$$A \cdot x = b$$

To determine whether a configuration b is winnable, we can analyze the system $b = Ax$. A configuration b is winnable if and only if it belongs to the column space of the matrix A, denoted by $\text{Col}(A)$.

Steps to analyze Col(A):

1. Do Gauss-Jordan elimination on A (or in other words, row reduction) to reduce it to its row echelon form E. This result is $RA = E$, where E is the reduced form of A, and R is the product of the elementary matrices that perform the row-reducing operations.

2. We can point out that A is symmetric, so $\text{Col}(A)$ equals $\text{Row}(A)$. Since $\text{Row}(A)$ is the orthogonal complement of $\text{Null}(A)$, and $\text{Null}(A) = \text{Null}(E)$, the task reduces to finding a basis for $\text{Null}(E)$.

3. If we do this calculation, we can see that the matrix E is of rank 23, leaving two free variables corresponding to the last two columns of E. These columns form the basis for $\text{Null}(E)$.

   Let's take a closer look at this:

   To express the system of equations in matrix form, we can write it as $Ax = b$, where A is a 25×25 block matrix like this:

$$A = \begin{bmatrix} Z & I & 0 & 0 & 0 \\ I & Z & I & 0 & 0 \\ 0 & I & Z & I & 0 \\ 0 & 0 & I & Z & I \\ 0 & 0 & 0 & I & Z \end{bmatrix}.$$

   In the above, I is the 5 by 5 identity matrix, and 0 represents the 5 by 5 matrix with all zero. Z is a 5 by 5 matrix that shows how different

parts of a system are connected. Each row in Z tells us how one element interacts with its neighbors:

$$Z = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

To solve the system, we explore the null space of A. Like we explained above, the null space of A can be computed as:

$$\text{Null}(A) = \text{Null}(E).$$

For $\vec{b}$ to have a solution, it must be orthogonal to every vector in the null space of A. This means $\vec{b}$ must belong to the orthogonal complement of the null space, denoted $\text{Null}(E)^{\perp}$.

To determine the null space of E, we examine its reduced structure and express the relationships between its variables. For instance, solving the equations derived from E gives relationships such as:

$$\begin{aligned} 0 &= x_1 + x_{25}, \\ 0 &= x_2 + x_{24}, \\ 0 &= x_3 + x_{24} + x_{25}, \\ &\vdots \\ 0 &= x_{23} + x_{24} + x_{25}. \end{aligned}$$

From this, we can see patterns in the dependencies among variables. These can be used to construct a basis for the null space of E. The general solution reveals that the variables x_{24} and x_{25} are free, while the other variables are defined in terms of them. Thus, the null space of E is spanned by vectors corresponding to these dependencies.

From this, we can say that a configuration b is winnable if and only if b is orthogonal to all basis vectors of the null space, denoted here as n_1 and n_2. These vectors n_1 and n_2 form a basis for the null space of the matrix A. This gives us the steps below to determine the winnability:

1. Find the dot products $b \cdot n_1$ and $b \cdot n_2$.
2. If both dot products are zero, then b is winnable. Otherwise, it is not.

Furthermore, if b is a winnable configuration with a winning strategy x, then $x+n_1$, $x+n_2$, and $x+n_1+n_2$ are also valid winning strategies. This is because, $A(x+n_i) = Ax + An_i = b + 0 = b$, as $An_i = 0$ for any n_i in the null space.

Divide and Conquer Method

At a high level, this strategy involves iteratively attempting different sequences of button presses until a solution is found. In Computer Science terms, we can look at this as a form of a "divide and conquer" approach to a problem.

Essentially, the "under the hood" solution is to divide the grid into smaller sections, progressively narrowing down the possible configurations. Specifically, we split the 5×5 array into a 1×5 array and a 4×5 array. Our goal is to focus on turning off the lights in the 1×5 array first, and then proceed to the next section, the 4×5 array.

Here are the steps more formally:

- Work on turning off all the lights in the 1×5 array (the top row).
- After handling the top row, move on to the remaining 4×5 array, and repeat the process (proceed to the second row).
- Continue separating the remaining arrays until only a 1×5 array remains (repeat the process—continue with the third and subsequent rows).

Handling the Bottom Row

However, there may be an issue. After finishing the process for the top rows, there may still be some lights left on in the bottom row. If this happens, we know that the lights in the bottom row can only be in one of seven possible configurations. Given the configuration of the lights in the bottom row, we can figure out which buttons need to be pressed to turn off the remaining lights. We can reference Figure 2 below for the specifications.

Overall, this strategy this poses some issues, as we may be pressing more buttons than we need to. We won't find the optimal solution to solving the board.

The Binary Method

I think that another interesting approach involves treating the five buttons in the top row as a five-digit binary number, starting at zero and incrementing by one with each subsequent press. Begin by pressing the buttons according to the binary representation 00000, then proceed with 10000, 01000, 00100, and so on, where a "1" signifies pressing the corresponding button, just like before. This method reveals that there are exactly 32 distinct ways to press the buttons in the top row.

But, there poses some issues with this method as well. One of the significant issues of this method is that there is no way to immediately check if the puzzle is solvable. Without going through all 32 combinations, it is impossible to determine if a solution exists for the given configuration.

And additionally, while the binary method offers a systematic way to attempt solutions, it is not the most efficient, especially when dealing with larger or more complex grids.

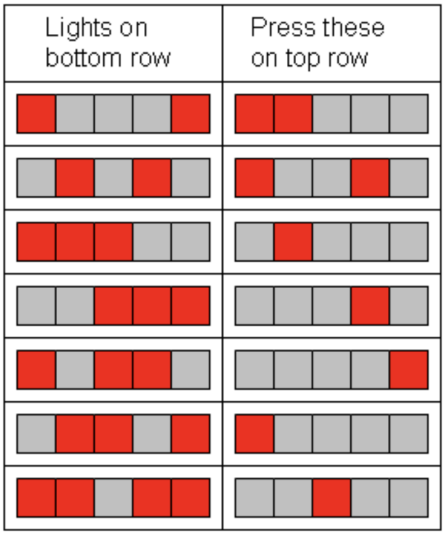

Figure 2: This picture above shows the seven possible configurations and the buttons that should be pressed to turn off the remaining lights. This is borrowed from http://cau.ac.kr/ mhhgtx/courses/LinearAlgebra/references/MadsenLightsOut.pdf

Some Insights

- **Does every Lights Out game have a solution?** I think it's important to note that not every Lights Out game is solvable and it depends on the toggle matrix. Whether a configuration is solvable depends on the linear algebraic properties of the game matrix A and the initial state.
- **What if we didn't have a 5×5 matrix, but an $n \times n$ matrix?** If the board is not a 5×5 matrix but an $n \times n$ matrix, we still carry out the same process. The main difference lies in the dimension of the null space of the corresponding matrices, which varies depending on the value of n. Regardless of whether the board is a $n \times n$ grid, a circular layout, or even a randomly scattered arrangement, the game remains playable as long as you know which buttons toggle when a given button is pressed. Thus, the underlying principles of the Lights Out game can be applied to any configuration as long as the toggle rules are defined.
- **What about Lights Out with more than one state?** Something interesting to look at is when Lights Out can be extended to cases where each light has more than two states. We can represent these states by numbers in $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, where p is a prime number, and 0 means the light is off. When $p > 2$, this variation introduces some interesting properties.

The Problem Representation

The Lights Out problem can be represented by the equation:

$$S_m \mathbf{p}_1 = -\mathbf{s}_b,$$

where:

- S_m is a matrix describing how each switch affects the lights.
- $\mathbf{p}_1$ is a vector representing the current states of the lights.
- $\mathbf{s}_b$ is a vector representing the desired states of the lights.

The goal is to find the vector $\mathbf{p}_1$, which tells us which switches to press to achieve the target configuration. The $-$ sign in $-\mathbf{s}_b$ accounts for the changes needed to transition from the current state to the target state.

When a button is pressed, it increases the state of a light (and possibly its neighbors) by 1 modulo p. And just like before, the order of button presses does not affect the final result, and each button needs to be pressed no more than $p-1$ times (or else it will just give us the same state).

Constructing the Matrix S_m

The matrix S_m for an $m \times m$ grid can be constructed recursively like this:

$$S_{m+1} = \begin{bmatrix} A & I \\ 0 & S_m \end{bmatrix},$$

where:

- A is the adjacency matrix that describes how the lights are connected.
- I is the identity matrix, representing the influence of directly connected rows.
- 0 is a matrix of zeros, indicating no influence between unrelated rows.

For $p > 2$, we can use the following formula (developed from `https://www.jstor.org/stable/10.4169/math.mag.90.2.126?seq=1#page_scan_tab_contents` which creates a closed form formula using Fibanocci polynomials) to calculate S_m:

$$S_m = \sum_{i=0}^{\lfloor m/2 \rfloor} (-1)^{m-i+1} \binom{m-i}{i} A^{m-2i}.$$

Let's Look at an Example: A 5×5 Grid in $\mathbb{Z}_3$: Let's consider a 5×5 grid with states in $\mathbb{Z}_3$. The matrix A remains structured as before but

now has entries from $\mathbb{Z}_3$, as lights can be in the 0, 1, or 2 state. Using the formula above, we can calculate S_5:

$$S_5 = \begin{bmatrix} 2 & 1 & 1 & 1 & 2 \\ 1 & 0 & 2 & 0 & 1 \\ 1 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 0 & 1 \\ 2 & 1 & 1 & 1 & 2 \end{bmatrix}.$$

If the desired bottom row configuration is, let's say, $\mathbf{s}_b = [1, 0, 2, 0, 1]$, we can solve $S_5\mathbf{p}_1 = -\mathbf{s}_b$ by adding $-\mathbf{s}_b = [2, 0, 1, 0, 2]$ modulo 3.

Building a Lookup Table

The column space of S_5 contains all solvable configurations for the bottom row. If $\text{rank}(S_5) = 2$, there are $3^2 = 9$ distinct configurations, expressible as linear combinations of two basis vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. These combinations generate all possible bottom-row configurations:

$$\mathbf{s}_b = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 \mod 3,$$

where $c_1, c_2 \in \mathbb{Z}_3$.

We can define a lookup table, as a precomputed mapping that simplifies solving the Lights Out problem. It is particularly useful for determining configurations of the bottom row in terms of the column space of the matrix S_m. The lookup table lists all possible values of $\mathbf{s}_b$ by iterating through every combination of c_1 and c_2. For each combination, the corresponding top-row configuration $\mathbf{p}_1$ is determined.

Example

Consider the example provided in the main text for a 5×5 grid with $p = 3$. The lookup table would contain all 9 possible configurations for $\mathbf{s}_b$, along with the corresponding $\mathbf{p}_1$. Each entry explicitly connects c_1 and c_2 to the bottom row $\mathbf{s}_b$ and the top-row solution $\mathbf{p}_1$.

c_1	c_2	Bottom Row	Top Row
0	0	00000	00000
0	1	01010	22000
0	2	02020	11000
1	0	10201	02000
1	1	11211	21000
1	2	12221	10000
2	0	20102	01000
2	1	21112	20000
2	2	22122	12000

Extensions to 3D or Higher Dimensions

Another interesting idea is extending the Lights Out game beyond the traditional 2D grid to three or more dimensions, which introduces a different perspective.

3D Grid Structure

In a three-dimensional (3D) version of Lights Out, the game grid is represented as an $n \times n \times n$ cube, where each cell has a state that can change based on interactions with neighboring cells. The following are ideas that I have come up with, after more research on data structures and algorithms. These ideas are just reflections of my own thoughts and research and are not backed by sources.

- In a 3D grid, we can say that each cell has six immediate neighbors—front, back, left, right, above, and below. As we saw, this differs from the 2D case, where each cell has only four neighbors. Thus, toggling a cell affects more neighbors, increasing the difficulty of predicting and controlling the state of the grid. Mathematically, the impact of toggling a single cell can be represented by extending adjacency matrices—an idea in graph theory and computer science that I learned about in my Algorithms class (CMSC351).

  ### Adjacency Matrix for a 3D Grid

  The adjacency matrix A_{3D} can describe how each cell affects its neighbors. Now, if we say that the total number of cells is $N = n^3$, A_{3D} is an $N \times N$ matrix, where:

  $$(A_{3D})_{ij} = \begin{cases} 1 & \text{if cell } i \text{ and } j \text{ are neighbors,} \\ 0 & \text{otherwise.} \end{cases}$$

  For a small $2 \times 2 \times 2$ grid, the adjacency matrix A_{3D} is:

  $$A_{3D} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

- After some research, I came across the concept of a sparse tensor (`https://www.tensorflow.org/guide/sparse_tensor`), which can be useful for representing 3D neighbor relationships efficiently. A sparse tensor is a

multi-dimensional array where most entries are zero, making it a memory-efficient way to represent large datasets with sparse interactions. For instance, instead of storing all possible neighbor interactions for a $n \times n \times n$ grid explicitly, we only store the non-zero entries that correspond to actual neighbor relationships. This approach is particularly useful in computational applications where minimizing memory usage and computational complexity is very important.

Using this idea, we can represent the 3D Lights Out game in matrix form by constructing a linear system over $\mathbb{Z}_p$. For a grid with n^3 cells, this system requires solving an $n^3 \times n^3$ linear system. The adjacency matrix A is replaced by a sparse tensor T capturing neighbor relationships in three dimensions. We started with an adjacency matrix because it's an easier way to visualize the neighbor connections. By transferring this data to a more computationally intensive representation, we are able to save memory (by storing only non-zero entries) and optimize the time. Each entry of T defines the influence of toggling one cell on another.

The system of linear equations can then be formulated as:

$$S_n \mathbf{p} = -\mathbf{s}_b \quad \text{over} \quad \mathbb{Z}_p,$$

where S_n represents the toggle matrix extended to 3D, and $\mathbf{p}$ and $\mathbf{s}_b$ denotes the button press and target state vectors, respectively. This sparse tensor approach ensures that the computations remain efficient and scalable even for larger grids.

- Another thing to keep in mind is that as the grid size n grows, the total number of cells increases cubically, meaning the system becomes much larger to compute. For example, if $n = 3$, the total number of cells is $3^3 = 27$, but if $n = 10$, the total grows to $10^3 = 1000$ cells (which is quite a lot). This rapid growth makes it harder to solve the game manually or with basic tools.

  To handle this, we can use more advanced software tools like TensorFlow or PyTorch (two popular and primarily Python open-source deep learning frameworks). These tools are commonly used in areas like artificial intelligence and can perform complex mathematical calculations very quickly. By using these libraries, we can solve larger versions of the game much more efficiently, even on a computer.

Further Applications

After researching Lights Out, I wanted to understand other ways (or more real-world ways) in which these ideas can be used. The main idea in Lights Out is that an action or change in one component influences neighboring components. I came up with a couple of analogies to Lights Out:

Electrical Grids

In an electrical grid, each power station can be thought of as a node connected to neighboring nodes through transmission lines. If one node is toggled (activated or deactivated), it can influence the power distribution and load on neighboring nodes:

- If one station fails, it can lead to an overload in adjacent stations, leading to a chain reaction similar to the toggle propagation in Lights Out.
- Ways to restore power after a blackout involve system-wide changes, which in a way is synonymous to finding the sequence of moves to turn off all lights.

Social Networks

Each button in the game can be viewed as a person in a social network, and pressing a button can represent a person spreading an idea, trend, or even just some piece of information.

- In social influence models, a change in behavior by one person impacts their immediate connections (or even neighbors), much like toggling the state of a button in Lights Out.
- Figuring out how to stop a network from getting out of control by turning certain buttons on or off might help us come up with ways to stop misinformation or rumors from spreading.

Biological Systems

Neural networks (from a biological perspective, not a computer science sense) also have qualities similar to those of Lights Out.

- When a neuron "fires" (neurons communicating with each other), it influences its adjacent neurons. We can try mapping this to a Lights Out scenario, which can help us understand how activation spreads through neural connections.
- Switching (or toggling, like in Lights Out) certain genes on or off can change how other genes behave, which can then affect bigger biological processes.

Thoughts and Reflections

Working on this project has been an eye-opening journey into the intersection of mathematics, computer science, and my love for games/board games. Initially, I approached the problem with a mindset focused solely on solving puzzles. However, diving deeper into the mathematical structures behind Lights Out

showed me the elegance and beauty in some ways of linear algebra in modeling and solving problems that I love.

I'm curious to see how these mathematical techniques extend to other more complex games, such as chess, where strategies are deeply rooted in combinatorial game theory, or Go, where pattern recognition and spatial reasoning play a big role. I'm also interested in exploring how these techniques can be applied to multiplayer games with more dynamic strategies, like poker, where probability and decision-making under uncertainty are very important. The potential to model and optimize strategies in these kinds of games is an exciting direction for future areas I want to learn more about.

Works Cited

- https://people.sc.fsu.edu/~jburkardt/classes/imps_2017/11_28/2690705.pdf
- http://cau.ac.kr/~mhhgtx/courses/LinearAlgebra/references/MadsenLightsOut.pdf
- https://www.jstor.org/stable/10.4169/math.mag.90.2.126?seq=8#page_scan_tab_contents