Lecture 24: Lights Out Puzzle

Contents

24.1 Lights Out 1
24.2 Lights Out: A Matrix Model 2
24.2.1 Imagine you are in a field... 5
24.2.2 Solving linear systems with SAGE. 6
24.2.3 Solvable Configurations 8
24.2.4 Optimal solution to Lights Out 10
24.3 Summary of 5×5 lights out puzzle 10
24.4 Other sized game goards 12
24.5 Light-Chasing Strategy 13
24.6 Exercises 13

In this lecture we look at an electronic puzzle called *Lights Out* and see how we can solve it using linear algebra.

24.1 Lights Out

The game consists of a 5-by-5 grid of lights; when the game starts, a set of these lights (random, or one of a set of stored puzzle patterns) are switched on. Pressing one of the lights will toggle it, and the four lights adjacent to it, on and off. (Diagonal neighbours are not affected.) The game provides a puzzle: given some initial configuration where some lights are on and some are off, the goal is to switch all the **lights off**, preferably in as few button presses as possible. See Figure 1 for sample game play.

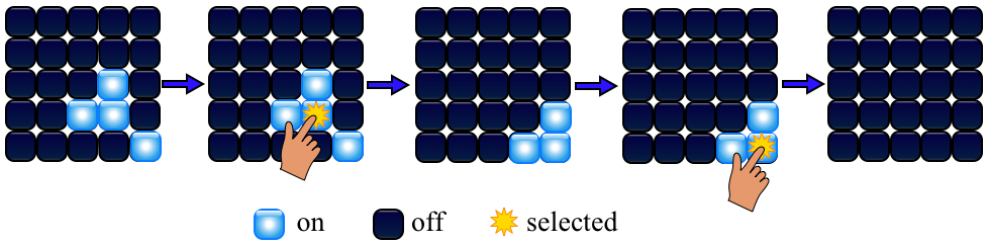
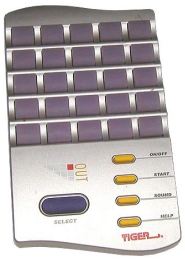


Figure 1: A demonstration of Lights Out play.

Two physical versions of the game are shown in Figure 2. The first one is the original game, each button has two states: on or off. The second one, called Lights Out 2000, has a further option of allowing 3 states for each button: red, green and off.



(a) Lights Out

(b) Lights Out 2000

Figure 2: Lights Out electronic games released by Tiger Toys

Variations of Lights Out: Lights Out is another puzzle that has been updated for the digital era. Many variations of this puzzle exist now in software form. Variations include: more states for the lights (i.e. more colours for the lights to cycle through), changing size of game boards, modifying how a button press changes the state of the lights. For example, we could make it so pressing a button changes the state of all lights in the same row and column as the button that was pressed.

Software: This puzzle is available for play on the web. There are also some versions available for the ipod. More information about where to find your own digital copy can be found in the software section of our course webpage.

24.2 Lights Out: A Matrix Model

A complete strategy for the game can be obtained using linear algebra, requiring only knowlege of Gauss-Jordan elimination and some facts about column and null space of a matrix.

We make some initial observations:

(a) Pushing button twice is is equivalent to not pushing it at all.

(b) The state of a button depends only on how often (whether even or odd) it and its neighbours have been pushed. Hence, the order in which the buttons are pressed does not matter. Together with (a), for any configuration, a solution exists in which each button is pushed no more than once.

We will represent the state of each light by an element of $F_2 = \{0, 1\}$; 1 for on, 0 for off. We can represent a lit button configuration by a 5×5 matrix A with entries from F_2, i.e. $A \in M_{5\times5}(F_2)$ where the $(i, j)^{\text{th}}$ entry is 1 if the button is on, or 0 if the button is off. See Figure 3. We call this matrix the lit button **configuration matrix**. Here,

$$M_{5\times5}(F_2) = \{[b_{i,j}] \mid 1 \leq i, j \leq 5, b_{i,j} \in F_2 = \{0, 1\}\}.$$

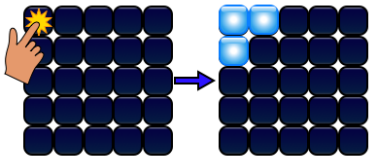
(a) sample lit button configuration

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

(b) corresponding configuration matrix

Figure 3: Matrix corresponding to a lit button configuration

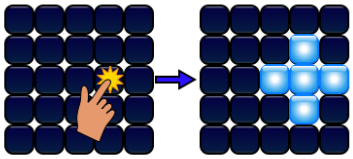
If a button is pressed the states of the lights around the button are toggled. For the standard lights out puzzle it is the button itself, and its vertical and horizontal neighbours that are toggled. For each button (i, j) we define a **toggle matrix** $T_{i,j}$ where the entry is 1 if the button in that location changes state, or 0 if it doesn't. For example, see Figure 4.



(a) pressing button $(1, 1)$

$$T_{1,1} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

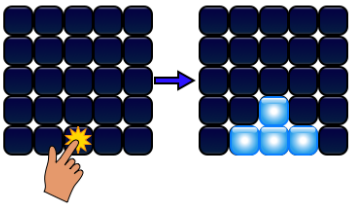
(b) corresponding toggle matrix $T_{1,1}$



(c) pressing button $(3, 4)$

$$T_{3,4} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(d) corresponding toggle matrix $T_{3,4}$



(e) pressing button $(5, 3)$

$$T_{5,3} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

(f) corresponding toggle matrix $T_{5,3}$

Figure 4: Some examples of the toggle matrix corresponding to pressing a button.

The sample game play shown in Figure 1 can be translated into a matrix equation using configuration and toggle matrices as follows.

Let B be the initial configuration matrix:

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Then the game play corresponds to

$$B + T_{4,4} + T_{5,5} = \mathbf{0},$$

where $\mathbf{0}$ is the zero matrix, and addition of matrices is done in the usual way – componentwise – but here entries are added modulo 2. Recall, modulo 2 arithmetic means $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$, $1 + 1 = 0$. Since a matrix in $M_{5\times5}(F_2)$ added to itself is $\mathbf{0}$ then adding B to both sides of the previous equation gives:

$$T_{4,4} + T_{5,5} = B.$$

In other words, to solve the puzzle we just have to determine how to write B as a linear combination of the toggle matrices.

Moreover, since for any matrices $A, C \in M_{5\times5}(F_2)$ we have $A+C = C+A$ and $A+A = \mathbf{0}$ then we can now easily see why (i) order in which buttons are presses doesn't matter, and (ii) no button needs to be pressed more than once.

In general, given any lit button configuration $B = [b_{i,j}]$, solving the puzzle is equivalent to solving the matrix equation:

$$\sum_{\substack{1 \le i \le 5 \\ 1 \le j \le 5}} x_{i,j} T_{i,j} = B \tag{1}$$

for the 25 coefficients $x_{i,j} \in \{0, 1\}$. The coefficients $x_{i,j}$ tell use exactly what buttons we need to press. We call $X = [x_{i,j}]$ the **strategy matrix**. We will sometimes write it as a vector $\boldsymbol{x} = (x_{1,1}, x_{1,2}, x_{1,3}, \ldots, x_{4,5}, x_{5,5})$ and call it the **strategy vector**. In general, we can turn any matrix into a vector by listing the entries in order from left-to-right, then top to bottom.

Matrix equation (1) corresponds to a system of $5 \cdot 5 = 25$ linear equations (one for each component of the matrix equation).

For example, the linear equation corresponding to entry $(1, 1)$ in matrix equation (1) is

$$x_{1,1} + x_{1,2} + x_{2,1} = b_{1,1},$$

since the only toggle matrices with 1 in position $(1, 1)$ are $T_{1,1}, T_{1,2}$, and $T_{2,1}$. Similarly, the linear equation corresponding to entry $(3, 4)$ in matrix equation (1) is

$$x_{2,4} + x_{3,3} + x_{3,4} + x_{3,5} + x_{4,4} = b_{3,4}.$$

Writing $\boldsymbol{b} = (b_{1,1}, b_{1,2}, b_{1,3}, \ldots, b_{4,5}, b_{5,5})$ for the vector corresponding to the configuration matrix B, it is straightforward to check that this big system (Equation (1)) can be written as a matrix product

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2}$$

where A is the 25×25 matrix whose columns are the toggle vectors (which we also denote by $T_{i,j}$): $A = [T_{1,1} \mid T_{1,2} \mid \cdots \mid T_{5,5}]$. We can write A as

$$A = \begin{pmatrix} C & I_5 & 0 & 0 & 0 \\ I_5 & C & I_5 & 0 & 0 \\ 0 & I_5 & C & I_5 & 0 \\ 0 & 0 & I_5 & C & I_5 \\ 0 & 0 & 0 & I_5 & C \end{pmatrix} \qquad \text{(lights out matrix)} \tag{3}$$

where C represents the 5×5 matrix

$$C = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix},$$

and I_5 denotes the 5×5 identity matrix. The matrix A is referred to as the **lights out matrix**.

Therefore, solving the puzzle for a general configuration $\boldsymbol{b}$ is equivalent to solving the 25×25 linear system 2 for a strategy vector $\boldsymbol{x}$ (where all arithmetic is done modulo 2).

We would like to know the answers to the following questions.

(a) Will the standard algorithm using Gauss-Jordan elimination work to solve this system? Recall, this method works if entries are real numbers under regular addition/multiplication. But here we are working over a different number system: $F_2 = \{0, 1\}$ under addition/multiplication modulo 2.

(b) Must there be a solution for every configuration $\boldsymbol{b}$?

(c) If not, what is the probability a random configuration is solvable?

(d) When there is a solution for $\boldsymbol{b}$, is it unique? If not, can we find the smallest solution (i.e. giving the least number of button presses)?

24.2.1 Imagine you are in a field...

The algorithm learned in linear algebra for solving linear systems of the form $A\boldsymbol{x} = \boldsymbol{b}$ is known as Gauss-Jordan elimination (or simply as Gaussian elimination). The steps of the algorithm are as follows:

(a) Form the augmented matrix $[A|\boldsymbol{b}]$.

(b) Reduce the augmented matrix to *reduced row echelon form* by using elementary row operations:

- (swap) Swap any two rows.
- (scalar multiply) Multiply any row by a non-zero number.
- (replacement) Replace any row with a multiple of another row added to the row itself.

(c) Read off the solution (or conclude there isn't a solution) directly from the reduced row echelon form.

In linear algebra we strictly used the real numbers $\mathbb{R}$ under addition/multiplication. If you were lucky you saw that the same thing could be done with complex numbers $\mathbb{C}$. We'd like to know, does all the theory developed in linear algebra carry over to more abstract sets of "numbers" under some sort of "addition" and "multiplication"? In particular what about the situation we are in with the lights out puzzle. Here our number system is

$$F_2 = \{0, 1\}$$

and the addition and multiplication tables are defined as follows. [1]

+	0	1
0	0	1
1	1	0

*	1
1	1

Does Gauss-Jordan elimination still work?

Let's consider a set F of objects which is closed under two operations $+$ and $*$. What properties would $(F, +, *)$ need to satisfy in order for Gauss-Jordan elimination to still possibly work?

First note the key to having this algorithm work is that the elementary row operations must be reversible. Clearly a row swap is reversible, just swap the rows back. Multiplying a row by a nonzero element is only reversible if the element has a multiplicative inverse in F. Therefore, the set $F^* = F - \{0\}$ should be a group under $*$. Also, another key part to the algorithm was that we could use additive inverses to make entries of the matrix 0. This means F should be a group under $+$.

We call a set F with two operations $+$ and $*$ a **field** if the following properties are satisfied:

(a) F is an abelian group under $+$.

(b) $F^* = F - \{0\}$ is an abelian group under $*$

(c) $a(b + c) = ab + ac$ and $(b + c)a = ba + ca$. (distributive law)

It turns out these were the only properties of $(\mathbb{R}, +, *)$ we used in linear algebra. Therefore, everything done in linear algebra holds true for matrices whose entries come from any field F.

Since F_2 is a field with two elements then Gauss-Jordan elimination will work to solve the linear system. Moreover, any result we want to use from linear algebra will carry over to this new setting where our "numbers" come from F_2.

[1] We left 0 out of the multiplication table since $0(a) = 0$ for any a.

24.2.2 Solving linear systems with SAGE.

In a first course in linear algebra you were typically asked to solve linear systems by-hand. This was to allow you to understand the details of the Gauss-Jordan elimination algorithm. In practice, people don't generally solve systems of equations by hand, these are generally done by computer. We'll now see how to use SAGE to solve linear systems.

To solve a linear system $Ax = b$ in SAGE, we must first define the matrix A, for example

`matrix(ZZ,[[1,2],[3,4],[5,6])` defines the matrix $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$, over the integers $\mathbb{Z}$. Here we defined each row. We can give SAGE a list and tell it how many rows, then have it split the list into a matrix as follows:

`matrix(QQ,2,[1,2,3,4,5,6])` defines the matrix $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, over the rationals $\mathbb{Q}$,

Here is an example using SAGE to solve the system

$$\begin{pmatrix} 1 & 0 & 2 \\ 3 & 2 & 5 \end{pmatrix} x = \begin{pmatrix} 3 \\ 0 \end{pmatrix}.$$

SAGE

```
sage: M=matrix(QQ,2,[[1,0,2],[3,2,5]])
sage: b=vector(QQ,[2,0])
sage: M.solve_right(b)          #command for solving Mx = b  (i.e. x is right of M)
(2, -3, 0)
```

The command for solving a linear system $A\boldsymbol{x} = \boldsymbol{b}$ is `A.solve_right(b)`. [2]

Coming back to lights out, we first need to construct the lights out matrix A defined in (3). We could do it one entry at a time, which would involve entering $25 \cdot 25 = 625$ numbers. This wouldn't be fun, and if we want to consider larger game boards than 5×5 we would have a lot more typing to do. Instead, we use two loops to define A, and we do this for a general $n \times n$ board. Keep in mind, we have to tell SAGE we are working over the field of integers modulo 2, F_2. SAGE knows this field by the name $GF(2)$, which stands for *Galois Field of size* 2.

SAGE

```
sage: # Definition of the matrix for Lights Out
sage: # input = integer n (where lights out board is nxn)
sage: # output = lights out matrix A which is nxn
sage: def lights_out(n):
sage:     M = MatrixSpace(GF(2),n*n,n*n)   # tells SAGE to work with matrices in M_n(F_2)
sage:     A = M.matrix()      # initializes A to a matrix in M, we then define entries below
sage:     for i in range(n):
sage:         for j in range(n):
sage:             m = n*i+j
sage:             A[(m,m)] = 1
sage:             if i > 0 : A[(m,m-n)] = 1
sage:             if i < n-1 : A[(m,m+n)] = 1
sage:             if j > 0 : A[(m,m-1)] = 1
sage:             if j < n-1 : A[(m,m+1)] = 1
sage:     return A
```

For example the lights out matrix for the 3×3 game board is

[2] Using the word "left" would be the command to solve $\boldsymbol{x}A = \boldsymbol{b}$, but in this case $\boldsymbol{x}$ and $\boldsymbol{b}$ would be row vectors, not column vectors.

SAGE
```
sage: lights_out(3)

[1 1 0 1 0 0 0 0 0]
[1 1 1 0 1 0 0 0 0]
[0 1 1 0 0 1 0 0 0]
[1 0 0 1 1 0 1 0 0]
[0 1 0 1 1 1 0 1 0]
[0 0 1 0 1 1 0 0 1]
[0 0 0 1 0 0 1 1 0]
[0 0 0 0 1 0 1 1 1]
[0 0 0 0 0 1 0 1 1]
```

Asking for the lights out matrix for the 5×5 game returns confirmation it is stored in memory, but SAGE saves us from having to look at it.

SAGE
```
sage: lights_out(5)
25 x 25 dense matrix over Finite Field of size 2
```

Now that A is loaded into SAGE let's solve some configurations.

Example 24.1 *Solve the following configuration:*

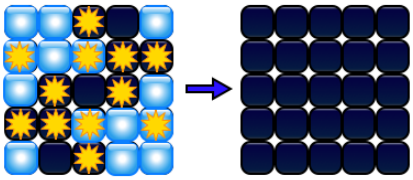
The configuration matrix is $B = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$ *which we can express as a vector*

$$\boldsymbol{b} = (1,1,0,0,1,\ 1,1,1,0,0,\ 1,0,0,0,1,\ 0,0,1,1,1,\ 1,0,0,1,1).$$

(Spaces are inserted after each group of 5 *entries in* $\boldsymbol{b}$ *just so it is easier for us to read.) Now we have* SAGE *solve* $A\boldsymbol{x} = \boldsymbol{b}$.

SAGE
```
sage: #current game configuration (i.e. buttons that are lit)
sage: b=vector(GF(2),[1,1,0,0,1, 1,1,1,0,0, 1,0,0,0,1, 0,0,1,1,1, 1,0,0,1,1]);

sage: #solving the game
sage: x=lights_out(5).solve_right(b);

sage: #now lets put the solution x in a nice matrix form so we can see what buttons to press
sage: button_press_matrix = matrix(GF(2),5,5,x.list())    # convert vector to a matrix
sage: button_press_matrix                                 # show matrix in output

[0 0 1 0 0]
[1 0 1 1 1]
[0 1 0 1 0]
[1 1 1 0 1]
[0 0 1 0 0]
```

Therefore, to solve the puzzle we just need to press the 12 *buttons shown in the diagram below.*



Rather than have to type out the previous lines of code every time we want to solve a configuration we could build a *solve* function as follows:

Lights-Out Solve function: (basic version)

SAGE

```
sage: # Definition of the solution function for Lights Out
sage: # input = integer n (where lights out board is nxn), and b the configuration vector
sage: # output = a matrix X indicating which buttons to press for solution
sage: def lights_out_solver(n,b):
sage:     x=lights_out(n).solve_right(b);
sage:     button_press_matrix = matrix(GF(2),n,n,x.list())
sage:     return button_press_matrix
```

For our previous example we could just type:

SAGE

```
sage: b=vector(GF(2),[1,1,0,0,1, 1,1,1,0,0, 1,0,0,0,1, 0,0,1,1,1, 1,0,0,1,1]);
sage: lights_out_solver(5,b)
[0 0 1 0 0]
[1 0 1 1 1]
[0 1 0 1 0]
[1 1 1 0 1]
[0 0 1 0 0]
```

24.2.3 Solvable Configurations

A lit button configuration $\boldsymbol{b}$ is solvable if the corresponding linear system $A\boldsymbol{x} = \boldsymbol{b}$ has a solution. From linear algebra we know

$$A\boldsymbol{x} = \boldsymbol{b} \text{ is solvable for } \textit{every } \boldsymbol{b} \iff A \text{ is invertible} \iff \det(A) \neq 0.$$

The lights out matrix (for 5×5 game) has determinant 0. Therefore, there exist unsolvable configurations $\boldsymbol{b}$.

SAGE

```
sage: lights_out(5).determinant()
0
```

For example, the configuration in Figure 5 is unsolvable.

SAGE

```
sage: b=vector(GF(2),[1,0,1,0,1, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0]);
sage: lights_out_solver(5,b)

Traceback (click to the left of this block for traceback)
...
ValueError: matrix equation has no solutions
```

Figure 5: An unsolvable configuration of lights.

Recall that $A\boldsymbol{x} = \boldsymbol{b}$ has a solution only when $\boldsymbol{b}$ is in the column space of A, denoted $\text{col}(A)$. This is just a fancy way of saying

$$\sum_{1 \le i,j \le 5} x_{i,j} T_{i,j} = \boldsymbol{b}$$

for some $x_{i,j}$, where $T_{i,j}$ are the toggle vectors, as we already know. However, phrased in this way we see that the set of solvable configurations is $\text{col}(A) = \text{span}(T_{1,1}, T_{1,2}, \ldots, T_{5,5})$, and the dimension of $\text{col}(A)$ is called the rank of A, denoted $\text{rank}(A)$.

SAGE

```
sage: lights_out(5).rank()
23
```

Therefore only 23 buttons are required to solve any configuration, and if each one can either be pressed or not, then there are 2^{23} solvable configurations, out of a possible 2^{25} configurations. This proves the following theorem.

Theorem 24.1 *For the 5×5 lights out puzzle, the probability that a random configuration is solvable is $1/4$.*

Quiet Patterns:
There exist sequences of button presses that will leave the lights unchanged. These are known as **quiet patterns**. Such a sequence $\boldsymbol{x}$ is a solution to the homogeneous equation $A\boldsymbol{x} = \boldsymbol{0}$. That is, $\boldsymbol{x}$ is in the null space of A, denoted by $\text{nul}(A)$. The dimension of this space is $\text{nullity}(A) = 25 - \text{rank}(A) = 25 - 23 = 2$. If we let $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ be a basis for $\text{nul}(A)$ then

$$\begin{aligned} \text{nul}(A) &= \text{span}(\boldsymbol{d}_1, \boldsymbol{d}_2) = \{r_1\boldsymbol{d}_1 + r_2\boldsymbol{d}_2 \mid r_1, r_2 \in F_2\} \\ &= \{\boldsymbol{0}, \boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_1 + \boldsymbol{d}_2\}. \end{aligned}$$

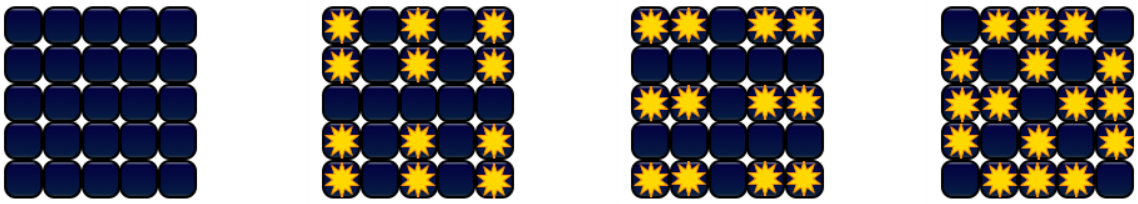
Therefore, there are only 4 such button sequences (vectors).

We can use SAGE to find these vectors. The command for computing the null space is `.right_kernel()`.

SAGE

```
sage: lights_out(5).right_kernel()
Vector space of degree 25 and dimension 2 over Finite Field of size 2
Basis matrix:
[1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1]
[0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0]
```

SAGE returns a basis for the nullspace. The span of these vectors (using coefficients from $F_2 = \{0, 1\}$) gives us the complete null space. These correspond to the button presses shown in Figure 6.



(a) $\mathbf{0}$ (b) $\boldsymbol{d}_1$ (c) $\boldsymbol{d}_2$ (d) $\boldsymbol{d}_1 + \boldsymbol{d}_2$

Figure 6: The 4 Quiet Patterns: These are the button press sequences in the nullspace of A.

24.2.4 Optimal solution to Lights Out

Let $\boldsymbol{b}$ be a (solvable) configuration of the lights. If $\boldsymbol{x}$ is a strategy vector (i.e. a solution to $A\boldsymbol{x} = \boldsymbol{b}$) then the set of *all* solution strategies is:

$$\boldsymbol{b} + \text{null}(A) = \{\boldsymbol{b}, \quad \boldsymbol{b} + \boldsymbol{d}_1, \quad \boldsymbol{b} + \boldsymbol{d}_2, \quad \boldsymbol{b} + \boldsymbol{d}_1 + \boldsymbol{d}_2\}.$$

The optimal solution will be the one with the fewest number of 1's as entries.

Let's go back to Example 24.1 and see if we can find an optimal solution. The one we found requires 12 button presses, perhaps we can do better.

It will be convenient to have SAGE count the number of occurrences of 1 in a strategy vector. We will define a function called `number_of_presses` to do this.

SAGE

```
sage: def number_of_presses(x):
sage:     counter=0;    # initialize counter, which is our variable to count 1's
sage:     for i in range(0,25):      # recall Python indexes lists from 0, not 1
sage:        if x[i]==1: counter=counter+1   # check if ith entry is 1
sage:    return counter
```

Now let's find all 4 solutions to Example 24.1.

SAGE

```
sage: b = vector(GF(2),[1,1,0,0,1, 1,1,1,0,0, 1,0,0,0,1, 0,0,1,1,1, 1,0,0,1,1]);
sage: x = lights_out(5).solve_right(b)   # one solution
sage: nulsp = lights_out(5).right_kernel()
sage: for d in nulsp:
sage:    print b+d, number_of_presses(b+d)
(0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0)  12
(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1)  8
(0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0)  8
(1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1)  20
```

There are two optimal solutions, each requiring 8 button presses. Therefore, an optimal solution to the configuration in Figure 7a is the strategy matrix in Figure 7b.

24.3 Summary of 5×5 lights out puzzle

Solving a configuration $\boldsymbol{b}$ of the lights out puzzle is equivalent to solving the linear system $A\boldsymbol{x} = \boldsymbol{b}$ for strategy vector $\boldsymbol{x}$ where A is a 25×25 lights out matrix. All arithmetic is done in the finite field of size 2: $F_2 = \{0, 1\}$.

(a) Configuration of lights

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) optimal strategy matrix

(c) optimal solution

Figure 7: An optimal solution requiring 8 button presses.

- $\text{rank}(A) = 23, \quad \text{nullity}(A) = 5^2 - \text{rank}(A) = 2$
- number of solvable configurations is $2^{\text{rank}(A)} = 2^{23}$
- probability that a random configuration is solvable is $2^{23}/2^{25} = 1/4$.
- number of quiet patterns (elements in the nullspace of A) is $|\text{nul}(A)| = |F_2|^{\text{nullity}(A)} = 2^2 = 4$.
- for a given strategy vector $\boldsymbol{x}$ the 4 equivalent vectors are the elements of $\boldsymbol{x} + \text{nul}(A)$.

Putting all the previous ideas into one code block, we can write a lights out solver which returns the optimal solution.

Lights-Out Solve function: (optimal version)

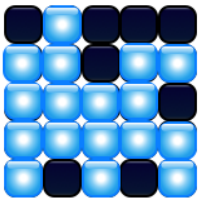
SAGE

```
sage: # Function:  number_of_presses
sage: # input = a vector x of dimension 25 with 0,1 entries
sage: # output = the number of times 1 appears as an entry
sage: def number_of_presses(x):
sage:     counter=0;
sage:     for i in range(0,25):
sage:         if x[i]==1: counter=counter+1
sage:     return counter

sage: # Function:  optimal_solution
sage: # input = a strategy vector x
sage: # output = an equivalent strategy vector which uses the least number of button presses
sage: def optimal_solution(x):
sage:     op_button_presses=x     # initialize variable to store optimal solution
sage:     n=number_of_presses(x)  # initial variable to store optimal presses
sage:     nul=lights_out(5).right_kernel()
sage:     for d in nul:
sage:         if number_of_presses(x+d)<n:
sage:             op_button_presses=x+d        # update variable
sage:            n=number_of_presses(x+d)      # update variable
sage:    return op_button_presses

sage: # Function:  lights_out_solver
sage: # input = b the configuration vector of lights on 5-by-5 game
sage: # output = an optimal strategy matrix which solves the puzzle
sage: def lights_out_solver(b):
sage:     x=lights_out(5).solve_right(b);  # one solution
sage:     x=optimal_solution(x)     # exchanges x for an optimal solution
sage:     button_press_matrix = matrix(GF(2),5,5,x.list())  #make output vector into a matrix
sage:     return button_press_matrix
```

As an example, to solve the configuration in Figure 8a we proceed as follows.

SAGE

```
sage: b=vector(GF(2),[0,1,0,0,0, 1,1,0,1,1, 1,1,1,1,0, 1,1,1,1,1, 1,0,1,0,1]);
sage: lights_out_solver(b)

[0 0 0 0 1]
[0 1 0 1 1]
[0 0 0 1 0]
[1 0 0 1 0]
[0 0 0 1 0]
```



(a) Configuration of lights

(b) optimal solution

Figure 8: An optimal solution requiring 8 button presses.

24.4 Other sized game goards

Lights Out has be modified and generalized in many ways: bigger games boards, different toggle conditions, more states (colours) for the lights to cycle through.

Here we mention briefly some results about larger games boards. We assume the toggling condition is the same as for the 5×5 game board. Let A_n be the lights out matrix for the $n \times n$ game board. The key to understanding solvability lies in knowing the whether the $n^2 - \text{rank}(A_n)$ is 0 or not. If it is 0 then A_n has full rank, and so it's columns are linearly independent, therefore A_n is invertible. This means *every* configuration is solvable. If it is non-zero then $\text{rank}(A_n) < n^2$ so A_n is not invertible, therefore there exist configuration which are not solvable. Moreover, the number of different solutions for a given configuration (if the configuration is solvable) is $2^{\text{nullity}(A_n)} = 2^{n^2 - \text{rank}(A_n)}$.

Table 1 lists the values of $n^2 - \text{rank}(A_n)$ for $3 \leq n \leq 10$.

n	$\text{rank}(A_n)$	$\text{nullity}(A_n) = n^2 - \text{rank}(A_n)$
3	9	0
4	12	4
5	23	2
6	36	0
7	49	0
8	64	0
9	73	8
10	100	0

Table 1: $\text{nullity}(A_n)$ and $\text{rank}(A_n)$ for various boards sizes of lights out.

24.5 Light-Chasing Strategy

There is a strategy for solving the 5×5 lights out puzzle which, though not optimal, will allow you to solve the puzzle without having to solve a linear system. The technique is known as *light-chasing*. Begin with the top row and press the button beneath any lit button in the top row. This will turn out all lights in the top row. Apply this strategy row by row until you reach the bottom row.

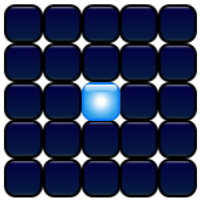
The lights in the bottom row will be one of the 7 configurations shown in Table 2, press the corresponding buttons in the top row as indicated in the table. Then apply the light-chasing strategy again, beginning from the top row. This will solve the puzzle.

Lights on bottom row$n$	Press these on top row
	
	
	
	
	
	
	

Table 2: Light-chasing strategy

24.6 Exercises

1. Solve each of the following configurations.
   (You can use the Lights-out puzzle on Jaap's puzzle page to edit the lights, then try out your solution.)


(a)


(b)


(c)


(d)

2. Show the following configuration is not solvable.