Generalidades, modelos y aplicaciones de los rompecabezas luminosos

Estudiante
Esperanza Margarita Palacios Vargas
Cod. 614161003

Supervisor
M.Sc. Rafael Melo Jiménez

Fundación Universitaria Konrad Lorenz
Facultad de Matemáticas e Ingenierías
2020 – I

Reconocimientos

Dedicado a mi tutor, por guiarme,
y a mi mamá, por apoyarme.

Resumen

Este trabajo de grado estudiará un tipo de rompecabezas a los que nos referiremos como ***rompecabezas luminosos***. De forma general, estos rompecabezas consisten en colecciones de botones con luz. Cuando los botones son presionados, los estados de las luces alternan según reglas definidas para cada rompecabezas en particular. Las luces no necesariamente están restringidas a la dualidad de encendido y apagado, algunos rompecabezas alternan entre más de dos estados.

Los rompecabezas luminosos comenzaron su existencia como juguetes electrónicos que fueron abstraídos matemáticamente, convirtiéndose en objeto de estudio científico. Naturalmente, el más famoso entre ellos se llama *Lights Out*, aunque aquí lo llamaremos *Lights Out clásico*. Se iniciará este documento comentando la historia de los rompecabezas luminosos.

Se dará fundamento teórico a los rompecabezas luminosos y, para hacerlo, se usará uno de sus subconjuntos como apoyo. Este subconjunto generaliza el Lights Out clásico, por lo que a sus miembros los llamaremos los *Lights Out*. Así, se describirá la definición del juego, se introducirán sus conceptos propios y se enunciarán los conceptos necesarios para representar matemáticamente a los Lights Out. Posteriormente, se construirá un modelo que los represente mediante ecuaciones matriciales modulares, al que llamaremos el *modelo algebraico*, y se discutirá en qué condiciones pueden ser resueltos.

Se ampliará el universo de los rompecabezas luminosos por medio de la introducción de las *variantes de Lights Out*, concebidas como rompecabezas con reglas que difieren de las de los ya modelados. Una vez introducidas las variantes, se comentarán los cambios que se deben hacer sobre el modelo introducido para que permita representar algunos de sus tipos.

Se introducirán tres aplicaciones del modelo construido: dos reproducciones de resultados computacionales conocidos acerca de un componente del modelo que llamaremos *matriz de adyacencia*, una introducción de cómo se modifica el modelo en contextos específicos que restringen la mecánica de juego, a los que llamaremos *problemas de restricción*, y la descripción de un algoritmo llamado *caza de luces* que permite la resolución de los Lights Out sin usar el modelo algebraico, cosa que lo hace sencillo de ejecutar por humanos.

Se expondrán las limitaciones del modelo algebraico para el análisis de los rompecabezas luminosos y se introducirán los beneficios de complementarlo con otras herramientas matemáticas, resaltando el uso de la teoría de grafos. Para finalizar, se presentarán conclusiones sobre el trabajo realizado y se considerarán posibles continuaciones a este.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Comentarios de apertura

Este trabajo de grado estudiará un tipo de rompecabezas[1] a los que nos referiremos como ***rompecabezas luminosos***. De forma general, estos rompecabezas consisten en colecciones de botones con luz. Cuando los botones son presionados, los estados de las luces alternan según reglas definidas para cada rompecabezas en particular.

Notamos que las luces no necesariamente están restringidas a la dualidad de encendido y apagado, por lo que algunos rompecabezas alternan entre más de dos estados. Antes de comenzar a ahondar en detalles, es conveniente primero hacer claridad sobre ciertos puntos:

1. De manera similar a como se usa el símbolo de cuadrado ($\square$) para notar el final de una demostración matemática, usaremos el símbolo de triángulo ($\triangle$) para notar el final de una definición y el símbolo de diamante ($\lozenge$) para notar el final de un ejemplo. Con esto buscamos mejorar la lectura del texto.

2. Para este proyecto se hizo necesario escribir algo de software y, además, algunas piezas de código, recopilamos todo ese material en un repositorio de GitHub accesible a través del vínculo:

   `https://github.com/mepalaciosv/Trabajo-de-grado-Rompecabezas-luminosos`

3. La investigación sobre este tema ha acarreado confusión, es decir, históricamente no ha habido uniformidad para dar nombre a sus teoremas, definiciones, o incluso al nombre mismo del problema matemático que ataca [23]. Por esta razón, si un concepto es referido por varios nombres, incluiremos uno como principal y algunos de los otros entre paréntesis. Además de esto, ocasionalmente bautizaremos conceptos a los que no se les ha dado nombre.

1.2. Definición del proyecto

Después de estas aclaraciones, procederemos a definir los límites de este trabajo de grado, comenzando por enunciar el problema que propone resolver:

[1] Ocasionalmente los llamaremos *juegos*.

1.2.1. Problema a resolver

Tomando como base el artículo de Anderson y Feil de 1998 (Ref. [11]):

1. Exponer la construcción del modelo fundamentado en álgebra lineal que este emplea para describir los rompecabezas luminosos.
2. Demostrar con claridad los teoremas que expone.
3. Reproducir sus resultados computacionales.

Hecho esto, enunciaremos el objetivo general y los específicos:

1.2.2. Objetivos

Objetivo general:

Describir matemáticamente el comportamiento de los rompecabezas luminosos y desarrollar un algoritmo computacional que permita, como en el artículo base, determinar la *nulidad*[2] de la *matriz de adyacencia*, para los *Lights Out cuadrados* y sus *variantes toroidales* de tamaño 2×2 a 21×21.

Objetivos específicos:

I. Introducir la historia de los rompecabezas luminosos.

II. Exponer la construcción de un modelo en ecuaciones matriciales modulares que describa los rompecabezas luminosos.

III. Demostrar los teoremas de *resolubilidad* expuestos por el artículo base.

IV. Introducir las *variantes* del juego, detallando la llamada *variante toroidal*.

V. Reproducir los resultados computacionales expuestos por el artículo base a través de un algoritmo extensivo a rompecabezas de tamaño arbitrario.

VI. Introducir los problemas asociados a estos rompecabezas a los que llamaremos *problemas de restricción*.

VII. Introducir el método de solución llamado *caza de luces*.

VIII. Introducir el uso de otras herramientas y modelos matemáticos para analizar y describir los rompecabezas luminosos.

1.2.3. Esquema de cumplimiento

Precisaremos los métodos a través de los que este trabajo cumple cada uno de sus objetivos específicos:

1. En la sección 1.4 se comentará la historia de los rompecabezas luminosos como juguetes electrónicos y como tema de investigación en matemáticas, garantizando el cumplimiento del objetivo específico I.

[2] Todos los términos en cursiva serán introducidos ordenadamente en este documento.

2. En la sección 2.3 se construirá una versión del modelo presentado por el artículo base generalizada a tableros de dimensión arbitraria, completando los vacíos presentes tanto en este como en otras referencias. Además, en la sección 3.2 se expondrá cómo extenderla a otros tipos de rompecabezas luminosos, garantizando el cumplimiento del objetivo específico II.
3. En la sección 2.4 se usarán conceptos de álgebra lineal y el modelo construido en la sección 2.3 para demostrar versiones de los dos teoremas expuestos por el artículo base generalizadas a tableros de dimensión arbitraria, garantizando el cumplimiento del objetivo específico III.
4. En la sección 3.1 se ampliará el universo de rompecabezas luminosos expuestos en la sección 2.3 por medio de la introducción del concepto de *variante de Lights Out* y se hará énfasis en la variante conocida como *toroidal*, garantizando el cumplimiento del objetivo específico IV.
5. En la sección 4.1.1 se expondrá el funcionamiento de un algortimo computacional construido en lenguaje Python para hallar la nulidad de la *matriz de adyacencia* tanto de los Lights Out como de sus *variantes toroidales* de tamaño arbitrario. Además, se reproducirán los resultados obtenidos por el artículo de base, garantizando el cumplimiento del objetivo específico V.
6. En la sección 4.2 se introducirá la existencia de problemas que imponen restricciones adicionales a los rompecabezas luminosos y se mostrará cómo modifican el modelo construido en la sección 2.3, garantizando el cumplimiento del objetivo específico VI.
7. En la sección 4.3 se introducirá la existencia de un método de solución iterativo para el *Lights Out clásico* que solo requiere aplicación de pasos intermedios según algunos casos, garantizando el cumplimiento del objetivo específico VII.
8. En la sección 5 se expondrán las limitaciones del modelo algebraico para el análisis de los rompecabezas luminosos y se introducirán los beneficios de complementarlo con otras herramientas matemáticas, garantizando el cumplimiento del objetivo específico VIII.

Finalizaremos la sección haciendo salvedades sobre el cumplimiento del objetivo general y el problema a resolver por el proyecto:

1. El cumplimiento de los objetivos específicos II y V garantiza el del objetivo general.
2. El cumplimiento del objetivo específico III, junto a los dos anteriormente nombrados, garantiza la resolución del problema a resolver.

1.2.4. Aporte de este proyecto

El algoritmo desarrollado para este trabajo de grado permite hallar la *nulidad* de la *matriz de adyacencia* para los *Lights Out* y sus *variantes toroidales* de *tamaño* arbitrario, eso significa que no solo puede calcular este dato para *tableros cuadrados*, sino también para *tableros rectangulares*.

Con un algoritmo así, puede tratarse el problema de hallar la *nulidad* asociada a todos los rompecabezas con tableros más *pequeños* que uno *cuadrado* de tamaño dado[3], al que se toma como límite. Entre las referencias consultadas, la Ref. [29] es la que avanza más lejos en esta empresa, llegando hasta 30×30. En este proyecto superamos ese límite y llegamos hasta 60×60. Esto podría constituir un aporte significativo si no hay otra referencia que haya progresado más que eso.

1.3. Prerrequisitos conceptuales

Habiendo delimitado este trabajo, introduciremos algunos conceptos matemáticos que nos ayudarán a entender el resto del documento:

Definición 1.3.1. *Una **matriz** A es un ordenamiento rectangular de números denotado por*

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}.$$

*Si A tiene m filas y n columnas diremos que A es una matriz de **tamaño** m por n ($m \times n$) y escribiremos $A_{m\times n}$ para indicarlo.*

*Si $m = n$, se dice que A es una **matriz (cuadrada) de orden** n, en ese caso escribimos simplemente A_n para indicarlo.*

*A menudo escribimos $A = [a_{i,j}]$ y nos referimos a $a_{i,j}$ como la **componente** (i, j) [1].* △

Definición 1.3.2. *Una **matriz lógica** es una matriz cuyas componentes son exclusivamente ceros o unos. Es decir, una matriz $A_{m\times n} = [a_{i,j}]$ es lógica si para todo $a_{i,j}$ se cumple que $a_{i,j} = 0$ ó $a_{i,j} = 1$ [13].* △

Definición 1.3.3. *Sean A y B conjuntos cualquiera[4]. El **producto cartesiano** de A y B es el conjunto de parejas ordenadas*

$$A \times B := \{(a, b) : a \in A \text{ y } b \in B\}.$$

*Una **relación binaria** $\mathcal{R}$ de A en B es un subconjunto de $A \times B$ ($\mathcal{R} \subseteq A \times B$) [14].* △

Las relaciones binarias entre conjuntos *finitos* se pueden representar como matrices lógicas:

Definición 1.3.4. *Si $\mathcal{R}$ es una relación binaria entre los elementos de dos conjuntos X, Y tales que $X = \{x_1, \dots, x_n\}$ y $Y = \{y_1, \dots, y_m\}$, $\mathcal{R}$ podrá ser representada por la matriz $R_{n\times m} = [r_{i,j}]$ definida como:*

$$r_{i,j} = \begin{cases} 1, \text{ si } (x_i, y_j) \in \mathcal{R} \\ 0, \text{ si } (x_i, y_j) \notin \mathcal{R}. \end{cases}$$

*A R se le conoce como **matriz de relación** [13]. Además, si $a \in X$ y $b \in Y$, por comodidad de la notación diremos que $a\mathcal{R}b$ si $(a, b) \in \mathcal{R}$ y diremos que $a\not\mathcal{R}b$ si $(a, b) \notin \mathcal{R}$ [14].* △

[3] Introduciremos este problema bajo el nombre de *problema del triángulo* en la sección 4.1.1.

[4] A y B pueden ser incluso conjuntos infinitos.

Las relaciones binarias de un conjunto consigo mismo tendrán especial importancia para este proyecto. Esa importancia radica en que los grafos son representaciones de este tipo de relaciones:

Definición 1.3.5. *Sea A cualquier conjunto*[5]*, sea $\mathcal{R}$ una relación sobre A ($\mathcal{R} \subseteq A \times A$) y sean $a, b, c \in A$ arbitrarios. Diremos que [14]:*

1. *$\mathcal{R}$ es **reflexiva (irreflexiva)** si $a\mathcal{R}a$ ($a\not\mathcal{R}a$) para cualquier a.*
2. *$\mathcal{R}$ es **simétrica** si $a\mathcal{R}b$ siempre implica que $b\mathcal{R}a$.*
3. *$\mathcal{R}$ es **transitiva** si $a\mathcal{R}b$ y $b\mathcal{R}c$ siempre implican que $a\mathcal{R}c$.*
4. *$\mathcal{R}$ es **antisimétrica** si $a\mathcal{R}b$ y $b\mathcal{R}a$ siempre implican que $a = b$.*

*Si A es un conjunto finito, diremos que la matriz de relación de $\mathcal{R}$ es una **matriz de adyacencia**[6] [19].* △

De ahora en adelante, cada vez que hablemos de relaciones binarias nos referiremos al tipo de las que acabamos de exponer. Además, vale la pena notar que las matrices de adyacencia son siempre cuadradas.

Extenderemos las propiedades de las relaciones binarias a sus matrices de adyacencia:

Definición 1.3.6. *Una matriz de adyacencia $R_{n\times n} = [r_{i,j}]$ es **reflexiva** si para todo índice i se cumple que $a_{i,i} = 1$. Si, en cambio se cumple que $a_{i,i} = 0$, la matriz será **irreflexiva**.*

*Una matriz de adyacencia $R_{n\times n} = [r_{i,j}]$ es **simétrica** si todos sus índices i, j cumplen que $a_{i,j} = a_{j,i}$. Es decir, si $R = R^T$ [13].* △

Enunciaremos algunos conceptos sobre matrices que nos serán de utilidad:

Definición 1.3.7. *El **núcleo** de una matriz A es el conjunto de todos los vectores x que satisfacen la ecuación $Ax = 0$ [1].* △

Teorema 1.3.1. *Si A es una matriz invertible, entonces la ecuación $Ax = b$ tiene exactamente una solución para cada b.*

Por otro lado, los *cuerpos* son estructuras algebraicas de comportamiento semejante al del conjunto de los reales ($\mathbb{R}$):

Definición 1.3.8. *Un **cuerpo** K es un conjunto con dos operaciones, usualmente notadas como suma ($+$) y producto ($*$), que para todos $a, b, c \in K$ cumple los siguientes axiomas [6]:*

1. *Cerradura de la suma y el producto:*

$$x + y \in K \qquad y \qquad x * y \in K.$$

2. *Asociatividad de la suma y el producto:*

$$a + (b + c) = (a + b) + c \qquad y \qquad a * (b * c) = (a * b) * c.$$

[5] A puede ser incluso un conjunto infinito.
[6] Este nombre proviene de la teoría de grafos (sección 5.2).

3. *Conmutatividad de la suma y el producto:*

$$a + b = b + a \qquad y \qquad a * b = b * a.$$

4. *Distributividad del producto sobre la suma:*

$$a * (b + c) = a * b + a * c.$$

5. *Existencia de neutros para la suma y el producto:*

$$\exists 0 \in K : 0 + a = a, \qquad \exists 1 \in K : 1 * a = a \qquad y \qquad 0 \neq 1.$$

6. *Existencia de inversos para la suma y el producto:*

$$\exists -a : a + (-a) = 0 \qquad a \neq 0 \Rightarrow \exists a^{-1} : a * (a^{-1}) = 1.$$

△

Algunos otros cuerpos son el de los racionales ($\mathbb{Q}$), el de los complejos ($\mathbb{C}$) y el *primo de orden p* ($\mathbb{Z}_p$). El cuerpo primo de orden p será especialmente importante para este proyecto:

Definición 1.3.9. *Sea p un número primo, el* ***cuerpo primo de orden*** *p* ($\mathbb{Z}_p$) *se define como* $\mathbb{Z}_p := \{m \mod p \mid m \in \mathbb{Z}\} = \{0, 1, \ldots, p-1\}$. *Este cuerpo también es notado como* $\mathbb{Z}/p\mathbb{Z}$ *[6].* △

Las propiedades de la aritmetica modular nos permiten expresar la suma y producto de $\mathbb{Z}_p$ ($+_p$ y $*_p$) en términos de las usuales de $\mathbb{Z}$:

$$a +_p b := (a + b) \mod p$$
$$a *_p b := (a * b) \mod p.$$

Aunque si no hay lugar a ambiguedad, es costumbre referirse a ellas con los símbolos usuales de suma y producto.

Esta forma de expresar las operaciones de cuerpo de $\mathbb{Z}_p$ es conveniente en la implementación de algoritmos computacionales.

Como caso particular, dado que un número solo puede ser par o impar, la aritmética nos permite determinar que $\mathbb{Z}_2 = \{0, 1\}$ y condensar su suma y producto como tablas de Cayley:

$+$	0	1
0	0	1
1	1	0

$*$	0	1
0	0	0
1	0	1

Adicionalmente, los cuerpos pueden usarse para construir *espacios vectoriales*, estos son estructuras de comportamiento algebraico semejante al de los vectores de $\mathbb{R}^n$:

Definición 1.3.10. *Un **espacio vectorial** V sobre un cuerpo*[7] *K, es un conjunto con dos operaciones, usualmente notadas como suma $(+)$ y producto escalar $(\cdot)$, que para todos $a, b \in K$ y $u, v, w \in V$ cumple los siguientes axiomas [1]:*

1. *Asociatividad de la suma:*

$$u + (v + w) = (u + v) + w.$$

2. *Conmutatividad de la suma:*

$$u + v = v + u.$$

3. *Existencia de neutros para la suma y el producto escalar:*

$$\exists 0 \in V : 0 + u = u \qquad y \qquad \exists 1 \in K : 1 \cdot u = u.$$

4. *Existencia de inversos para la suma:*

$$\exists -u : u + (-u) = 0.$$

5. *Compatibilidad del producto escalar y el de cuerpo:*

$$a(bu) = (ab)u.$$

6. *Distributividad del producto escalar sobre la suma y de la suma sobre el producto escalar:*

$$a(u + v) = a \cdot u + a \cdot v \qquad y \qquad (a + b) \cdot u = a \cdot u + b \cdot u.$$

△

Si este espacio vectorial es también un espacio de matrices, las matrices que lo conforman tendrán también el comportamiento algebraico que conocemos de $\mathbb{R}^{m \times n}$.

Definición 1.3.11. *Sea $n \in \mathbb{N}$, $n \geq 1$, y p un número primo, $\mathbb{Z}_p^n$ es el espacio vectorial definido sobre $\mathbb{Z}_p$ de todos los n-vectores cuyas componentes son elementos de $\mathbb{Z}_p$, es decir,*

$$\mathbb{Z}_p^n := \{(a_1, \ldots, a_i, \ldots, a_n)^T \mid a_i \in \mathbb{Z}_p \text{ y } 1 \leq i \leq n\} [6].$$

△

Definiremos de forma natural la suma en $\mathbb{Z}_p^n$ como la suma de $\mathbb{Z}_p$ aplicada componente a componente y definiremos el producto escalar de $\mathbb{Z}_p^n$ sobre $\mathbb{Z}_p$ aplicando el producto de $\mathbb{Z}_p$ sobre cada componente de $\mathbb{Z}_p^n$. Así:

Definición 1.3.12. *Si $a, b \in \mathbb{Z}_p^n$ con $a = (a_1, \ldots, a_n)^T$ y $b = (b_1, \ldots, b_n)^T$, entonces*

$$a + b = (a_1, \ldots, a_n)^T + (b_1, \ldots, b_n)^T = (a_1 + b_2, \ldots, a_n + b_n)^T [6].$$

△

[7] En este contexto, a los elementos de K se les llama *escalares*.

Definición 1.3.13. *Si $s \in \mathbb{Z}_p$ y $v = (v_1, \ldots, v_{mn})^T \in \mathbb{Z}_p^{mn}$, entonces*

$$s * v = s * (v_1, \ldots, v_{mn})^T = (sv_1, \ldots, sv_{mn})^T \text{ [6].}$$

△

Para finalizar, usamos los vectores para exponer algunos conceptos que nos serán útiles:

Definición 1.3.14. *Sean K un cuerpo y V un espacio vectorial sobre K [1]. Un vector $v \in V$ es una **combinación lineal** de los vectores $v_1, \ldots, v_n \in V$ si existen escalares $a_1, \ldots, a_n \in K$ tales que*

$$v = a_1 v_1 + \cdots + a_n v_n.$$

△

Definición 1.3.15. *El **espacio columna** de una matriz A ($col(A)$) es el conjunto de todas las combinaciones lineales de sus vectores columna [1].* △

Definición 1.3.16. *El **rango** de una matriz A ($ran(A)$) es la dimensión[8] de su espacio columna. La **nulidad** A ($ker(A)$) es la dimensión de su núcleo [1].* △

Teorema 1.3.2. *Un sistema lineal $Ax = y$ tiene al menos una solución si y solo si y pertenece a $col(A)$.*

Teorema 1.3.3 (Teorema del rango y la nulidad)**.** *Si $A = (a_{i,j})_{m \times n}$ es una matriz, entonces $ran(A) + nul(A) = n$.*

1.4. Contexto histórico

Ya introducidos los conceptos matemáticos que necesitamos, procederemos a comentar de forma breve la historia de los rompecabezas luminosos, resumiremos los eventos principales de esta historia en la Fig. 1.2:

Los rompecabezas luminosos comenzaron su existencia como juguetes electrónicos que fueron abstraidos matemáticamente, convirtiéndose en objeto de estudio científico.

La compañía Parker Brothers lanzó en 1978 uno de los primeros modelos de rompecabezas electrónicos de botones luminosos bajo el nombre de *MERLIN* (Fig. 1.1a). Sus luces, dispuestas en una pequeña cuadrícula, se encendían y apagaban al presionarlas según patrones definidos [19].

[8] La dimensión de un espacio vectorial es el número de vectores necesarios para construirle una base.

(a) *MERLIN* (b) *XL-25* (c) *Lights Out*

(d) *Orbix* (e) *Lights Out 2000*

Figura 1.1: Primeros rompecabezas luminosos.

La Ref. [2] es la más temprana de un problema asociado a los rompecabezas luminosos. Fue publicada en 1979 y en ella László Lovász da crédito a Tibor Gallai[9] por resolver el *problema del tablero totalmente encendido* desde una perspectiva de teoría de grafos [23].

La compañía Vulcan Electronics produjo un juego similar a *MERLIN* llamado *XL-25* en 1983 (Fig. 1.1b). No obstante, el juego más popular de esta clase fue creado por Tiger Electronics en 1995 usando el nombre de *Lights Out* (Fig. 1.1c) [24].

Estos juegos despertaron más interés en círculos de investigación. Don Pelletier publicó en 1987 uno de los primeros artículos sobre ellos, modelando a *MERLIN* desde una perspectiva de álgebra lineal [3].

Klaus Sutner publicó dos artículos sobre temas afines a estos juegos, ambos desde una perspectiva de autómatas celulares bidimensionales. El primero, de 1989, trataba un problema asociado llamado el *problema del jardín del edén*. El segundo, de 1990[10], estudiaba el vínculo entre los rompecabezas y los autómatas [4][5].

Lights Out generó una popularidad creciente para estos juegos, pues a lo largo de la segunda mitad de los años 90 se lanzaron otros de este tipo que incluían rejillas de varios tamaños y variaciones en la jugabilidad. Nombres como *Orbix* (Fig. 1.1d), de Milton Bradley lanzado en 1995, y *Lights Out 2000* (Fig. 1.1e), de Tiger Electronics, son algunos de los más importantes [19]. *Orbix* tiene especial importancia en cuanto sus botones están distribuidos en una esfera y no como cuadrícula.

[9] Su supervisor doctoral.

[10] Aunque Sutner afirma que fue escrito en 1986.

1978	Parker Brothers lanza el juego *MERLIN*.
1979	Primera referencia bibliográfica de un problema referente a rompecabezas luminosos.
1983	Vulcan Electronics lanza el juego *XL-25*.
1987	Pelletier publica su artículo sobre el juego *MERLIN*.
1989	Sutner publica su artículo sobre el jardín del eden.
1990	Sutner publica su artículo sobre el σ-juego.
1995	Tiger Electronics lanza el juego *Lights Out*.
1995	Milton Bradley lanza el juego *Orbix*.
1998	Anderson y Feil publican su artículo sobre el modelo de álgebra lineal.

Figura 1.2: Hitos en la historia de los rompecabezas luminosos.

Goldwasser, Klostermeyer, Trapp y Zhang publicaron en 1995 uno de los primeros artículos que discutían el método llamado *caza de luces*, un método iterativo de resolución para rompecabezas luminosos cuya aplicación requiere seguir casos pero no hacer cálculos [7].

Anderson y Feil publicaron un artículo en 1998 que describe *Lights Out* por medio del álgebra lineal, lo generaliza a cuadrículas cuadradas de tamaño arbitrario e introduce una aplicación computacional calculando la nulidad de una parte del modelo llamada la *matríz de adyacencia* [11].

La fama de este tipo de juegos conllevó a que no solo pudieran encontrarse como dispositivos electrónicos, sino también como juegos en línea e incluso como aplicaciones para teléfonos inteligentes [24]. Esa fama también logró generar gran cantidad de rompecabezas con mecánicas y disposiciones de tablero novedosas.

La labor de estos primeros investigadores motivó mucha más investigación sobre el tópico de los rompecabezas luminosos, extendiendo así el la aparente simplicidad del juego a ramas tan diversas como la teoría de grafos, las secuencias polinómicas, los autómatas y otros [19].

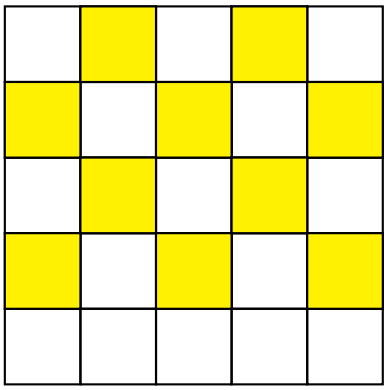
Capítulo 2

Teoría

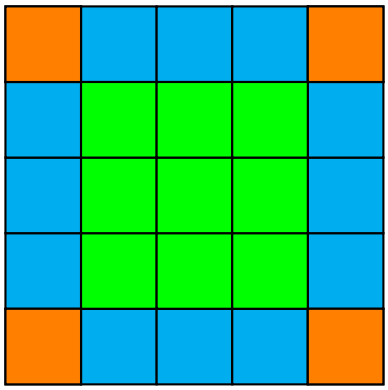
2.1. Mecánica del juego

Hemos comentado bastante sobre rompecabezas luminosos, no obstante, aún no conocemos sus mecánicas. En esta sección mostraremos como se juega *Lights Out*, el juego más popular de esta clase:

Lights Out se juega en una cuadrícula cuadrada de 25 *botones* luminosos organizados en 5 filas y 5 columnas a la que llamaremos el *tablero*, podemos verla representada en la Fig. 2.1a.



(a) Representación del tablero de *Lights Out*



(b) Zonas del tablero.

Figura 2.1: Representación y zonas del tablero.

Cada luz puede estar en uno de dos *estados*: encendido o apagado, a los que representaremos respectivamente con los colores amarillo y blanco. Al comenzar el juego se nos presenta algún *patrón inicial* de luces encendidas y apagadas, como podemos ver en la Fig. 2.1a.

Como se aprecia en la Fig. 2.1b, el tablero tiene tres zonas: el centro (en verde), los bordes (en azul) y las esquinas (en naranja). Las desglosaremos a continuación:

La Fig. 2.2a nos muestra que un botón del centro (en azul) tiene ocho vecinos (en verde). De estos vecinos, como muestra la Fig. 2.2b, cuatro serán ortogonales (en naranja) mientras

que los otros cuatro serán diagonales (en verde).

(a) Total de vecinos de un botón del centro. (b) Vecinos ortogonales de un botón del centro.

Figura 2.2: Vecinos de un botón del centro.

La Fig. 2.3a nos muestra que un botón del borde[1] (en azul) tiene cinco vecinos (en verde), el límite del tablero elimina tres posibles. De estos vecinos, como muestra la Fig. 2.3b, tres serán ortogonales (en naranja) mientras que los otros dos serán diagonales (en verde).

(a) Total de vecinos de un botón de borde. (b) Vecinos ortogonales de un botón de borde.

Figura 2.3: Vecinos de un botón de borde.

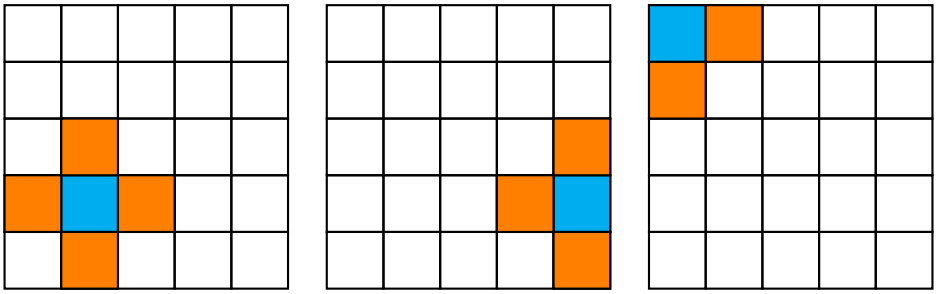
Finalmente, la Fig. 2.4a nos muestra que un botón de la esquina[2] (en azul) tiene tres vecinos (en verde), los límites del tablero eliminan cinco posibles. De estos vecinos, como muestra la Fig. 2.4b, dos serán ortogonales (en naranja) mientras que el restante será diagonal (en verde).

(a) Total de vecinos de un botón de esquina. (b) Vecinos ortogonales de un botón de esquina.

Figura 2.4: Vecinos de un botón de esquina.

En la Fig. 2.5 mostramos ejemplos, para cada zona del tablero, de cuáles serían los vecinos ortogonales (en naranja) de un botón dado (en azul).

[1] En este caso del borde superior.
[2] En este caso de la esquina inferior derecha.

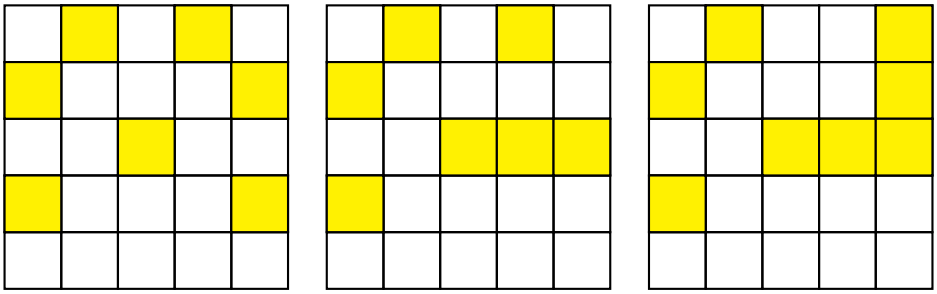


(a) Centro. (b) Borde. (c) Esquina.

Figura 2.5: Ejemplos de los vecinos ortogonales según la zona del tablero.

Presionar un botón constituye un *movimiento*. Hacerlo alternará su *estado* y el de sus vecinos ortogonales, es decir, un botón encendido se apagará y uno apagado se encenderá. Esto modificará el *patrón actual* del tablero.

En la Fig. 2.6 mostramos qué pasaría si se toma el patrón de la Fig. 2.1a y, midiendo de arriba a abajo y de izquierda a derecha: primero se presiona el botón de la 3^{era} fila con 3^{era} columna (Fig. 2.6a), luego se presiona el botón de la 3^{era} fila con 5^{ta} columna (Fig. 2.6b) y, finalmente, se presiona el botón de la 1^{era} fila con 5^{ta} columna (Fig. 2.6c).



(a) 3^{era} fila 3^{era} columna. (b) 3^{era} fila 5^{ta} columna. (c) 1^{ra} fila 5^{ta} columna.

Figura 2.6: Ejemplos de movimientos.

Resulta comparativamente más sencillo pensar que el efecto de presionar un botón (en azul) es alternar los botones contenidos en la intersección del tablero con la forma de una cruz (+), como la de la Fig. 2.7, que lo toma como centro. A esta cruz la llamaremos el *área de efecto* del movimiento.

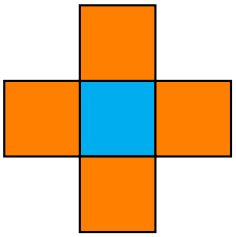


Figura 2.7: Área de efecto de un movimiento.

El *objetivo* del juego consiste en apagar todas las luces, es decir, llegar a un patrón totalmente blanco. A ese patrón objetivo se conoce como *estado final*. El *objetivo secundario*, y por tanto opcional, del juego es llegar al estado final en la menor cantidad de movimientos posible.

Para que el lector se familiarice con la mecánica del juego, hemos desarrollado una aplicación web que ubicamos en la carpeta **Lights Out** del repositorio GitHub. Para usarla, se debe hacer clic en el archivo HTML de nombre **index** contenido en ella. Al hacerlo, la aplicación se abrirá en el navegador en caso de que este soporte Javascript[3]. Acto seguido, aparecerá un cuadro de diálogo con la pregunta

"¿De qué tamaño quieres el tablero?"

al que se le deberá ingresar un cinco (5). Cuando se haga, la aplicación simulará un Lights Out con todas sus luces encendidas.

2.2. Definición del juego

Este juego que hemos aprendido a jugar, Lights Out, puede ser generalizado con las mismas reglas a tableros de dimensión arbitraria:

Definición 2.2.1. *Como un símil al lenguaje de las matrices; si este juego se lleva a un tablero con m filas y n columnas, al juego resultante lo llamaremos un* ***Lights Out de tamaño***[4] *$m \times n$. Al conjunto de todos los posibles lo llamaremos los* ***Lights Out****.*

Si $m = n$, al juego resultante lo llamaremos un ***Lights Out (cuadrado) de orden*** *n. Si un Lights Out no es cuadrado lo llamaremos* ***rectangular*** *[11].* △

Al rompecabezas que discutimos en la sección 2.1 lo llamaremos de ahora en adelante el *Lights Out clásico* o el *Lights Out original*.

Ya tenemos el lenguaje para revelar la verdad sobre la aplicación que desarrollamos: esta aplicación es un simulador de Lights Out cuadrados que inician con todos sus botones encendidos y el parámetro que pide al comenzar es el orden n del tablero. Si el lector ingresa un $n \in \mathbb{Z}^+$, $n \neq 5$, podrá experimentar con Lights Out de distintos tamaños[5].

Antes de continuar, es menester dedicarle unas palabras a la confusión de la que habíamos hablado en la sección 1.1:

- Muchos artículos se refieren a este problema matemático como *Lights Out*, incluyendo la Ref. [11].
- Otros nombres usados para referirse a él son *problema de ajuste de interruptores* [10][9], y *problema de las nueve colas* (para un tablero de 3×3 como el de *MERLIN*) [23].

[3] Prácticamente todos los navegadores web lo hacen actualmente.
[4] O símplemente un *Lights Out de $m \times n$*.
[5] Recomendamos que ese n no sea muy grande para que el tamaño de los botones no afecte la jugabilidad.

- Adicionalmente, algunos autores lo trataron desde la perspectiva de los autómatas celulares, y se refirieron a él como σ-juego, σ-autómata o σ^+-autómata[6] [4][5][8].

Hemos tomado la decisión de llamar *rompecabezas luminosos* al conjunto de todos los rompezabezas de botones con luz y llamar Lights Out solo a estos que con las mismas reglas generalizan el Lights Out original. Antes de continuar, resumiremos la mecánica de juego de los Lights Out:

- *Los Lights Out se juegan en cuadrículas de botones luminosos de dimensión arbitraria.*
- *Cada botón puede estar en uno de dos estados posibles: encendido y apagado.*
- *Al comenzar el juego se nos presenta algún patrón inicial de luces prendidas y apagadas.*
- *Presionar un botón alterna la intersección del tablero con una cruz que lo toma como centro.*
- *El objetivo del juego consiste en apagar todas las luces.*
- *El objetivo secundario, y por tanto opcional, del juego es llegar al estado final en la menor cantidad de movimientos posible.*

Hasta este punto, hemos introducido algunos conceptos sin precisarlos demasiado. Por este motivo, apelaremos a las intuiciones que hemos construido para poder hablar de ellos de forma inequívoca *en el contexto de los Lights Out*. Nos apoyaremos en la Fig. 2.8 para construir ejemplos:

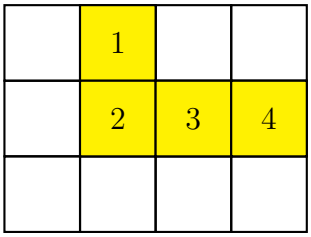


Figura 2.8: Ejemplo de Lights Out de tamaño 3×4.

Noción 2.2.1. *Podemos entender los* ***botones (luces)*** *como la idea de los botones físicos de los rompecabezas electrónicos o de las casillas que tienen sus contrapartes digitales. Por ejemplo, cada celda de la Fig. 2.8 es un botón.* △

Noción 2.2.2. *Los* ***tableros*** *son las cuadrículas en las que los botones se distribuyen. Por ejemplo, la Fig. 2.8 muestra un tablero de tamaño* 3×4. △

Los tableros no deben ser necesariamente cuadrículas, en el capítulo 3 introduciremos otros rompecabezas luminosos con tableros que **no** consisten en cuadrículas.

Noción 2.2.3. *Podemos entender los* ***estados*** *de los botones como la idea de que una luz puede estar encendida ó apagada. Por ejemplo, en la Fig. 2.8 hay tanto luces encendidas como apagadas.* △

[6]Esta terminología fue introducida por Sutner, el + es por el área de efecto del juego.

Las luces no necesariamente deben están restringidas a la dualidad de encendido y apagado, en el capítulo 3 introduciremos otros rompecabezas con luces que alternan entre más de dos estados.

Noción 2.2.4. *Podemos entender los* ***patrones (configuraciones) de luces*** *como la forma en que en un tablero se distribuyen los estados de los botones, es decir, toda la información de qué botones están en cuál estado en un tablero dado. Por ejemplo, en la Fig. 2.8, los botones marcados como 1, 2, 3 y 4 están encendidos mientras que los demás están apagados.* △

Noción 2.2.5. *Podemos entender los* ***movimientos (acciones, jugadas)*** *como el acto de presionar uno (y solo uno) de los botones. Hacer esto alternará la intersección del tablero con una cruz centrada en el botón presionado.* △

Los movimientos no necesariamente deben estar restringidos al área de efecto que introdujimos en la sección 2.1. En la sección 3 introduciremos otros rompecabezas luminosos donde los botones tienen efectos diferentes.

Noción 2.2.6. *Al comenzar el juego, el tablero se encontrará en algún patrón dado que llamaremos* ***estado inicial (patrón inicial, configuración inicial)****. Cada vez que ejecutamos una jugada, el patrón de luces cambiará a un nuevo* ***estado (patrón) actual****.* △

Noción 2.2.7. *El objetivo del juego es apagar todas las luces. A este patrón con todas las luces apagadas se le conoce como* ***estado (patrón) final****. Cuando esto pasa, decimos que hemos alcanzado una* ***solución*** *de la partida.* △

El objetivo no necesariamente debe ser apagar todas las luces. En la sección 3 introduciremos otros rompecabezas luminosos con objetivos diferentes.

Hemos introducido las nociones que necesitabamos, con ellas podemos pasar a definir unos conceptos que nos serán de ayuda:

Definición 2.2.2. *Una* ***solución óptima*** *es una solución que alcanza el objetivo del juego en la cantidad mínima posible de movimientos [19].* △

Definición 2.2.3. *El* ***espacio de configuración*** C_G *de un tablero* G *es el conjunto de todos sus posibles estados iniciales*[7] *[19].* △

Definición 2.2.4. *El* ***espacio de jugabilidad*** C_{C_G} *de un tablero* G *es el conjunto de todas las configuraciones posibles que se pueden alcanzar a través de ejecutar movimientos sobre un patrón inicial dado [19].* △

Para un tablero G, C_G y C_{C_G} serán iguales si cualquier patrón es alcanzable por medio de movimientos a partir de cualquier otro. No obstante, como veremos en la sección 2.4, con frecuencia este no será el caso [19].

Como sabemos que un Lights Out cualquiera tiene m filas y n columnas, el siguiente teorema es consecuencia de una pequeña cuenta:

[7] Usar letras mayúsculas para notar los tableros es una costumbre usual cuando se estudian los Lights Out a través de la teoría de grafos [19].

Teorema 2.2.1. *El número de casillas de un Lights Out cualquiera es mn.*

El estado de un botón no restringe el estado de los demás, esto ocurre porque los estados iniciales se comportan como variaciones con repetición. Si tenemos esto en cuenta, el siguiente teorema surge naturalmente como ejercicio de combinatoria y, además, nos será útil en la sección 2.4:

Teorema 2.2.2. *Si G es un Lights Out con mn botones, el espacio de configuración C_G tiene un tamaño (cardinal) de 2^{mn}.*

Después de haber definido los Lights Out, procederemos a construir un modelo matemático que los describa de forma precisa en términos de *ecuaciones matriciales modulares.*

2.3. Construcción del modelo

Nuestro objetivo principal en esta sección es encontrar una *solución universal* para los Lights Out, es decir, un algoritmo general que nos indique cuáles botones presionar dado un cierto patrón inicial.

Según Khoury, las preguntas naturales que emergen respecto a esta solución son [17]:

1. ¿Existe solución para cualquier patrón inicial? Si la hay, ¿por qué? Si no existe, ¿cuál es el conjunto de patrones iniciales que tienen solución?
2. Si suponemos que existe una solución para un patrón inicial dado, ¿cómo podemos derivarla del patrón?

Como medio para encontrar esta solución universal se desarrollará un modelo consistente en una ecuación que describa cómo presionar los botones afectará a las luces. Este modelo no es de autoría propia, es producto de completar los vacíos argumentales de las Refs. [3], [11], [17] y [21]. Usaremos el Lights Out original como instrumento didáctico para acompañar el desarrollo del modelo con ejemplos.

Anderson y Feil nos aportan un par de observaciones sobre el comportamiento de los Lights Out que nos serán útiles más adelante [11]:

1. Un botón alternado dos veces vuelve al estado en que inició, luego, presionar un botón un número de veces par es equivalente a no haberlo presionado en absoluto. Así, como las soluciones óptimas no repiten movimientos, para alcanzar una cada botón debe presionarse máximo una vez.
2. Una luz es alternada siempre no importando cuál de los botones que pueden afectar su estado es accionado. De esta forma, su estado actual solo depende del estado en el que empezó y de la paridad del número de veces que ella y sus vecinos con poder para alternarla se han presionado. Por tanto, el orden en los botones se presionen en una partida es irrelevante.

Procederemos a construir el modelo ahora que contamos con suficientes herramientas:

Lo primero que deberíamos hacer es rotular los botones de alguna manera. Como los tableros de Lights Out son cuadrículas, estaríamos tentados a usar dos índices para representar una luz dada, como en las matrices[8]. No obstante, preferiremos que cada casilla tenga un solo índice, la razón la comentaremos más adelante [17].

Hay muchas formas de hacerlo, pero una de ellas es indexar los botones de izquierda a derecha y de arriba hacia abajo. Estos índices comenzarán, solo por convención, desde el 1. Usaremos la letra i para denotarlos y, como el tamaño del tablero es mn (teorema 2.2.1), $i = 1, \ldots, mn$ [17].

Ejemplo 2.3.1. *En la Fig. 2.9 mostramos como sería la indexación para el Lights Out original, donde* $mn = 25$.

Constatamos que el índice del botón de la 1^{era} *fila con* 1^{era} *columna es* 1 *y que el índice del botón de la* 2^{da} *fila con* 4^{ta} *columna es* 9. ◊

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figura 2.9: Enumeración de los botones del Lights Out clásico.

Con los botones indexados, podemos representar sus estados. Como solo hay dos estados posibles para los botones, una idea natural es representarlos como ceros y unos, es decir, como elementos del cuerpo $\mathbb{Z}_2$ donde, solo por convención [11][17]:

$$1 \rightarrow \text{encendido} \qquad \text{y} \qquad 0 \rightarrow \text{apagado.}$$

Ejemplo 2.3.2. *Para el Lights Out clásico, la Fig. 2.10 muestra una configuración de luces cualquiera: en ella, el estado del botón* $i = 1$ *es* 1, *mientras que el estado del botón* $i = 9$ *es* 0. ◊

Ahora, introduciremos una notación para los estados del tablero: los notaremos con letras minúsculas donde, por convención arbitraria [3]:

$$y \rightarrow \text{estado inicial,} \qquad p \rightarrow \text{estado actual} \qquad \text{y} \qquad f \rightarrow \text{estado final.}$$

[8]Más aún porque los tableros se introdujeron usando su lenguaje.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figura 2.10: Ejemplo de un patrón de luces del Lights Out clasico.

Usaremos letras con subíndices para referirnos al estado de cada botón de una configuración, donde la letra será la misma que la de la configuración. De forma general, dada una configuración c, diremos por notación que $c_k = z$ si en c, el botón de índice k está en el estado z. Así, para todo índice $i = 1, \ldots, mn$ y cada configuración c, $c_i \in \mathbb{Z}_2$ por construcción [3][17].

Ejemplo 2.3.3. *Si la Fig. 2.10 representa una configuración c, podemos decir que $c_1 = 1$ y que $c_9 = 0$.* ◊

Representaremos las configuraciones de una forma bastante directa: como vectores conformados por los estados de los botones. Así, para cualquier configuración c:

$$c = (c_1, \ldots, c_i, \ldots, c_{mn})^T.$$

Estos son vectores columna por convención y los representamos con la transpuesta para ahorrar espacio. Podemos ver que cada patrón de luces está representado por un mn-vector de ceros y unos, es decir, por un elemento del conjunto $\mathbb{Z}_2^{mn}$ [17].

Ejemplo 2.3.4. *La configuración c de la Fig. 2.10 se puede representar*[9] *como*

$$c = (1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1)^T.$$

◊

Entendemos ahora por qué preferíamos que cada casilla tuviera un solo índice: las configuraciones se representan como vectores y carece de sentido que las componentes de una estructura de una sola dimensión estén determinadas por más de un índice[10].

La forma de representar los estados de tablero que hemos construido nos servirá también para representar los efectos de presionar los botones sobre el patrón actual, es decir, las acciones. Lo haremos de una forma que no nos exige escapar de nuestro conjunto de trabajo ($\mathbb{Z}_2^{mn}$).

Recordemos que para un tablero dado, existen mn acciones disponibles: una por cada botón que lo conforma. Notaremos las acciones por medio de la letra a con subíndices correspondientes con los de los botones. De esta forma, a_i será el efecto sobre el patrón actual de presionar el i-ésimo botón. Naturalmente, $i = 1, \ldots, mn$ [17].

[9] Es notorio el ahorro de espacio que logramos al usar la transpuesta.

[10] Como curiosidad, esta es la forma en la que Fortran entiende las matrices. Esto hace a las operaciones computacionalmente más rápidas a costa de hacer más complicado su manejo abstracto.

Ejemplo 2.3.5. *Para el Lights Out clásico, a_2 es la acción del botón con índice 2 mientras que a_9 es la acción del botón con índice 9.* ◊

Las acciones serán mn-vectores de ceros y unos, como las configuraciones, donde cada posición $j = 1, \ldots, mn$ del vector indica si esa acción alterna o no al botón con índice j. Cada botón tiene, por la mecánica del juego, un grupo de luces a las que alternará y otro grupo a las que no. Notaremos por convención que [3][17]:

$$1 \ \rightarrow \ \text{si alterna} \qquad \text{y} \qquad 0 \ \rightarrow \ \text{no alterna.}$$

Sea $a_i \in \mathbb{Z}_2^{mn}$ cualquiera, con $i = 1, \ldots, mn$, la acción de al presionar el i-ésimo botón, entonces a_i se representa como

$$a_i = (a_{i,1}, \ldots, a_{i,j}, \ldots, a_{i,mn})^T,$$

donde los $a_{i,j} \in \mathbb{Z}_2$, $j = 1, \ldots, mn$ representan el efecto de la acción i sobre el botón j [17]. Es nuevamente comprensible por qué decidimos usar un solo índice al rotular las luces: los efectos de una acción sobre los botones acarrean dos subíndices, hacerlos acarrear tres haría este proceso innecesariamente engorroso.

Ejemplo 2.3.6. *Siguiendo con el Lights Out clásico,*

$$a_2 = (1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T \ y$$
$$a_{19} = (0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0)^T.$$

◊

Teniendo representaciones para las configuraciones y las acciones podemos ya representar cómo las acciones van modificando el estado actual, es decir, cómo evoluciona el tablero a medida que jugamos el juego. Si partimos del estado inicial y y ejecutamos una sucesión de acciones $a_{k(1)}, \ldots, a_{k(j)}, \ldots, a_{k(l)}$, donde los $k(j)$ son índices $i = 1, \ldots, mn$ y $l \in \mathbb{Z}^+$ cumple $l \geq 1$, llegaremos a un nuevo estado actual p [17].

Ejemplo 2.3.7. *Si partimos del patrón mostrado en la Fig. 2.10 como patrón inicial (y) y ejecutamos las acciones a_1, a_7, a_{13}, a_{19} y a_{25}, llegaremos a una configuración actual (p) que coincide con el estado final del juego (f).* ◊

Describiremos la ejecución sucesiva de acciones sobre una configuración usando un operador suma ($+$) que operará sobre el conjunto donde están definidas ($\mathbb{Z}_2^{mn}$) [17]. Así, la evolución de cómo se llega a un p dado si notamos de izquierda a derecha la ejecución de las acciones $a_{k(j)}$ antes listadas será representada como

$$a_{k(l)} + (\cdots + (a_{k(1)} + y)) = p.$$

Ejemplo 2.3.8. *Podemos representar la ejecución de las acciones del ejemplo anterior como*

$$a_{25} + (a_{19} + (a_{13} + (a_7 + (a_1 + y)))) = p = f.$$

◊

Hemos descrito la ejecución sucesiva de acciones, pero no hemos mostrado aún una forma para determinar un estado actual a partir de uno inicial y unas acciones que se le apliquen. No obstante, podemos hacerlo fácilmente usando la suma de $\mathbb{Z}_2^{mn}$ [17].

Este operador es conmutativo y asociativo, por lo que podemos suprimir el uso de paréntesis sin ambiguedad y alterar libremente el orden de operaciones sin cambios en el resultado. Además, nos permite prescindir naturalmente de esa definición artificiosa de acciones sucesivas a izquierda. Como vemos, esta bien hubiera podido ser a derecha, alternando lados o en cualquier orden [17]. Solo para reflejar esto diremos que

$$y + a_{k(1)} + \cdots + a_{k(l)} = p. \tag{2.1}$$

Ejemplo 2.3.9. *Respetando el orden, si calculamos el resultado del ejemplo anterior obtendremos*

$$\begin{aligned}
&(1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T+\\
&(0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0)^T+\\
&(0,0,0,0,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0,0)^T+\\
&(0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0)^T+\\
&(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1)^T+\\
&(1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1)^T=\\
&(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T.
\end{aligned}$$

Podemos ver que para el ejemplo que acompañamos, $f = (0,\ldots,0)^T \in \mathbb{Z}_2^{mn}$. ◊

Este operador de suma refleja bastante bien nuestras intuiciones iniciales sobre el rompecabezas. Además, nos permite describir de forma precisa el mecanismo alternante de los botones cuando (en cualquier orden) un sumando representa el estado de una luz y el otro representa si bien o no esa luz fue alternada [17]. De forma más detallada:

- No alternar un botón apagado resulta en un botón apagado,
- así como alternar un botón encendido;
- No alternar un botón encendido resulta en un botón encendido,
- así como alternar un botón apagado.

Ya estamos en capacidad de hacer nuestro primer intento de escribir una ecuación que resuelva el rompecabezas Lights Out. Sea $y \in \mathbb{Z}_2^{mn}$ el patrón inicial del rompecabezas, nuestro objetivo es encontrar una sucesión de índices finita n con elementos $n(1), n(2), \ldots, n(N)$, $N \in \mathbb{N}$, tal que

$$a_{n(N)} + a_{n(N-1)} + \cdots + a_{n(1)} + y = f.$$

Definición 2.3.1. *A una sucesión de acciones* $a_{n(1)}, \ldots, a_{n(N)}$ *que soluciona el rompecabezas se le conoce como una* ***estrategia ganadora****. A las sucesiones de acciones en general se les conoce simplemente como* ***estrategia*** *[17].* △

Recordemos que la suma en $\mathbb{Z}_2^{mn}$ es conmutativa, así que, por conveniencia, podemos escoger la secuencia de índices $n(1), n(2), \ldots, n(N)$ como no decreciente. Más aún, como $v + v = 0$ para todo $v \in \mathbb{Z}_2^{mn}$, podemos incluso escoger la sucesión como estrictamente creciente dado que en una solución óptima ningún botón necesita ser activado más de una vez[11] [17].

Ejemplo 2.3.10. *Representamos la estrategia usada en el ejemplo anterior como*

$$a_1, a_7, a_{13}, a_{19}, a_{25}.$$

Verificamos además que, para este caso,

$$a_1 + a_7 + a_{13} + a_{19} + a_{25} + y = f.$$

$\Diamond$

Naturalmente, esto significa que $N \leq mn$. Para saber exactamente qué botones debemos y no debemos presionar en esa solución óptima, podemos lograr una forma elegante de la ecuación que contenga exactamente mn sumandos, haremos esto definiendo un conjunto $x_1, \ldots, x_i, \ldots, x_{mn}$ de elementos de $\mathbb{Z}_2$ a los que llamaremos ***incidencias***. Estos indicarán si el botón de índice i interviene o no en la solución del rompecabezas. Para cada índice i notamos por convención que [17]:

$$1 \ \rightarrow x_i \text{ interviene en la Sol.} \qquad \text{y} \qquad 0 \ \rightarrow x_i \text{ no interviene en la Sol.}$$

Notaremos la presencia de una acción en una solución usando el producto escalar $(*)$ de $\mathbb{Z}_2$ sobre $\mathbb{Z}_2^{mn}$. Así, cada acción a_i operará con un x_i correspondiente que conservará en la sucesión las acciones que intervienen (producto por unidad) y anulará las que no lo hacen (producto por cero) [17].

Por convención y facilidad de lectura notaremos el producto escalar como yuxtaposición de símbolos. Por ejemplo, escribiremos ab en vez de $a * b$. Reescribiremos la ecuación anterior como una combinación lineal equivalente, así [3][11][17]:

$$x_1 a_1 + \cdots + x_{mn} a_{mn} + y = f.$$

Organizamos las incidencias en un vector $x = (x_1, \ldots, x_{mn})^T \in \mathbb{Z}_2^{mn}$ al que llamaremos ***vector de estrategia***[12]. Esto lo hacemos buscando una forma de la ecuación en la que este vector tome el rol de las estrategias que habíamos definido.

Ejemplo 2.3.11. *La ecuación del ejemplo anterior puede ser reescrita como*

$$\begin{array}{ccccccccccc} 1a_1 & + & 0a_2 & + & 0a_3 & + & 0a_4 & + & 0a_5 & & \\ +0a_6 & + & 1a_7 & + & 0a_8 & + & 0a_9 & + & 0a_{10} & & \\ +0a_{11} & + & 0a_{12} & + & 1a_{13} & + & 0a_{14} & + & 0a_{15} & & \\ +0a_{16} & + & 0a_{17} & + & 0a_{18} & + & 1a_{19} & + & 0a_{20} & & \\ +0a_{21} & + & 0a_{22} & + & 0a_{23} & + & 0a_{24} & + & 1a_{25} & + y = f, & \end{array}$$

donde $x = (1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1)^T$. $\Diamond$

[11] Esta es, en forma de ecuación, la primera observación con la que abrimos esta sección.
[12] Creamos este término a falta de un nombre dado por la literatura.

Como para todo $k \in \mathbb{Z}_2^{mn}$ se cumple que $k = -k$, restando y a ambos lados de la ecuación tenemos que [17]

$$x_1 a_1 + \cdots + x_{mn} a_{mn} = f + y.$$

Escribiremos el lado izquierdo de la ecuación como producto matricial. Si hacemos que A sea la matriz cuya i-ésima fila es la acción a_i transpuesta y acomodamos los x_i de forma ordenada en x, nuestra ecuación pasará a ser [11][17]

$$Ax = f + y,$$

donde

$$A_{mn\times mn} = \begin{bmatrix} a_1^T \\ \vdots \\ a_{mn}^T \end{bmatrix} = \begin{bmatrix} (a_{1,1} & \dots & a_{1,mn}) \\ & \vdots & \\ (a_{mn,1} & \dots & a_{mn,mn}) \end{bmatrix}.$$

Por último, como el objetivo del juego es llegar a $f = (0, \dots, 0)^T \in \mathbb{Z}_2^{mn}$, entonces, la ecuación se convertirá en [3][11][17][21]

$$Ax = y. \tag{2.2}$$

A la ecuación 2.2 la llamaremos el *modelo algebraico* de Lights Out [24].

La matriz $A_{mn\times mn} = [a_{i,j}]$ puede construirse directamente[13] como matriz de adyacencia para cualquier par de botones i, j del tablero través de una relación binaria $\mathcal{R}$ de

"presionar el botón de índice i alterna al de índice j"

donde

$$a_{i,j} = \begin{cases} i\mathcal{R}j, & a_{i,j} = 1 \\ i\not\mathcal{R}j, & a_{i,j} = 0. \end{cases}$$

Por esta razón, a A se le llama la ***matriz de adyacencia*** del modelo algebraico [19].

La teoría de grafos[14] permite modelar los Lights Out a través de una ecuación idéntica al modelo algebraico y construye directamente a A usando a $\mathcal{R}$ como se mostró anteriormente [19]. De hecho, el término *matriz de adyacencia* proviene de esa rama de las matemáticas. [19][24].

En la sección 5.2 discutiremos que, al menos para los Lights Out cuadrados, las matrices de adyacencia se pueden construir de forma sistemática y mostraremos su forma general. No mostraremos esa forma en esta sección pues este es un resultado obtenido a través de la teoría de grafos y no sabemos si se puede obtener a través de técnicas algebraicas.

Finalizaremos esta sección con unos comentarios:

- A pesar de que la Ec. 2.1 no es suficiente para analizar la resolubilidad de un Lights Out, sí es suficiente para simularlo. Usar esta ecuación como directriz es especialmente útil para desarrollar aplicaciones que simulen el juego, de hecho, esta es la lógica de programación que utiliza la aplicación que introdujimos en la sección 2.1. La única

[13] Sin usar todo el procedimiento presentado en esta sección.
[14] Sobre la que hablaremos en la sección 5.2.

consideración es que la aplicación utiliza una forma matricial de esa ecuación donde los botones están rotulados por dos índices correspondientes a los que tendrían en una matriz de igual tamaño a la del tablero.

- Esta forma matricial de la ecuación se usa porque permite definir las configuraciones y las acciones de forma intuitiva y sin necesidad de transformar coordenadas. De hecho, cuando se van a usar vectores como en el modelo algebráico, resulta más sencillo construirlos desde matrices y luego transformarlas a vectores. La función `flatten` de Python hace esta transformación resultando en un vector con el mismo orden de indexación que usa el modelo algebraico.
- La Ec. 2.2 es válida para los Lights Out, otros rompecabezas luminosos tendrán asociadas ecuaciones propias. No obstante, muchas partes de este desarrollo son extensibles a otros casos. Haremos algunos comentarios sobre como hacerlo en la sección 3.2.
- A ecuaciones matriciales como estas con matrices cuyos componentes son elementos de alguno de los $\mathbb{Z}_n$ (así n no sea primo) se les acostumbra llamar *ecuaciones matriciales modulares*.

2.4. Resolubilidad

En la sección 2.3 se ha desarrollado un modelo para los Lights Out que nos permite describirlos a través de la ecuación 2.2. Podemos sintetizar esa descripción afirmando que:

Una sucesión de acciones $x \in \mathbb{Z}_2^{mn}$, $x = (x_1, \ldots, x_{mn})^T$, permitirá resolver el patrón de luces $y \in \mathbb{Z}_2^{mn}$ si y solo si $Ax = y$ [17].

No obstante, hubiera sido poco práctico desarrollar este modelo para que solo permitiera describir el rompecabezas cuando la ecuación 2.1 también puede hacerlo. Sobre todo teniendo en cuenta que que A crece muy rápido con el tamaño del tablero, hemos visto que un tablero de tamaño $m \times n$ induce siempre una matriz de adyacencia de tamaño $mn \times mn$, haciendo que trabajar con ella sea bastante más engorroso que con una suma de vectores [12].

Afortunadamente, nuestro modelo fue construido precisamente para permitir resolver los Lights Out: es desde luego una ecuación matricial donde x es incógnita y A y y están dados. Siendo así, estamos facultados para aplicar los métodos de solución típicos del álgebra lineal si usamos las operaciones del cuerpo en que están definidas sus componentes ($\mathbb{Z}_2$) [12].

Por la estructura del modelo, podemos esperar que los resultados y planteamientos que nos esperan no sean más que un reflejo del álgebra lineal sobre los Lights Out. De esta forma, como consecuencia del teorema 1.3.1 tenemos que:

Teorema 2.4.1. *Si la matriz de adyacencia A de un tablero es invertible, podremos solucionar todo su espacio de configuración, es decir, todos sus posibles estados iniciales y. Así, x siempre podrá ser expresado como $x = A^{-1}y$ [19].*

Pero, ¿que hacemos entonces si esa matriz no es invertible?

Ejemplo 2.4.1. *La matriz de adyacencia A del Lights Out clásico valida que $det(A) = 0$, por lo que **no** es invertible.*

*La matriz de adyacencia A del Lights Out 3×3 valida que $det(A) \neq 0$, por lo que **si** es invertible.* ◊

Observamos del ejemplo anterior que la matriz del Lights Out clásico no es invertible y que la del Lights Out 3×3 sí lo es. Eso significa que:

- *No todos los patrones iniciales tienen solución. Algunos la tendrán, otros no.*
- *Esta aproximación a través de la matriz invertible no nos va a ser tan útil.*

Esto no significa que en un Lights Out con una matriz de adyacencia no invertible todos los patrones iniciales sean imposibles de resolver, siempre habrá patrones iniciales que si puedan ser resueltos. Para entender esto es conveniente que nos apoyemos en nuestra intuición sabiendo que:

Si un grupo de botones se presiona en un tablero vacío para generar una configuración, iniciar con esa configuración y presionar esos mismos botones resolverá el juego [11][24].

Esa intuición motiva la siguiente definición:

Definición 2.4.1. *Un patrón inicial y es **ganable (resoluble)** si existe una estrategia x que permite alcanzar el objetivo del juego, es decir, que satisface $Ax = y$ [21].* △

Además, preguntar si todos los patrones iniciales motiva la que le sigue:

Definición 2.4.2. *Un Lights Out de dimensiones dadas es **completamente ganable (completamente resoluble)** si todas sus configuraciones iniciales son ganables, es decir, si su matriz de adyacencia A es invertible [21].* △

Como consecuencia de esta situación, necesitamos un criterio para saber qué patrones son ganables y cuales no lo son. Para encontrarlo usaremos el siguiente resultado, consecuencia del teorema 1.3.2:

Lema 2.4.1. *Un patrón inicial y será ganable si y solo si pertenece al espacio columna de A ($col(A)$) [24].*

Para continuar planteémonos lo siguiente:

Habíamos dicho en la construcción del modelo (sección 2.3) que partiendo de un estado inicial, si ejecutábamos una sucesión de movimientos llegaríamos a un estado actual pero, ¿y si usamos jugadas no redundantes y el estado final coincide con el inicial?

Eso nos motiva la definición de unos elementos cuya base coincide con el núcleo de A ($ker(A)$):

Definición 2.4.3. *Un **patrón estacionario** es una sucesión de movimientos x que no produce cambios en el tablero después de ser aplicada, es decir, que satisface $Ax = 0$ [19].* △

Notaremos los patrones estacionarios de un tablero por medio de la letra n con subíndices. Ahora necesitaremos recordar detalles sobre la estructura de las matrices de adyacencia: esas matrices se construyen usando a las acciones como filas. En vez de pensarlas así, pensémoslas por ahora simplemente como matrices. Así:

$$A = (\alpha_{i,j}) \in \mathbb{Z}_2^{mn \times mn},$$

donde, como ya hemos dicho, $\alpha_{i,j}$ representa el efecto de la acción del botón con índice i sobre el botón con índice j. Es fácil para nosotros ver que, como nos dice Scherphuis (Ref. [21]), para la matriz de adyacencia A de cualquier juego Lights Out:

- *Como presionar un botón alternará su propio estado, A será siempre reflexiva.*
- *Si presionamos un botón y eso alterna otro, veremos que, al presionar el otro botón se alterna el primero. Así, A será siempre simétrica.*

Tenemos ahora suficientes herramientas para derivar un criterio para la resolubilidad de un patrón inicial:

Teorema 2.4.2. *Un patrón inicial y será ganable si y solo si es ortogonal a la base de todos los patrones estacionarios del tablero.*

Demostración. Como vimos en el lema anterior, y será ganable si y solo si pertenece al espacio columna de A. A es simétrica, y por tanto el espacio columna de A equivale a su espacio fila ($col(A) = row(A)$). Como $row(A)$ es el complemento ortogonal de $ker(A)$ y la base de todos los patrones estacionarios es una base de $ker(A)$, entonces y será ganable si y solo si es ortogonal a la base de todos los patrones estacionarios [24]. □

Ahora estamos capacitados para, a manera de resumen, responder las preguntas que introducidas antes de empezar a construir el modelo (sección 2.3):

1. No cualquier patrón inicial tiene solución, los que tienen solución son aquellos que son ortogonales a la base de todos los patrones estacionarios del tablero. Para ver si una configuración es ganable, simplemente realizamos su producto escalar canónico ($\cdot$) con todos los vectores de esa base.
2. Depende:

   a) Si su matriz de adyacencia A es invertible, podemos encontrarle solución con la matriz inversa. Podemos verificar que sea invertible usando el determinante.

   b) Si A no es invertible y sabemos que el patrón inicial tiene solución, como nos es posible usar los métodos de solución típicos del álgebra lineal, podremos encontrarla haciendo reducción de Gauss-Jordan sobre la matriz aumentada

   $$[A \mid y]^{15}.$$

   También podemos usar este método si A es invertible, aunque en este caso sería preferible usar la matriz inversa para poder tener de una vez solución para todos los patrones iniciales.

[15] O con métodos de pseudoinversa que se escapan del alcance de esta disertación.

c) En caso de que el patrón inicial no tenga solución, Gauss-Jordan es suficientemente robusto como para indicárnoslo. Lo verificaremos observando que la matriz escalonada reducida por filas de A (*rref*) resulta con filas nulas[16] [12].

Ejemplo 2.4.2. *Para el Lights Out clásico, la base de los patrones estacionarios es* $\{n_1, n_2\}$*, donde*

$$n_1 = (1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1)^T \text{ y}$$
$$n_2 = (1,1,0,1,1,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,1,1,0,1,1)^T.$$

Podemos ver que el criterio de ganabilidad que derivamos se reduce en este caso a decir que un patrón inicial y *es ganable si y solo si* $y \cdot n_1 = y \cdot n_2 = 0$. ◊

Después del trabajo realizado, parece como si hubieramos concluido el estudio que nos propusimos hacer sobre los Lights Out. No obstante, abrimos un panorama nuevo de resultados a partir del concepto de resolubilidad que acabamos de introducir. Usaremos el resto de la sección para mostrar a manera de contexto algunos de los más importantes (y también porque nos serán útiles posteriormente).

Los siguientes resultados nos serán de mucha ayuda para introducir la reproducción de resultados (sección 4.1.1). Para motivar el primero veamos el siguiente ejemplo:

Ejemplo 2.4.3. *Cuando se resuelve el ejemplo anterior a través de Gauss-Jordan, se aprecia que* A *resulta teniendo dos variables libres:* x_{24} *y* x_{25}*. Así, cualquier patrón inicial ganable tendrá* 4 *soluciones distintas, es decir, el número de patrones estacionarios de* A*, porque podemos escoger arbitrariamente a* x_{24} *y* x_{25} *de* $\mathbb{Z}_2$*.* ◊

Teorema 2.4.3. *Un estado inicial ganable* y *tiene tantas soluciones como patrones estacionarios hay.*

Demostración. Sea T un tablero con patrones estacionarios, x una solución, A matriz de adyacencia y $n_1, \ldots, n_i, \ldots, n_e$ $(i = 1, \ldots, e)$ base de sus patrones estacionarios. Consideremos primero el conjunto

$$E = \{c_1 n_1 + \cdots + c_e n_e\},$$

donde $c_i \in \mathbb{Z}_2$ $(i = 1, \ldots, e)$. Este es el conjunto de patrones estacionarios de T y tiene un tamaño de 2^e. Consideremos ahora el conjunto

$$S = \{x + c_1 n_1 + \cdots + c_e n_e\},$$

veremos todos sus elementos resuelven el rompecabezas. En efecto: si multiplicamos cualquier elemento de S a izquierda por A veremos que

$$\begin{aligned} A(x + c_1 n_1 + \cdots + c_e n_e) &= Ax + c_1 An_1 + \cdots + c_e An_e \\ &= y + 0 + \cdots + 0 \\ &= y. \end{aligned}$$

Así, este será el conjunto de soluciones de T y, como el conjunto de patrones estacionarios, tendrá un tamaño de 2^e [24]. □

[16] Esto es equivalente a que el determinante de A sea nulo $(det(A) = 0)$.

Del teorema anterior se sigue directamente el siguiente resultado:

Corolario 2.4.3.1. *Un estado inicial ganable y tiene 2^e soluciones posibles, donde e es el número de elementos de la base de los patrones estacionarios. [24].*

Ejemplo 2.4.4. *Para el Lights Out clásico, si x es solución de un patrón inicial, entonces todas sus soluciones son: x, $x + n_1$, $x + n_2$ y $x + n_1 + n_2$.*

Podemos ver con este ejemplo que si conocemos la base de patrones estacionarios de un Lights Out, podremos conocer todos sus patrones estacionarios haciendo las 2^e combinaciones lineales de estos. ◊

El siguiente teorema es importante en la medida en que nos muestra realmente cuántas configuraciones iniciales son ganables en un Lights Out:

Teorema 2.4.4. *De las 2^{mn} configuraciones iniciales posibles, solo 2^{mn-e} son ganables, donde e es el número de elementos de la base de los patrones estacionarios.*

Demostración. Sabemos como consecuencia del teorema 2.2.2 que en un tablero de tamaño mn habrá 2^{mn} configuraciones posibles. El teorema 2.4.2 enuncia que una configuración será ganable si y solo si es ortogonal a la base de todos los patrones estacionarios, es decir, el producto escalar de la configuración por cualquiera de los dos vectores debe dar 0 como resultado.

Sea y la configuración $y = (y_1, \ldots, y_{mn})^T$, como cada componente de y pertenece a $\mathbb{Z}_2$ sabemos que la operación que decidirá finalmente si y es ortogonal al primer elemento de la base de los patrones estacionarios, llamémoslo n_1, será la última que se realice entre un elemento de y y un elemento no nulo de n_1. Luego uno de los elementos de y tendrá que tener un valor fijo para que y sea ortogonal a n_1. Es decir, habrá 2^{mn-1} configuraciones ortogonales a n_1.

Este proceso se debe repetir sistemáticamente para conseguir que y sea ortogonal a todos los elementos de la base de los patrones estacionarios, es decir, se debe dar un valor fijo a e posiciones. Por tanto, solo 2^{mn-e} configuraciones son ortogonales a todos los elementos de la base de los patrones estacionarios, y por tanto, ganables [24]. □

El siguiente corolario es consecuencia directa del teorema anterior:

Corolario 2.4.4.1. *1 de cada 2^e configuraciones iniciales posibles son ganables.*

Ejemplo 2.4.5. *Para el caso que acompañamos: como $e = 2$, apreciamos que solo 1 de cada 4 estados iniciales posibles son ganables. Además, como $r = 25 - 2$, verificamos que cada estado inicial ganable tiene 4 soluciones posibles.*

Esto nos muestra de forma palpable que conforme menos patrones iniciales ganables tenemos, más soluciones resultan para los que sí lo son. ◊

Terminaremos la sección mostrando el criterio de resolubilidad, un resultado que aunque no es práctico, nos permitirá obtener resultados teóricos importantes en la sección 4.2.2:

Teorema 2.4.5. *(criterio de resolubilidad) Un patrón de luces es resoluble si y solo sí el número de luces que tiene en común con cada patrón estacionario es par.*

Demostración. Por definición, presionar los botones de un patrón estacionario alternará cada luz un número par de veces de forma que al final no ocurra nada en la configuración del tablero. Por lo tanto, cada una de las luces será afectada por un número par de los botones en el patrón estacionario. El juego es simétrico, por lo que el recíproco también vale: cada uno de los botones afectará un número par de luces en el patrón estacionario. [21]. □

Estos son todos los resultados sobre resolubilidad que mostraremos, no obstante, este tema tiene mucho más para ofrecer. Invitamos al lector a leer otras referencias para ahondar en él, especialmente las Refs. [24] y [21]. Ya que conocemos bastante bien los Lights Out, ampliemos nuestro panorama sobre los rompecabezas luminosos.

Capítulo 3

Variantes

3.1. Variantes de Lights Out

En la sección 2.2 introdujimos los Lights Out como una clase entera de rompecabezas construidos generalizando el Lights Out clásico a tableros de dimensión arbitraria y, naturalmente, manteniendo intactas todas sus otras reglas. Además de esto, en la sección 1.4 pudimos ver que el juego Orbix se juega en una esfera, por lo que no debería sorprendernos que haya rompecabezas luminosos fuera de la clase Lights Out.

Motivados en parte por la confusión de la que hemos hablado en las secciones 1.1 y 2.2 y en parte por Refs. como [20] y [22] definimos:

Definición 3.1.1. *Una **variante de Lights Out** es todo rompecabezas luminoso que **no** sea un Lights Out.* △

Ilustramos en la Fig. 3.1 la clasificación y algunos ejemplos de los rompecabezas luminosos que hemos inducido con esta definición a través de conjuntos.

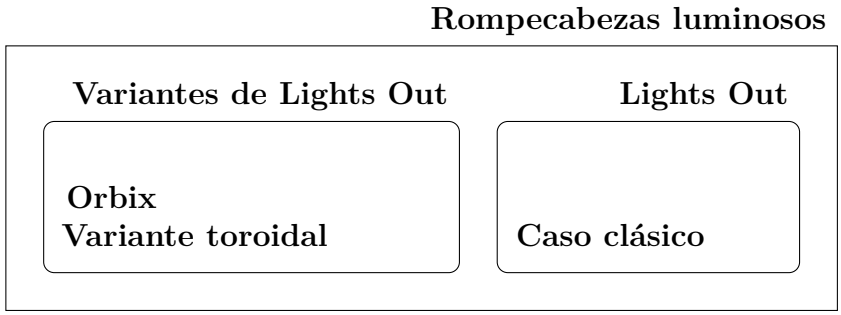


Figura 3.1: Clasificación de los rompecabezas luminosos.

Pareciera que no pertenecer a la clase Lights Out es un criterio demasiado amplio como para poder hablar con precisión de las variantes. No obstante, en la sección 2.2 introdujimos nociones sobre los elementos de la definición de los Lights Out y Martin Kreh en la Ref. [22] nos dice qué elementos de esa definición podemos cambiar para constituir una variante:

1. La forma y cantidad de botones del tablero.

2. La cantidad de estados posibles para las luces.
3. La definición de las acciones.
4. La definición del objetivo[1].

Sabiendo esto, podemos generar variantes de forma sistemática partiendo de los Lights Out y haciendo cambios a uno o varios elementos de su definición a la vez. De esta forma, podemos construir un panorama muy nutrido de las variantes de Lights Out [22]. En la siguiente sección haremos algunos comentarios con respecto a los posibles cambios que podemos hacer en la definición de un juego.

3.1.1. Estados posibles

Solo hay una forma en la que se puede modificar la cantidad de estados posibles en un rompecabezas luminoso: agregando más. En efecto, un tablero con menos de dos estados está estático y, por tanto, es inutil como rompecabezas [24].

La convención cuando se construyen juegos con más de dos estados es mantener el estado de apagado y sustituir el estado de encendido por varios colores. Un rompecabezas construido de esta forma tendrá el estado de apagado y otros cuantos más [21]. Esta forma de agregar estados a los rompecabezas es intuitiva porque alternar entre estar apagado o iluminar con distintos colores es algo que puede hacer un tablero electrónico.

Ejemplo 3.1.1. *Los estados posibles para un tablero que tenga cuatro podrían ser blanco, azul, rojo y apagado.* ◊

Una forma especial en la que un juego puede alternar entre sus estados es a través de un ciclo. Precisaremos esta idea con la siguiente definición:

Definición 3.1.2. *Sea un rompecabezas luminosos con un conjunto de estados posibles* $e = \{e_1, \ldots, e_n\}$ *donde* $n \geq 2$. *Si los estados de las luces son alternados por las acciones siguiendo el ciclo*

$$e_1 \to e_2 \to \cdots \to e_n \to e_1,$$

entonces diremos que el rompecabezas es ***cíclico****.* △

Como curiosidad, notamos que los Lights Out son rompecabezas cíclicos.

3.1.2. Forma y cantidad de botones del tablero

En la sección 2.2 introdujimos la noción de rompecabezas luminosos con tableros que no están definidos por cuadrículas. Esta posibilidad de evidente cuando vemos ejemplos con tableros que distribuyen sus botones en formas muy diversas: la esfera de Orbix (Fig. 1.1d) y la superficie de un cubo son ejemplos de esto (Fig. 3.4d) [21].

La distribución de los botones en un tablero es prácticamente libre. No obstante, podemos generar tableros en el plano de forma sistemática siguiendo las siguientes indicaciones:

[1]Este elemento fue una sugerencia propia.

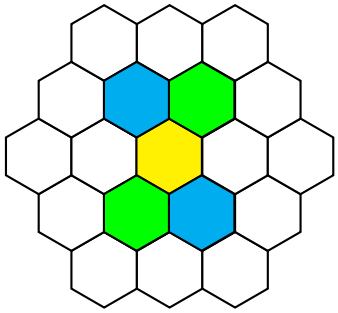


Figura 3.2: Ejemplo de rompecabezas de anillos hexagonales.

1. Usando botones que tengan formas que teselen el plano.
2. Usando botones que tengan la misma forma y tamaño.
3. Distribuyendo los botones de forma que pavimenten una porción del plano.

Si seguimos estas indicaciones, al tablero que resulte lo llamaremos ***tablero teselar***[2]. Podemos encontrar ejemplos de tableros de este tipo en las Figs. 3.4c y 3.2.

Los diseñadores de rompecabezas luminosos han tomado ventaja de esto y así, podemos encontrar abundantes ejemplos de rompecabezas con botones triangulares, cuadrados y hexagonales, entre otras [21]. En la Fig. 3.2 podemos encontrar un ejemplo de rompecabezas con botones hexagonales de tres estados.

3.1.3. Definición de movimientos

En la sección 2.2 introdujimos la idea de que las acciones alternan los estados de los botones que están comprendidos en la intersección de una figura en forma de cruz (+) y el tablero. Podemos generalizar esta noción de *área de efecto*[3] para construir rompecabezas de forma sistemática.

Definición 3.1.3. *Un **área de efecto** es una región constante de botones en un tablero cuyos botones tienen una forma y distribución que tesela el plano, centrada en uno de ellos y con un efecto definido para cada uno de los botones de la región.* △

En la Fig. 3.3 podemos ver algunos ejemplos de área de efecto para rompecabezas con botones rectangulares. Los rompecabezas para los que fueron definidas tienen dos estados, por lo que el efecto que tiene el área en cada uno de los botones en los que actua es simplemente alternar el estado.

Para que el lector se familiarece con este concepto, al menos en tableros con forma de cuadrícula, puede experimentar jugando con ellos usando la aplicación disponible a través del vínculo

`https://www.jaapsch.net/puzzles/javascript/lightjvr.htm`

[2]Introducimos este nombre a falta de un nombre en la bibliografía.
[3]Este nombre es un reflejo de la terminología que se usa en el diseño de videojuegos.

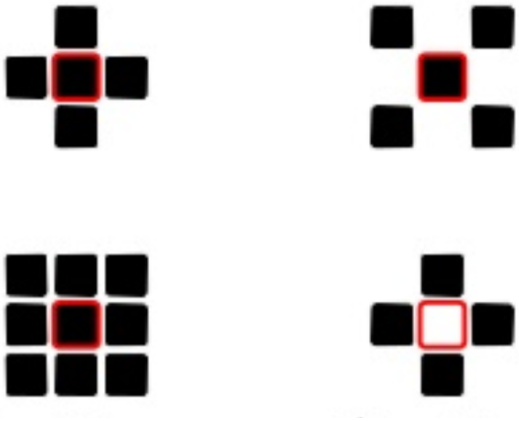


Figura 3.3: Ejemplos de área de efecto para rompecabezas con botones rectangulares.

En tableros cuadriculados podemos hacer un símil con la topología considerando al tablero como un espacio al que se puede deformar y así, cerrar algunos de sus bordes. De esta forma, obtendremos tableros que asemejarán geometría de cilindro, el toroide e incluso de la banda de Möbius [26].

Realmente lo que ocurre a nivel de la mecánica de las acciones cuando cerramos un borde de un tablero es que los botones de un borde cerrado consideran como vecinas a las casillas del borde opuesto, por lo que la intersección del área de efecto con el tablero no deja tantas casillas por fuera. Cuando se cierran los dos bordes no se deja ninguna por fuera [21].

3.2. Comentarios sobre el modelo algebraico

En esta sección haremos algunos comentarios sobre cómo modificar el modelo algebraico para que funcione con algunos tipos de variantes:

- La versión del modelo algebraico que en general podremos adaptar para trabajar con variantes es
$$Ax = f - y.$$
La causa de esto es que la ecuación $Ax = y$ descansa en el supuesto de $\mathbb{Z}_2^{mn}$ de que $y = -y$ y el supuesto de los Lights Out de que $f = (0, \ldots, 0) \in \mathbb{Z}_2^{mn}$.
- En variantes con tableros no rectangulares, no existe en general un orden natural para asignarle un índice a cada casilla[4]. Para asignarle índices a un tablero sin un orden natural de indexación la mejor alternativa es numerarlos a voluntad y construir el modelo prestando atención a esos índices. En una circunstancia como esta, el uso de un solo índice es una buena idea, ya que es más fácil prestar atención a un índice arbitrario que a dos (o más).
- Para variantes cuyos botones producen acciones diferentes a la cruz + con la que estamos familiarizados, el único cambio que debemos hacerle al modelo es construirles

[4]Los anillos hexagonales como el de la Fig. 3.2 son una excepción a esto

su matriz de adyacencia para que refleje esta definición de las acciones. Si estas acciones están definidas por un área de efecto será posible incluso construirlas algoritmicamente.

- Para variantes cíclicas con más de dos estados, digamos que sean k, habrá dos modificaciones: la primera es que las matrices de adyacencia seguirán siendo matrices lógicas, lo que cambiará es el significado de los ceros y unos, que estarán asignados como:

$$1 \rightarrow \text{alterna al estado siguiente} \qquad \text{y} \qquad 0 \rightarrow \text{no alterna.}$$

  El segundo cambio es que las componentes de x, f y y ahora vivirán en $\mathbb{Z}_k$, por lo que sus componentes podrán tomar cualquiera de sus valores y, además, las operaciones numéricas que hagamos ahora deberán hacerse bajo las definiciones de $\mathbb{Z}_k$.

Como curiosidad, agregamos que la definición de suma y producto en $\mathbb{Z}_2$ coinciden de forma respectiva con las definiciones de los operadores lógicos XOR y AND. A pesar de los dispositivos electrónicos digitales se construyen usando compuertas lógicas, esto no constituye evidencia de la forma en que las consolas como *MERLIN* están cableadas o programadas.

3.3. Algunas variantes

Después de haber introducido el tema de las variantes, mostraremos algunas de ellas para apreciar qué tan diferentes pueden ser estos entre sí:

(a) *Lights Out Mini* (b) *Lights Out 2000*

(c) *Gamze Lights Out* (d) *Lights Out Cube*

Figura 3.4: Ejemplos de rompecabezas luminosos.

- Una de las variantes más conocidas es la toroidal. En ocasiones se le llama *Lights Out toroidal*. Es idéntica a los Lights Out a excepción de un detalle: en ella las columnas izquierda y derecha se consideran vecinas, así como las filas superior e inferior. Como

curiosidad, sobre esta variante aplican todos los resultados expuestos en el capítulo anterior, a excepción del criterio de resolubilidad. Para construir su versión del modelo algebráico solo debemos construirle la matriz de adyacencia apropiadamente.

Para que el lector se familiarice con ella, construimos una aplicación que los simula como la que nombramos anteriormente. Se encuentra en la carpeta **Lights Out toroidal** del repositorio GitHub y su uso es exáctamente idéntico.

- La consola *Lights Out Mini* es un ejemplo de rompecabezas Lights Out toroidal en un tablero de 4×4 (figura 3.4a).
- Algunos rompecabezas tienen una definición de objetivo diferente a apagar todas las luces, la consola *MERLIN* constituye un ejemplo de esto. Esta además tiene una definición de movimiento diferente según el botón que se presione.
- La consola *Lights Out 2000* es un ejemplo de rompecabezas cíclico con tres estados. Estos son respectivamente rojo, verde y apagado (figura 3.4b).
- La esfera *Orbix* es un ejemplo de rompecabezas luminoso que no solo es tridimensional, sino que tiene una matriz de adyacencia irreflexiva.

Hemos introducido el tema de las variantes de Lights Out, así, tenemos herramientas suficientes para poder comprender la reproducción de resultados computacionales que vamos a hacer a continuación. Pasaremos entonces a discutir esta y otras aplicaciones de la teoría.

Capítulo 4

Resultados, restricciones y métodos

4.1. Aplicaciones computacionales

Nota: En la carpeta **cuaderno** del repositorio GitHub podemos encontrar todos los archivos a los que nos referiremos en esta sección.

Como forma de aplicar la teoría que hemos desarrollado hasta ahora, expondremos dos aplicaciones computacionales relacionadas con los Lights Out y sus variantes toroidales. Para poder entender su importancia consideremos que:

"Los patrones estacionarios son elementos del nucleo de la matriz de adyacencia $A_{mn\times mn}$ de los Lights Out. Así, el número e de elementos de su base es la nulidad $nul(A_{mn\times mn})$ de esas matrices $(e = nul(A_{mn\times mn}))$".

Considerando esto, podemos reformular algunos resultados anteriores:

- Un estado inicial ganable tiene $2^{nul(A_{mn\times mn})}$ soluciones posibles (corolario 2.4.3.1).
- De las 2^{mn} configuraciones iniciales posibles, solo $2^{mn-nul(A_{mn\times mn})}$ son ganables (teorema 2.4.4).
- 1 de cada $2^{nul(A_{mn\times mn})}$ configuraciones iniciales posibles son ganables(corolario 2.4.4.1).

Ahora, recordemos que:

"El método de reducción de Gauss-Jordan nos permite hallar la matriz rref de A. Además, su número de filas nulas es precisamente $nul(A_{mn\times mn})$".

Si interpretamos todo esto junto veremos que estamos en capacidad de hallar, usando un método práctico, un dato de gran importancia para los Lights Out y para sus variantes toroidales. No olvidemos que, por sus semejanzas, este planteamiento también les aplica.

Las aplicaciones de las que hablaremos a continuación requirieron que desarrollaramos algo de código, para esto nos servimos del lenguaje Python[1]. Todo este código está disponible en

[1]Lenguaje de alto nivel que por diseño facilita la lectura y comprensión del código.

el apéndice A y en un archivo de *Jupyter Notebook*[2] de nombre **Lights Out**. La ejecución de este código genera los resultados que expondremos en esta sección, exceptuando los del apéndice D.

4.1.1. Reproducción de resultados

La primera aplicación que expondremos será la reproducción de los resultados expuestos por Anderson y Feil en la Ref. [11], a la que llamaremos el *artículo base*.

Nota: Por comodidad de notación, a lo largo de esta sección llamaremos C_n a la matriz de adyacencia $A_{n^2\times n^2}$ de un Lights Out (cuadrado) de orden n y T_n a la de un Lights Out toroidal del mismo orden.

Los resultados expuestos por el artículo base consisten en tablas que contienen los valores de $nul(C_n)$ y $nul(T_n)$ para n desde 2 hasta 21. Agregamos que los autores de ese artículo obtuvieron esos resultados a través de métodos computacionales que no mencionaron.

Logramos superar los resultados obtenidos por el artículo base, pudiendo llegar a órdenes n desde 1 hasta 60. Podemos encontrar estos resultados en el apéndice B organizados en tablas. No pudimos llegar más lejos por limitaciones de la fecha límite del proyecto.

Además de esto, a partir de estos datos generamos gráficas ubicadas en el apéndice C, no pudimos encontrar algún patrón seguido por estos datos. No obstante, vale la pena enunciar que [27][28]:

- C_n siempre es par y $C_n \leq n$ para todo n.
- T_n siempre es múltiplo de 4 y $T_n \leq 2n$ para todo n.

Para visualizar el coste computacional que requirió obtener estos resultados, calculamos el tiempo empleado por el computador en construir y reducir[3] las matrices C_n y T_n de todos los órdenes tratados, podemos encontrar estos resultados en gráficas ubicadas en el apéndice D y también en un archivo EXCEL de nombre **Gráficas de tiempo** ubicado en la carpeta **resultados** del repositorio.

A pesar de que los órdenes de complejidad computacional de la construcción de una matriz y su reducción son respectivamente de $O(n^2)$ y $O(n^3)$ [12], podemos ver que, en ambas gráficas, los tiempos de ambos procesos son aproximados por regresiones polinomiales de orden cúbico. Esto sugiere una posible falta de eficiencia de los algoritmos que desarrollamos para la obtención de resultados.

No obstante, con un tiempo aproximado de 52 y de 73 min. para las reducciones de C_{60} y T_{60} respectivamente (apéndice D), consideramos que aún no nos vimos limitados por la

[2]Tipo de archivo que se comporta como un cuaderno digital, permite hacer comentarios y visualizar directamente imágenes y resultados de consola.

[3]Por el método de Gauss-Jordan, en adelante nombrado como *reducción*.

complejidad computacional del problema. Deberíamos poder llegar más lejos con más tiempo.

Lastimosamente, no pudimos romper el límite impuesto por las referencias consultadas que más lejos han llegado en esta empresa, puesto que muestran resultados para ordenes n de 1 a 1000 cada una [27] y [28].

Consideramos que en un orden tan grande como 1000 si nos veríamos limitados por la creciente complejidad computacional del problema. Lo más probable es que para tratarlo nos viéramos en la necesidad de recurrir a multiprocesamiento[4] o a optimizaciones basadas en la estructura de las matrices dado que, al menos para los Lights Out, las conocemos (sección 5.2) [12].

4.1.2. Problema del triángulo

Antes de exponer la segunda aplicación que expondremos, introduciremos una definición:

Definición 4.1.1. *Diremos que un tablero rectangular de tamaño $m \times n$ es **más pequeño** que tablero cuadrado de orden p si $m \leq p$ y $n \leq p$.* △

De forma intuitiva, un tablero A es más pequeño que B si, al poner a A sobre B y los alineamos por una esquina, ninguna parte de A queda por fuera de los límites de B.

Nota: Por comodidad de notación, en lo que queda de esta sección llamaremos $C_{m,n}$ a la matriz de adyacencia $A_{mn\times mn}$ de un Lights Out de tamaño $m \times n$ y $T_{m,n}$ a la de un Lights Out toroidal del mismo tamaño.

Algunas referencias como las Refs. [21] y [29] tratan una extensión de la aplicación anterior a la que llamaremos el ***problema del triángulo***[5], lo definiremos a continuación:

Definición 4.1.2. *Sean i, j respectivamente el número de filas y columnas de K, un tablero cualquiera, el **problema del triángulo** consiste en hallar $nul(C_{i,j})$ o $nul(T_{i,j})$ para todos los K más pequeños que un tablero cuadrado de orden n dado.* △

Como un Lights Out (y su variante toroidal) de tamaño $m \times n$ se comporta igual que su contraparte de tamaño $n \times m$ girada 90°, sus matrices de adyacencia serán iguales salvo la nomenclatura de los botones. Esto quiere decir que $nul(C_{i,j}) = nul(C_{j,i})$ (y que $nul(T_{i,j}) = nul(T_{j,i})$).

De esta forma, no necesitamos hallar $nul(C_{i,j})$ (o $nul(T_{i,j})$) para todos los i, j que cumplan $i \leq n$ y $j \leq n$, bastará con hallar la mitad. Es decir, solo aquellos que satisfagan $1 \leq i \leq n$ y $1 \leq j \leq i$.

El nombre de este problema tiene una explicación gráfica. Si en una cuadrícula rectangular representamos filas y columnas en direcciones ortogonales y señalamos las casillas que satisfagan la condición anterior, la región que se cubre tendrá forma triangular[6].

[4] Aunque algunos procesos de los algoritmos implementados no se pueden paralelizar.
[5] Creamos este término a falta de un nombre dado por la literatura.
[6] El apéndice E muestra un ejemplo de una región como esta.

Resolvimos este problema hasta llegar a $n = 60$, lo cual sería un avance si resulta que la referencia consultada que más lejos llegó en esta empresa en esta empresa (Ref. [29]) es también la que más lejos ha llegado[7] ($n = 30$). Podemos encontrar estos resultados en las figuras del apéndice E.

Como curiosidad, podemos ver que si queremos la otra mitad del triángulo, basta con reflejar los resultados de las gráficas sobre la diagonal, la cual contiene los resultados del problema anterior. Además, llamó nuestra atención que en estas figuras hay filas enteras totalmente nulas. Lastimosamente, no pudimos encontrar un patrón para predecir cuales son o si, extendiendo el triángulo a todo $\mathbb{Z}^+$, estas alguna vez acaban.

Los casos más exigentes de este problema son los correspondientes a $C_{60,60}$ y $T_{60,60}$, de los cuales ya conocemos sus tiempos de reducción. Así, podemos decir nuevamente que no nos vimos limitados por la complejidad computacional del problema y que deberíamos llegar más lejos con más tiempo para obtener resultados.

4.2. Problemas de restricción

El tema de las variantes (sección 3.1) tiene la característica de que amplía el universo de los rompecabezas luminosos. No obstante, también hay un tema que tiene la característica opuesta: reducirlo. De forma análoga al tema de las variantes, podemos preguntarnos cómo se puede hacer más específico un juego de Lights Out. Una respuesta sencilla es imponerle restricciones. De esta forma, motivamos la siguiente definición:

Definición 4.2.1. *Decimos que un* ***problema de restricción***[8] *es un caso particular de un juego tipo Lights Out donde le imponemos una o más restricciones.* △

Hay varias formas en que podemos imponerle restricciones a los rompecabezas, estas no son necesariamente excluyentes ni exhaustivas:

- exigir condiciones sobre la solución,
- exigir condiciones sobre el método de solución, o
- exigir condiciones sobre la configuración inicial.

En esta sección mostraremos cuatro de los muchos problemas de restricción que hay, aunque solo hablaremos de uno de ellos en detalle:

4.2.1. Problema del tablero totalmente encendido

El problema del ***tablero totalmente encendido***, también conocido como el problema del *jardín del eden* exige la condición de que el juego inicie con todas las luces encendidas. Si lo pensamos desde la perspectiva del modelo algebráico, este problema es equivalente a resolver la ecuación

$$Ax = 1,$$

[7] Adicionalmente, la Ref. [21] llegó hasta $n = 25$.

[8] Creamos este término a falta de un nombre dado por la literatura.

donde $1 = (1, \ldots, 1)^T$ es el vector de unos correspondiente al tamaño del tablero [4]. Algo curioso de este problema es que induce un resultado interesante que expondremos en un teorema. No obstante, para demostrarlo necesitamos primero demostrar un lema.

Lema 4.2.1. *En un juego tipo Lights Out con matriz de adyacencia simétrica que tenga un número impar de botones siempre hay algún botón que afecta un número par de los otros botones.*

Este teorema puede enunciarse en una forma que es mucho más fácil de entender: *en una reunión donde hay un número impar de personas donde todos saludan a quienes los saludan, siempre hay alguien que saluda un número par de veces.*

Demostración. Supongamos que hay n personas en la reunión, con n impar. Sea m_i el número de veces que la persona i saluda. El número total de saludos deberá ser

$$\frac{m_1 + m_2 + \cdots + m_n}{2},$$

puesto que cada saludo involucra a dos personas. Este número solo puede ser entero si la suma es par, lo que solo puede significar que no todos los m_i pueden ser impares. Por lo tanto, al menos una persona saluda un número par de veces [21]. □

Teorema 4.2.1. *Cualquier juego tipo Lights Out de dos estados con matriz de adyacencia reflexiva y simétrica permite resolver el problema del tablero totalmente encendido.*

Demostración. Si hubiera tableros donde este problema es irresoluble, tendría que haber uno con el número más pequeño de luces, digamos que sean n. En otras palabras, cualquier juego con $n - 1$ botones sería ganable, pero tenemos un juego particular con n botones que no lo es. Tenemos permiso para afirmar esto porque, en el peor de los casos, el juego con 1 botón siempre es ganable.

Para cualquier botón i, los otros $n - 1$ botones forman un juego más pequeño que es, por tanto, resoluble. Alternar todas esas $n-1$ luces no alternará la i-ésima, pues hemos supuesto que este juego era irresoluble. Por lo tanto, podemos encontrar n estrategias que alternan todo exceptuando una única luz.

Supongamos que ejecutamos estas n estrategias que alternan todo excepto una luz. Así, cada luz habrá alternado $n - 1$ veces. Si n fuera par, esto significaría que cada luz ha cambiado de estado, haciendo ganable nuestro juego. Por lo tanto, nuestro juego irresoluble debe tener un n impar.

Supongamos que hubiera una estrategia que alternara el estado de un número impar de luces (máximo $n - 2$). Ejecutémosla, y luego para cada luz que alternó, ejecutemos la estrategia que alterna todo excepto esa luz. Ahora todas las luces habrán alternado un número impar de veces, lo cual haría a nuestro juego ganable. Por lo tanto, todas las estrategias en un juego irresoluble deben alternar un número par de luces.

Hasta ahora hemos demostrado que el juego irresoluble más pequeño debe tener un número impar de luces y que cada estrategia alterna un número par de luces. No obstante, si el juego

es reflexivo y simétrico, por el lema anterior debe haber algún botón que alterne un número impar de luces (a sí mismo y a un número par de otras). Por lo tanto, no puede haber un juego tipo Lights Out reflexivo y simétrico irresoluble [21]. □

4.2.2. Otros problemas de restricción

El segundo problema que vamos a presentar es el de ***solo botones encendidos***. Consiste en ganar el juego solo presionando botones que estén encendidos. La mayoría de resultados sobre este teorema se escapan del alcance de esta disertación, pues involucran conceptos de autómatas celulares (sección 5.1), pero presentaremos uno que tiene una prueba muy sencilla de entender:

Teorema 4.2.2. *En un juego Lights Out, el problema del jardín del edén puede resolverse presionando solo botones encendidos.*

Demostración. Del teorema 4.2.1 nos es claro que el patrón que tiene todos los botones encendidos puede ser resuelto. Solo nos queda entonces mostrar que los movimientos pueden ser ejecutados en un orden tal que solo se presionen botones encendidos. Imaginemos la rejilla de botones marcada como si fuera un tablero de ajedréz, alternando botones blancos y negros.

Primero presionemos todos los botones en la solución que están marcados como negros y luego todos los botones de ella que están marcados blancos. Presionar un botón negro nunca cambiará otro negro (porque nunca son adyacentes), así que al presionar botones negros siempre estaremos presionando botones encendidos. Después de presionar los botones blancos restantes estarán todos apagados, así que antes de presionarlos debían estar encendidos. Por lo tanto, esta es una solución [21]. □

El tercer problema que vamos a presentar es el problema de la ***alternación***, consiste en apagar el tablero apagando botones encendidos y apagados de forma alternante. Es bastante más complejo que el de solo botones encendidos y exige obligatoriamente usar autómatas celulares lineales.

El cuarto y último problema que presentaremos se conoce como el problema del *tablero fácil*, consiste en encontrar los patrones donde las luces que se deben presionar para ganar el juego son las mismas que están encendidas [18]. Desde la perspectiva del modelo algebráico esto equivale a tener la ecuación

$$\begin{aligned} Ax &= x \\ (A - I)x &= 0. \end{aligned}$$

Estos patrones son los mismos autovectores de A. Es interesante que notemos que estos patrones x pueden pensarse como los patrones estacionarios de una variante con matriz de adyacencia $A - I$, que es no reflexiva.

Una variación de este problema es buscar los patrones donde las luces que se deben presionar para ganar el juego son las mismas que están apagadas. Desde la perspectiva del modelo esto

Botones encendidos en la última fila	Botones a presionar en la primera fila
00111	00010
01010	10010
01101	10000
10110	00001
10001	11000
11011	00100
11100	01000

Figura 4.1: Tabla de casos para el algoritmo caza de luces.

equivale a tener la ecuación:

$$Ax = 1 - x$$
$$(A + I)x = 1.$$

Por propiedades de $\mathbb{Z}_2$, tenemos que $A - I = A + I$, así que x es una secuencia de acciones que enciende todo el tablero en este juego irreflexivo. Los botones sobre una diagonal forman un patrón estacionario en este juego irreflexivo, por lo que el criterio de resolubilidad (2.4.5) muestra que este juego no se puede ganar cuando un tablero cuadrado de $n \times n$ tiene un n impar. Por el contrario, cuando n es par el problema si es resoluble [21].

4.3. Caza de luces

En esta sección discutiremos algunos métodos de solución de los rompecabezas Lights Out diferentes a los expresados en la sección de resolubilidad (sección 2.4):

Como vimos anteriormente, es posible encontrar una solución general del rompecabezas, es decir, un método que permita determinar el vector de incidencias x a partir de un patrón inicial resoluble y. Esto es posible si reducimos la matriz aumentada $[A \mid y]$.

No obstante, un método semejante solo es práctico de ejecutar en un computador. Si al lector le interesa conocer un algoritmo para resolver el Lights Out de tamaño 5×5 que una persona pueda ejecutar sin hacer complejas cuentas en matrices, le sugerimos aplicar el siguiente algoritmo iterativo sobre patrones iniciales ganables [23]:

1. Empezando desde la segunda fila (hasta la última), presione los botones que tienen su vecino ortogonal superior encendido. Esto generará un tablero que solo tiene botones encendidos en la última fila.
2. Para cada uno de los 7 casos que le queden en la 7^{ma} fila, presione los botones de la 1^{era} que se indican en la figura 4.1.
3. Repita el paso 1. Al final de este, el tablero estará resuelto.

Este algoritmo se llama ***caza de luces*** porque el proceso de reducir el rompecabezas a la última fila se asemeja a una cacería [23]. Mediante una síntesis podemos reemplazar su segundo paso de la siguiente manera. Las indicaciones presentadas no son excluyentes [17]:

- Si el primer botón de la última fila está encendido, presione el cuarto y el quinto de la primera.
- Si el segundo botón de la última fila está encendido, presione el segundo y el quinto de la primera.
- Si el tercer botón de la última fila está encendido, presione el cuarto de la primera.
- Los últimos dos botones de la última fila no exigen ejecutar ninguna acción.

Demostrar la validez de la *caza de luces* conlleva un considerable esfuerzo computacional que una vez hecho, afortunadamente no hay que repetir. Si deseáramos hacerlo tendríamos que realizar las siguientes tareas [17]:

- Mostrar que para cualquier patrón inicial hay una sucesión de movimientos que lo reduce a la primera fila.
- Suponiendo resolubilidad, mostrar que hay 7 patrones no triviales que tienen luces encendidas solo en la primera fila.
- Derivar las soluciones de estos 7 patrones del modelo matricial.

Ejecutar este algortmo casi nunca resulta en una solución óptima: lo más probable es que presionemos botones más de una vez. No obstante, es una solución que los humanos podemos aplicar de forma práctica. Notamos de forma adicional que pueden generarse algoritmos como este para tableros de dimensión arbitraria [17].

Ya que hemos discutido sobre algunas aplicaciones computacionales sobre el tema de los rompecabezas luminosos, pasemos a comentar sobre otras herramientas diferentes al álgebra lineal que se usan en su estudio.

Capítulo 5

Otras herramientas

5.1. Sondeo de herramientas

Cuando estudiamos rompecabezas luminosos usando técnicas de álgebra lineal, los resultados a los que podemos llegar están limitados por las limitaciones del álgebra lineal. Esto hará que solo nos sea posible llegar hasta un cierto tipo de conclusiones [24].

Por este motivo se justifica usar herramientas de otras ramas de las matemáticas para estudiar las propiedades de los rompecabezas sobre los que trabajemos. Esas herramientas nos permitirán alcanzar resultados que serían imposibles (o demasiado difíciles) por medio de solo el álgebra lineal. A continuación presentaremos una lista no exhaustiva de ellas:

- Polinomios de Fibonacci.
- Polinomios de Chebyshev.
- Polinomios característicos.
- Autómatas celulares (lineales).
- Teoría de grafos.

Cabe notar que la teoría de autómatas celulares permite describir de forma precisa la evolución del tablero a medida que se juega. Esto la hace muy útil para analizar problemas de restricción que involucran el método de solución a utilizar. Además de esto, permite expresar de forma sencilla juegos con movimientos con definiciones complejas [21].

5.2. Teoría de grafos

Aplicar técnicas de teoría de grafos para analizar rompecabezas luminosos tiene la gran ventaja de permitirnos llegar a resultados más generales que los obtenidos por medio del álgebra lineal. Los resultados obtenidos serán válidos para todos los rompecabezas que compartan las propiedades que tenga el tipo de grafo que estemos analizando [19].

Esa capacidad de lograr resultados más generales es una ganancia del análisis a través de la teoría de grafos. No obstante, como es común en matemáticas: cuando ganamos algo, también

perdemos algo [14]. El intercambio que hacemos al estudiar rompecabezas luminosos usando grafos es que solo nos permite estudiar rompecabezas que tengan dos estados posibles [24].

Gracias a esta teoría podemos generar un modelo matemático para los Lights Out que, aunque tenga un fundamento matemáticamente más complejo, induce una ecuación algebráica idéntica al modelo algebraico. Por este motivo, se considera usualmente que el uso de ambas herramientas es de igual importancia para estudiar estos rompecabezas [24].

Los modelos de teoría de grafos a grandes rasgos se construyen siguiendo el procedimiento mostrado a continuación [19]:

1. Creamos un nodo por cada botón del tablero. Se acostumbra usar la letra v con subíndices para representar los nodos.
2. Si la acción de presionar un botón A afecta a B, lo representamos por medio de una flecha de A hacia B.
3. En caso de la acción de un botón lo afecte, lo representamos por un superíndice 1 sobre el nodo. Hacemos esto para no congestionar su lectura.
4. En caso de que las acciones de todos los botones los afecten a sí mismos, omitiremos todos los superíndices.
5. En caso de que dos nodos se afecten a la vez, los uniremos por una linea en vez de una flecha.

Ejemplo 5.2.1. *Podemos observar cómo quedaría el grafo del Lights Out clásico en la figura 5.1.* ◊

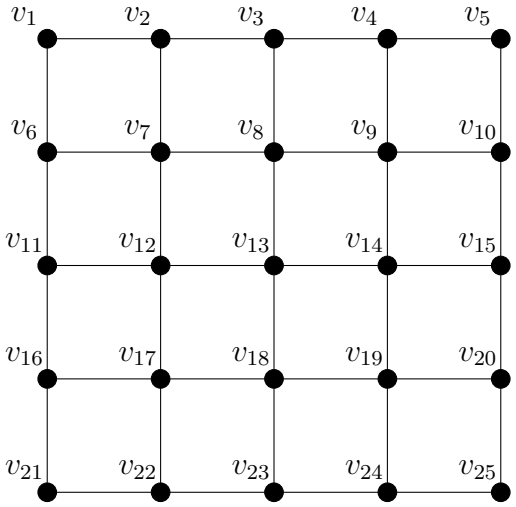


Figura 5.1: Grafo asociado al Lights Out clásico.

El término *matriz de adyacencia* realmente proviene de analizar los rompecabezas luminosos a través de la teoría de grafos. Esta teoría las genera al considerar los vínculos entre nodos como una relación binaria y, como es de esperarse, siempre coinciden con las del modelo

algebraico [24].

Uno de los resultados más notables que podemos obtener a través de la teoría de grafos es:

"Las matrices de adyacencia de los Lights Out siempre se pueden construir como matrices por bloques conformadas por matrices identitarias, nulas y tridiagonales".

Este resultado es sorprendente en la medida en que no tiene en cuenta la dimensión del tablero que tratemos [19].

Establecer la estructura de las matrices de adyacencia sin tratar los rompecabezas de los que provienen es algo que no siempre se puede hacer. Por ejemplo, para el rompecabezas de dimensión 2×3 podemos ver que naturalmente su matriz de adyacencia es una matriz de tamaño 6×6. Lo que no es tan obvio es saber que esta es una matriz de 4 bloques de 3×3.

No obstante, esto sí se puede hacer para tableros cuadrados. Así, la matriz de adyacencia A del Lights Out de orden n es la matriz por bloques

$$A_{n^2 \times n^2} = \begin{bmatrix} M & I & 0 & \dots & 0 & 0 & 0 \\ I & M & I & \dots & 0 & 0 & 0 \\ 0 & I & M & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & M & I & 0 \\ 0 & 0 & 0 & \dots & I & M & I \\ 0 & 0 & 0 & \dots & 0 & I & M \end{bmatrix}$$

donde

$$M_{n \times n} = \begin{bmatrix} 1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & 1 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & 1 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 1 \end{bmatrix},$$

I es la matriz identidad de tamaño $n \times n$ y 0 es la matriz nula del mismo tamaño [19].

Capítulo 6

Discusión

6.1. Conclusiones

En esta sección concluiremos sobre los logros más importantes de este proyecto:

1. Para este proyecto desarrollamos dos aplicaciones que simulan el problema del tablero totalmente encendido para Lights Out cuadrados y sus variantes toroidales de dimensión arbitraria. En su lógica de programación se constata que la Ec. 2.1 es suficiente para simular los Lights Out (o sus variantes toroidales).

   Vale la pena notar que el código de estas aplicaciones requiere calcular los efectos de las acciones cada vez que se van a utilizar. Esta es una solución aceptable porque los tamaños de los rompecabezas con los que jugamos las personas son pequeños como para no generar un coste computacional importante y porque de esta forma nos evitamos tener que usar métodos de persistencia de memoria.

2. En este proyecto logramos resolver el problema del triángulo hasta un orden $n = 60$. Si los avances presentados por las referencias consultadas son realmente los más actuales, eso significaría que hemos roto el límite obteniendo 30 órdenes nuevos, cosa que constituye un avance significativo.

   Esta resolución no pudo llegar más lejos por motivos del tiempo de entrega del proyecto. No obstante, comentamos que tiempos de cálculo aún no se habían hecho imprácticos.

   Los resultados obtenidos fueron alcanzados con un algoritmo no optimizado para el tipo de matriz que resulta del problema (una práctica que es común para mejorar la eficiencia de los cálculos). En caso de usarse un algoritmo optimizado, de seguir calculando ordenes más grandes podríamos ganar una ventaja cuando el tiempo de computación empiece a hacerse impráctico con el algoritmo que tenemos.

3. A pesar de que logramos reproducir y superar los resultados mostrados por el artículo de base, estos aún distan mucho de romper el límite de $n = 1000$ que establece la referencia consultada que más lejos ha llegado. Es posible que en caso de querer romper ese límite necesitemos usar técnicas de multiprocesamiento u otras herramientas de

computación de alto rendimiento por el coste computacional que acarrearía procesar una matriz de ese tamaño.

4. El orden del costo computacional de crear ($O(n^2)$) y de reducir ($O(n^3)$) una matriz es un resultado teórico conocido contra el que nos comparamos de forma experimental por medio de los tiempos empleados por las matrices de los rompecabezas cuadrados que procesamos.

6.2. Perspectivas

En esta sección propondremos algunas formas en las que podríamos encaminar trabajos posteriores sobre el tema de los rompecabezas luminosos:

1. Profundizar en los alcances y limitaciones del método de caza de luces, su teoría y resultados para los Lights Out y sus variantes de distintos tamaños.

2. Profundizar en los alcances y limitaciones de los autómatas celulares, su teoría y resultados para estudiar rompecabezas luminosos y sus problemas de restricción.

3. Profundizar en los resultados teóricos que existen para distintos problemas de restricción.

4. Tratar de romper el límite de orden $n = 1000$ impuesto por la referencia consultada que más lejos ha llegado en el cálculo de la nulidad para Lights Out cuadrados.

5. Seguir obteniendo resultados para órdenes más grandes en el problema del triángulo.

Bibliografía

[1] Kolman, B., 1977: Elementary Linear Algebra. Macmillan Publishing Co. Inc.

[2] Lovasz L., 1979: Combinatorial Problems and Exercises, North-Holland Publishing. https://doi.org/10.1002/zamm.19800600413

[3] Pelletier, D., 1987: Merlin's Magic Square. *The American Mathematical Monthly*, **94 Issue 2**, 143-150, https://doi.org/10.1080/00029890.1987.12000606.

[4] Sutner, K., 1989: Linear cellular automata and the garden-of-eden. *The Mathematical Intelligencer*, **11**, 49–53, https://doi.org/10.1007/BF03023823.

[5] Sutner, K., 1990: The σ-Game and Cellular Automata, *The American Mathematical Monthly*, **97 Issue 1**, 24-34, https://doi.org/10.1080/00029890.1990.11995540.

[6] Dummit, D., Foote, R.: 1991. Abstract algebra. Englewood Cliffs, N.J.: Prentice Hall.

[7] Goldwasser, J., Klostermeyer, W., Trapp, G., Zhang, C., 1995: Setting Switches on a Grid, Technical Report TR-95-20, Dept. of Statistics and Computer Science, West Virginia University. https://www.researchgate.net/publication/2384576_Setting_Switches_in_a_Grid

[8] Barua, R., Ramakrishnan, S., 1996: σ-game, σ^+game, and two-dimensional cellular automata, *Theoret. Comput. Sci.*, **154 no. 2**, 349–366, https://core.ac.uk/reader/81979062.

[9] Goldwasser, J., Klostermeyer, W., 1997: Maximization versions of "Lights Out" games in grids and graphs, *Congr. Numer.*, **126**, 99–111, https://www.unf.edu/~wkloster/fibonacci/congnum.pdf.

[10] Goldwasser, J., Klostermeyer, W., Trapp, G., 1997: Characterizing switch-setting problems, *Linear and Multilinear Algebra*, **43 Issue 1-3**, 121-135, https://doi.org/10.1080/03081089708818520

[11] Anderson, M., Feil, T., 1998: Turning Lights Out with Linear Algebra, *Mathematics Magazine*, **71 Issue 4**, 300-303. https://doi.org/10.1080/0025570X.1998.11996658.

[12] Chapra, S., Canale, R., 2006: Numerical methods for engineers. Boston: McGraw-Hill Higher Education.

[13] Hogben, L., 2006: Handbook of Linear Algebra (Discrete Mathematics and Its Applications), Boca Raton: Chapman & Hall/CRC.

[14] Zalamea, F.: 2007, Fundamentos de Matemáticas, Universidad Nacional de Colombia.

[15] Joyner, D., 2008: Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys. The Johns Hopkins University Press.

[16] Edwards. S. et al., 2010: Lights Out on finite graphs, *Involve*, **3 No. 1**, 17-32. https://doi.org/10.2140/involve.2010.3.17.

[17] Khoury, J., 2010: Blog sobre aplicaciones del álgebra lineal, recuperado el 8 de enero del 2020. http://aix1.uottawa.ca/~jkhoury/gamesolution.htm.

[18] Torrence, B., 2011: The Easiest Lights Out Games, *The College Mathematics Journal*, **42 Issue 5**, 361-372, https://doi.org/10.4169/college.math.j.42.5.361.

[19] Meyer, R., 2013: The game of Lights out. EWU Master's Thesis Collection. 167. https://dc.ewu.edu/theses/167.

[20] Fleischer R., Yu J., 2013: A Survey of the Game "Lights Out!". In: Brodnik A., López-Ortiz A., Raman V., Viola A. (eds) Space-Efficient Data Structures, Streams, and Algorithms. Lecture Notes in Computer Science, vol 8066. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40273-9_13.

[21] Scherphuis, J., 2014: Blog sobre teoría de rompecabezas, recuperado el 5 de enero del 2020. https://www.jaapsch.net/puzzles/index.htm.

[22] Kreh, M., 2017: "Lights Out" and Variants, *The American Mathematical Monthly*, **124 Issue 10**, 937-950. https://doi.org/10.4169/amer.math.monthly.124.10.937.

[23] Leach, C., 2017: Chasing the Lights in Lights Out, *Mathematics Magazine*, **90 Issue 2**, 126-133. https://doi.org/10.4169/math.mag.90.2.126.

[24] Salamanca, I., 2017: Lights Out. Proyecto Fin de Carrera / Trabajo Fin de Grado, E.T.S. de Ingenieros Informáticos (UPM), Madrid, España. http://oa.upm.es/47118/.

[25] Barile, M.: Lights Out Puzzle. From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. https://mathworld.wolfram.com/LightsOutPuzzle.html

[26] Losada, R.: Cuaderno Geogebra que redirige a las publicaciones del autor, recuperado el 6 de enero del 2020. https://www.geogebra.org/m/JexnDJpt

[27] Entrada de la enciclopedia OEIS sobre la nulidad de C_n, recuperada el 15 de junio del 2020. https://oeis.org/A159257

[28] Entrada de la enciclopedia OEIS sobre la nulidad de T_n, recuperada el 15 de junio del 2020. http://oeis.org/A165738

[29] Brower, A.: Página web asociada a la de la OEIS con los resultados más extensos sobre el problema del triángulo, recuperada el 17 de junio del 2020. https://www.win.tue.nl/~aeb/ca/madness/madrect.html

[30] Li, P.: Diapositiva con información de las variantes de Lights Out, recuperada el 20 de junio del 2020. https://www.slideshare.net/PengfeiLi1/lop-38272545

Apéndice A

Código usado para la obtención de resultados

Nota: Toda vez que en este código se enuncie la referencia "tesis de Salamanca" se está refiriendo a la Ref. [24].

Importo las librerías que voy a usar.

```
import numpy as np
from time import time

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sb
```

Defino funciones para generar las matrices de adyacencia del *Lights Out clásico*, estas son compatibles con tableros rectangulares de tamaño arbitrario:

```
def accion_clasico(y_toggle, x_toggle, height, width):
    # Esta funcion genera una matriz que al aplanarse (funcion flatten)
    # coincide con la accion del boton correspondiente al indice
    # determinado por sus coordenadas

    # Comienzo con una matriz nula de tamano dado
    toggle = np.zeros((height, width))
    # El boton que se presiona sufre alternacion
    toggle[y_toggle][x_toggle] = 1

    # Valido condiciones de que los vecinos no diagonales esten
    # contenidos en los limites del tablero, si se cumplen
    # es porque el boton sufre alternacion
    if (y_toggle - 1 >= 0):       # Boton superior
        toggle[y_toggle - 1][x_toggle] = 1
    if (y_toggle + 1 <= height - 1): # Boton inferior
```

```
        toggle[y_toggle + 1][x_toggle] = 1
    if (x_toggle - 1 >= 0):       # Boton izquierdo
        toggle[y_toggle][x_toggle - 1] = 1
    if (x_toggle + 1 <= width - 1): # Boton derecho
        toggle[y_toggle][x_toggle + 1] = 1

    return toggle # Con la matriz construida hago retorno

def construccion_clasico(height, width):
    # Esta funcion construye la matriz de adyacencia a partir de
    # las acciones de los botones del tablero

    # Inicio creando una matriz compatible sin elementos, esto
    # es un truco para que al agregar filas no haya problemas
    adjacency = np.zeros((0, height*width))

    # El proceso se hace una vez por cada boton del tablero
    for i in range(height):
        for j in range(width):
            # Construyo la accion como matriz en el orden de indexacion
            toggle = accion_clasico(i, j, height, width)
            # La aplano en forma de fila como en el modelo algebraico
            row = toggle.flatten()
            # Anado la fila construida a la matriz de adyacencia
            adjacency = np.vstack((adjacency,row))

    # Retorno la matriz como lista porque el algoritmo que haya la
    # matriz rref fue implementado para trabajar con este tipo de dato
    return adjacency.tolist()
```

Defino funciones para generar las matrices de adyacencia de la *variante toroidal de Lights Out*, estas son compatibles con tableros rectangulares de tamaño arbitrario:

```
def accion_toroidal(y_toggle, x_toggle, height, width):
    # Esta funcion hace el proceso de la funcion 'accion_normal'
    # pero para la variante toroidal, por lo que su logica es
    # casi identica
    toggle = np.zeros((height, width))

    toggle[y_toggle][x_toggle] = 1

    # Solo cambia en que cuando un vecino no diagonal escapa de
    # los limites del tablero, se marca la alternancia en el
    # borde opuesto del tablero
    if (y_toggle - 1 >= 0):
        toggle[y_toggle - 1][x_toggle] = 1
    else:
        toggle[height - 1][x_toggle] = 1
```

```
    if (y_toggle + 1 <= height - 1):
        toggle[y_toggle + 1][x_toggle] = 1
    else:
        toggle[0][x_toggle] = 1
    if (x_toggle - 1 >= 0):
        toggle[y_toggle][x_toggle - 1] = 1
    else:
        toggle[y_toggle][width - 1] = 1
    if (x_toggle + 1 <= width - 1):
        toggle[y_toggle][x_toggle + 1] = 1
    else:
        toggle[y_toggle][0] = 1

    return toggle

def construccion_toroidal(height, width):
    # Esta funcion hace el proceso de la funcion 'construccion_normal'
    # pero para la variante toroidal, por lo que su logica es
    # casi identica
    adjacency = np.zeros((0, height*width))

    for i in range(height):
        for j in range(width):
            # Solo cambia al depender de las acciones toroidales
            toggle = accion_toroidal(i, j, height, width)
            row = toggle.flatten()
            adjacency = np.vstack((adjacency,row))

    return adjacency.tolist()
```

Se definen dos funciones: una que calcula el opuesto aditivo de un número e en el cuerpo $\mathbb{Z}/m\mathbb{Z}$ con m dado y otra que, de forma similar, calcula su opuesto multiplicativo:

```
def elemento_opuesto(e, base):
    # Esta funcion proviene de la tesis de Salamanca sin cambios;
    # busca entre los elementos del cuerpo el primer elemento de
    # cero al sumarse con 'e'. Como Z/mZ es un cuerpo, la funcion
    # tiene un retorno unico garantizado
    for k in range(base):
        if (k + e) % base == 0:
            return k

def elemento_inverso(e, base):
    # Esta funcion proviene de la tesis de Salamanca sin cambios.
    # La logica de esta funcion es identica a la de la funcion
    # 'elemento_opuesto'
    for k in range(base):
        # Solo cambia al depender de la estructura multiplicativa
```

```
        if (k * e) % base == 1:
            return k
```

Se define una función que halla la matriz *rref* de una matriz dada que contenga elementos del cuerpo $\mathbb{Z}/m\mathbb{Z}$ por medio del método de Gauss-Jordan. La función funciona apropiadamente siempre que m sea primo[1]:

```
def rref_gauss_jordan(L, base):
    # Esta funcion proviene de la tesis de Salamanca, halla la
    # matriz rref por medio de reduccion de Gauss-Jordan.
    # Cambios: la logica de reduccion por filas se pasa a una
    # funcion de nombre 'reduccion_filas' para mejorar la legibilidad.

    # Se recorren todas las filas en orden descendente aplicando
    # operaciones de reduccion por filas que dependen del valor
    # particular de los terminos de la diagonal principal
    for i in range(len(L)):

        # En caso de que el termino que debe ser pivote valga 0.
        # Los juegos a trabajar son reflexivos, este caso no les aplica
        if L[i][i] == 0:
            # Se busca el primer candidato a pivote posible en las
            # filas inferiores. Luego se permutan las filas involucradas
            for k in range (i + 1, len(L)):
                if L[k][i] == 1:
                    filaAux = L[i]
                    L[i] = L[k]
                    L[k] = filaAux
                    break
            # Se anulan los otros terminos de la columna sumando filas
            L = reduccion_filas(L, base, i)

        # En caso de que el termino que debe ser pivote ya valga 1
        elif L[i][i] == 1:
            # Se anulan los otros terminos de la columna sumando filas
            L = reduccion_filas(L, base, i)


        # Este caso solo aplica cuando m >= 2. No se usa en este caso
        # donde m = 2
        else:
            # Se escala la fila para que el termino que debe ser
            # ser pivote valga 1
            inverso = elemento_inverso(L[i][i], base)
            for j in range(len(L[i])):
                L[i][j] = (L[i][j]*inverso) % base
```

[1]Si m no es primo, no hay garantía de que $\mathbb{Z}_m$ sea un cuerpo. Eso significa que puede haber elementos con múltiples opuestos (como cuando $m = 4$) y, por tanto, será imposible hacer reducción de Gauss-Jordan.

```
            # Se anulan los otros terminos de la columna sumando filas
            L = reduccion_filas(L, base, i)

    return L

def reduccion_filas(L, base, i):
    # Esta funcion hace operaciones de reduccion por filas sobre las
    # filas distintas a la i-esima, anulando todos los terminos de
    # la columna distintos al pivote
    for j in range(len(L)):
        if L[j][i] != 0 and j != i:
            piv = elemento_opuesto(L[j][i], base)
            for k in range(len(L[0])):
                L[j][k] = (L[j][k] + piv * L[i][k]) % base
    return L
```

Se define una función que halla la nulidad de una matriz dada:

```
def nulidad(L):
    # Esta funcion proviene de la tesis de Salamanca sin cambios;
    # calcula la nulidad de la matriz como el numero de filas
    # nulas, para esto recorre las filas e incrementa un contador
    # inicialmente nulo con cada nueva fila nula encontrada
    cont = 0
    for i in range(len(L)):
        todo_ceros = True
        for j in range(len(L[0])):
            if L[i][j] != 0:
                todo_ceros = False
        if todo_ceros:
            cont += 1
    return cont
```

Defino una función para reproducir los resultados del artículo base:

```
def res_tab_cuad(a, b, txt_sucesion, txt_tiempo, func_cons):
    # Esta funcion obtiene los valores de nul(A_n) entre dos limites
    # (incluyendolos), ademas de los tiempos de generacion de las
    # matrices de adyacencia y las rref. Consigna estos datos en dos
    # archivos de texto (.txt) distintos.
    # La apertura y cierre oportunos de los .txt mejora los tiempos
    # de ejecucion

    archivo_suc = open(txt_sucesion + ".txt", "a")
    archivo_tiempo = open(txt_tiempo + ".txt", "a")
```

```
# Anotacion de limites de las filas
archivo_suc.write("lim_inf = " + str(a) +" \n")
archivo_suc.write("lim_sup = " + str(b) +" \n \n")

archivo_tiempo.write("lim_inf = " + str(a) +" \n")
archivo_tiempo.write("lim_sup = " + str(b) +" \n \n")

# Anotacion de rotulos de las columnas
archivo_suc.write("i nul \n")
archivo_tiempo.write("i t_matriz t_rref \n")

archivo_suc.close()
archivo_tiempo.close()

for i in range(a, b + 1):
    # Matrices de adyacencia y conteo de tiempo
    tiempo_inicio = time()
    adyacencia = func_cons(i, i)
    tiempo_matriz = time() - tiempo_inicio

    # Matrices rref y conteo de tiempo
    tiempo_inicio = time()
    rref = rref_gauss_jordan(adyacencia, 2)
    tiempo_rref = time() - tiempo_inicio

    nul = nulidad(rref) # Calculo de nul(A_n)

    # Escritura de resultados
    archivo_suc = open(txt_sucesion + ".txt", "a")
    archivo_tiempo = open(txt_tiempo + ".txt", "a")

    archivo_suc.write(str(i) + ", " + str(nul) + "\n")
    archivo_tiempo.write(str(i) + ", " + str(tiempo_matriz) + ", " +
        str(tiempo_rref) + "\n")

    archivo_suc.close()
    archivo_tiempo.close()

# Espacio adicional entre procesos
archivo_suc = open(txt_sucesion + ".txt", "a")
archivo_tiempo = open(txt_tiempo + ".txt", "a")

archivo_suc.write("\n")
archivo_tiempo.write("\n")

archivo_suc.close()
archivo_tiempo.close()
```

Defino una función para obtener los resultados necesarios del problema del triángulo:

```
def res_tab_arb(a, b, txt_mapa, func_cons):
    # Esta funcion obtiene los valores de la nulidad de la matriz de
    # adyacencia entre entre dos limites a <= i <= b (incluyendolos),
    # para todos los tableros rectangulares entre ellos, de esta forma,
    # la otra dimension j del tablero queda libre entre 1 <= j <= i.
    # Consigna los datos obtenidos en un archivo de texto.
    # La apertura y cierre oportunos de los .txt mejora los tiempos
    # de ejecucion

    archivo_mapa = open(txt_mapa + ".txt", "a")

    # Anotacion de limites de las filas
    archivo_mapa.write("lim_inf = " + str(a) +" \n")
    archivo_mapa.write("lim_sup = " + str(b) +" \n \n")
    # Anotacion de rotulos de las columnas
    archivo_mapa.write("i j nul \n")

    archivo_mapa.close()

    for i in range(a, b + 1):
        for j in range(1, i + 1):
            # Calculo de la nulidad entre los limites establecidos
            adyacencia = func_cons(i, j)
            rref = rref_gauss_jordan(adyacencia, 2)
            nul = nulidad(rref)

            # Escritura de resultados
            archivo_mapa = open(txt_mapa + ".txt", "a")
            archivo_mapa.write(str(i) + ", " + str(j) + ", " + str(nul) + "\n")
            archivo_mapa.close()

    # Espacio adicional entre procesos
    archivo_mapa = open(txt_mapa + ".txt", "a")
    archivo_mapa.write("\n")
    archivo_mapa.close()
```

A través de las siguientes lineas reproducimos los resultados computacionales del artículo base:

```
res_tab_cuad(1, 60, "sucesion_clasico", "tiempo_clasico", construccion_clasico)
res_tab_cuad(1, 60, "sucesion_toroidal", "tiempo_toroidal",
    construccion_toroidal)
```

A través de las siguientes lineas calculamos los resultados obtenidos sobre el problema del triángulo:

```
res_tab_arb(1, 60, "mapa_clasico", construccion_clasico)
res_tab_arb(1, 60, "mapa_toroidal", construccion_toroidal)
```

Defino dos funciones: una para obtener graficas sobre los contenidos del artículo base y otra para obtener gráficas del problema del triángulo.

Para ejecutar estas funciones se requiere copiar los archivos de resultados que se van a utilizar en una carpeta interna de nombre **resultados** y que a estos se les quiten los encabezados. Esto se hace para tener copias intactas con los resultados con los encabezados. Los encabezados son información valiosa sobre el progreso realizado.

```
def grafica_articulo(archivo, nombre_matriz):
    # Esta funcion grafica los resultados de la reproduccion de resultados
    # del articulo base para los Lights Out y sus variantes toroidales

    # Lectura del archivo de resultados
    df = pd.read_table('resultados/' + archivo + '.txt', sep = ', ', engine =
        'python', names = ('Orden', 'Valores'))

    # Definicion del tamano de la figura
    sb.set(rc = {'figure.figsize': (20, 15)})

    # Construccion y generacion de la imagen
    plt.xlabel('$n$')
    plt.ylabel('$nul(' + nombre_matriz + ')$')
    plt.title('Orden $n$ vs. $nul(' + nombre_matriz + ')$')
    plt.plot(df.Orden, df.Valores)

def grafica_triangulo(archivo):
    # Esta funcion grafica los resultados del problema del
    # triangulo para los Lights Out y sus variantes toroidales

    # Lectura del archivo de resultados
    df = pd.read_table('resultados/' + archivo + '.txt', sep = ', ', engine =
        'python', names = ('Filas', 'Columnas', 'Valores'))

    # Paso de columnas a un DataFrame
    df.drop_duplicates(['Filas', 'Columnas'], inplace = True)
    pivot = df.pivot(index = 'Filas', columns = 'Columnas', values = 'Valores')

    # Definicion del tamano de la figura
    sb.set(rc = {'figure.figsize': (20, 15)})
    # Construccion de la imagen
    sb.heatmap(pivot, annot = True)
```

A través de las siguientes lineas graficamos la reproducción de resultados del artículo base:

```
grafica_articulo('sucesion_clasico', 'C_n')
grafica_articulo('sucesion_toroidal', 'T_n')
```

A través de las siguientes lineas graficamos los resultados obtenidos para los problemas del triángulo:

```
grafica_triangulo('mapa_clasico')
grafica_triangulo('mapa_toroidal')
```

Apéndice B

Reproducción del artículo base

n	$nul(C_n)$
1	0
2	0
3	0
4	4
5	2
6	0
7	0
8	0
9	8
10	0
11	6
12	0

n	$nul(C_n)$
13	0
14	4
15	0
16	8
17	2
18	0
19	16
20	0
21	0
22	0
23	14
24	4

n	$nul(C_n)$
25	0
26	0
27	0
28	0
29	10
30	20
31	0
32	20
33	16
34	4
35	6
36	0

n	$nul(C_n)$
37	0
38	0
39	32
40	0
41	2
42	0
43	0
44	4
45	0
46	0
47	30
48	0

n	$nul(C_n)$
49	8
50	8
51	0
52	0
53	2
54	4
55	0
56	0
57	0
58	0
59	22
60	0

Figura B.1: Reproducción de resultados para los Lights Out.

n	$nul(T_n)$
1	0
2	0
3	4
4	0
5	8
6	8
7	0
8	0
9	4
10	16
11	0
12	16

n	$nul(T_n)$
13	0
14	0
15	12
16	0
17	16
18	8
19	0
20	32
21	4
22	0
23	0
24	32

n	$nul(T_n)$
25	8
26	0
27	4
28	0
29	0
30	24
31	40
32	0
33	44
34	32
35	8
36	16

n	$nul(T_n)$
37	0
38	0
39	4
40	64
41	0
42	8
43	0
44	0
45	12
46	0
47	0
48	64

n	$nul(T_n)$
49	0
50	16
51	20
52	0
53	0
54	8
55	8
56	0
57	4
58	0
59	0
60	48

Figura B.2: Reproducción de resultados para los Lights Out toroidales.

Apéndice C

Graficas de los resultados reproducidos

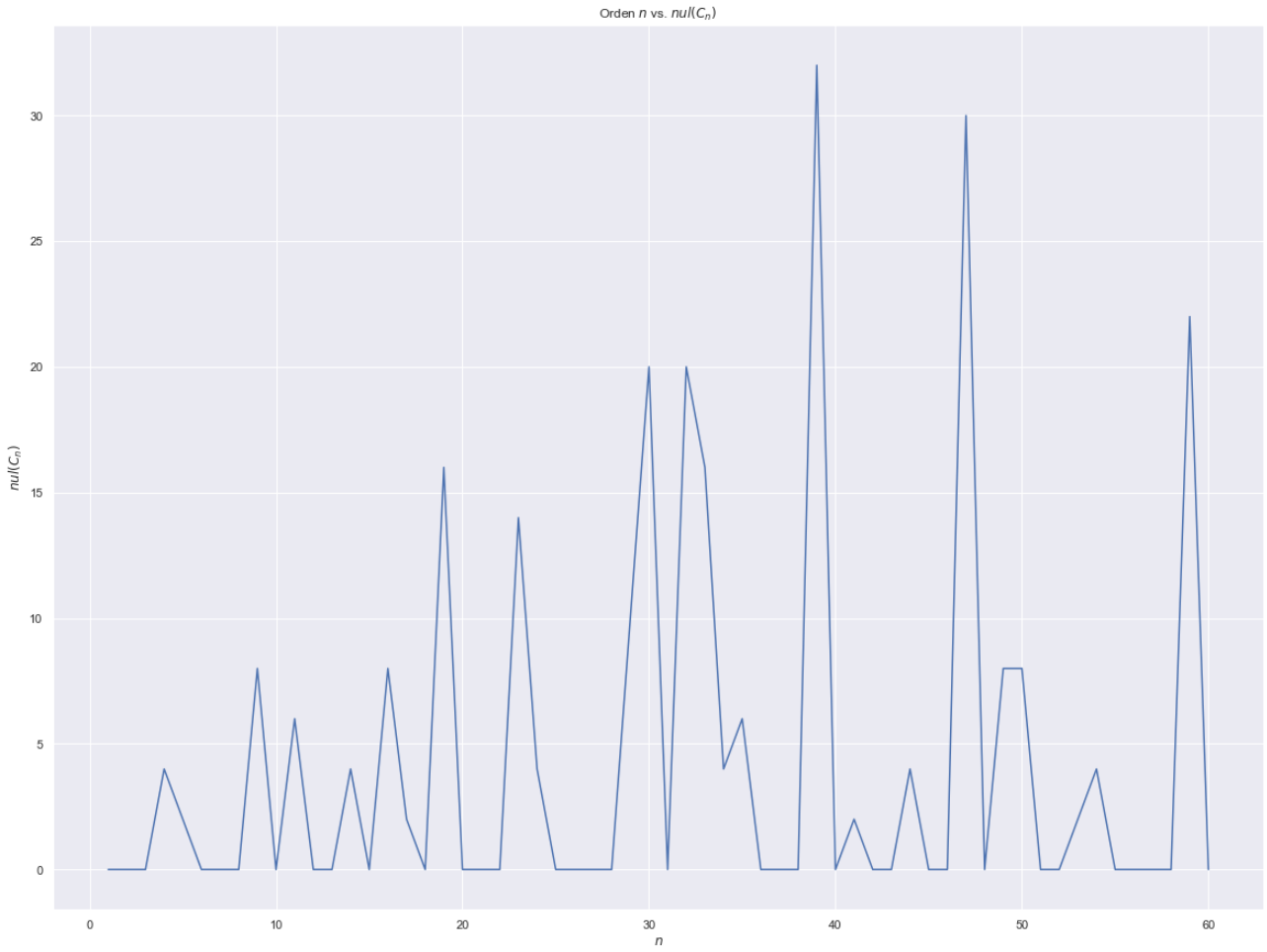


Figura C.1: Gráfico de n vs. $nul(C_n)$

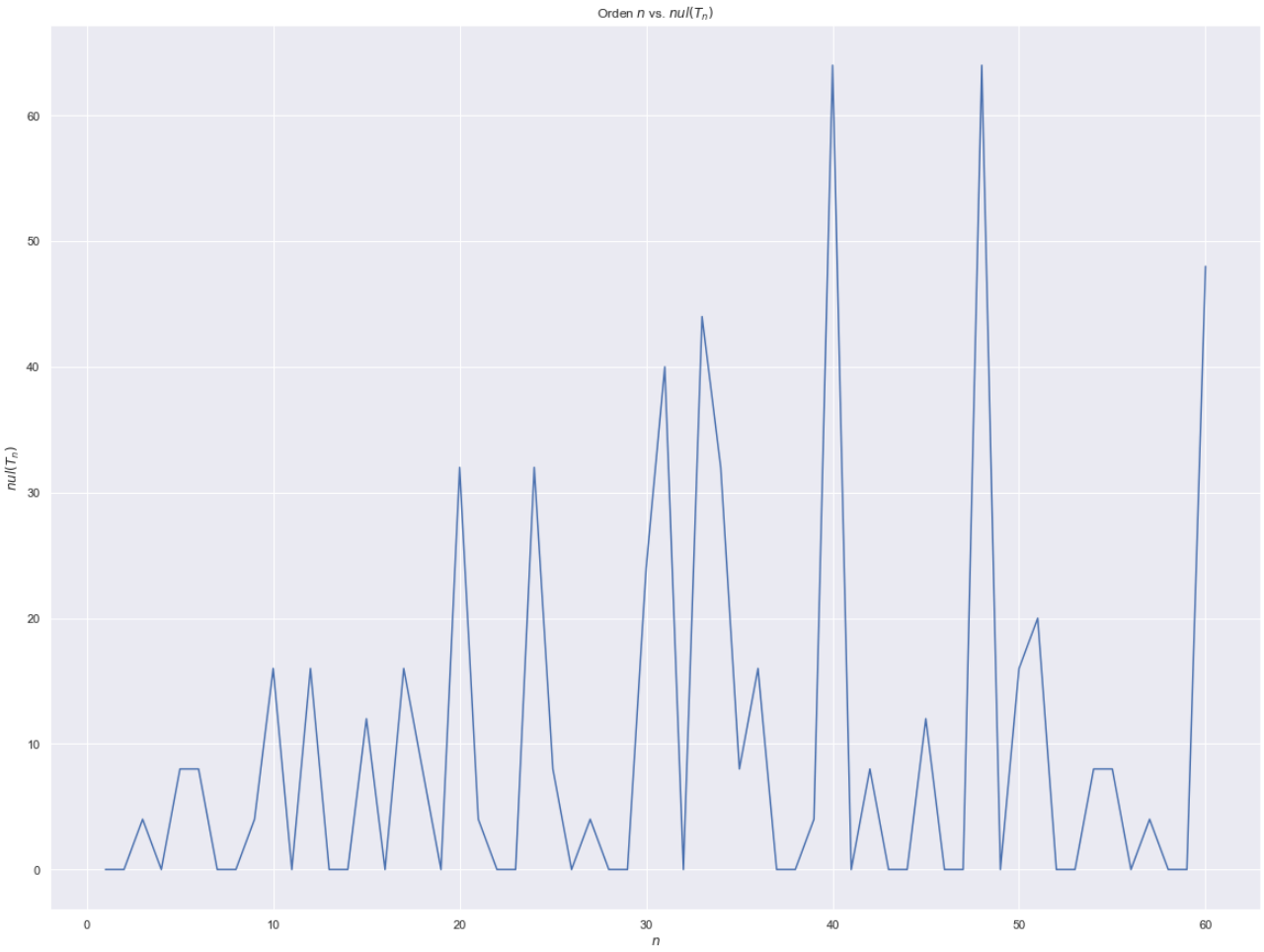


Figura C.2: Gráfico de n vs. $nul(T_n)$

Apéndice D

Tiempos de computación

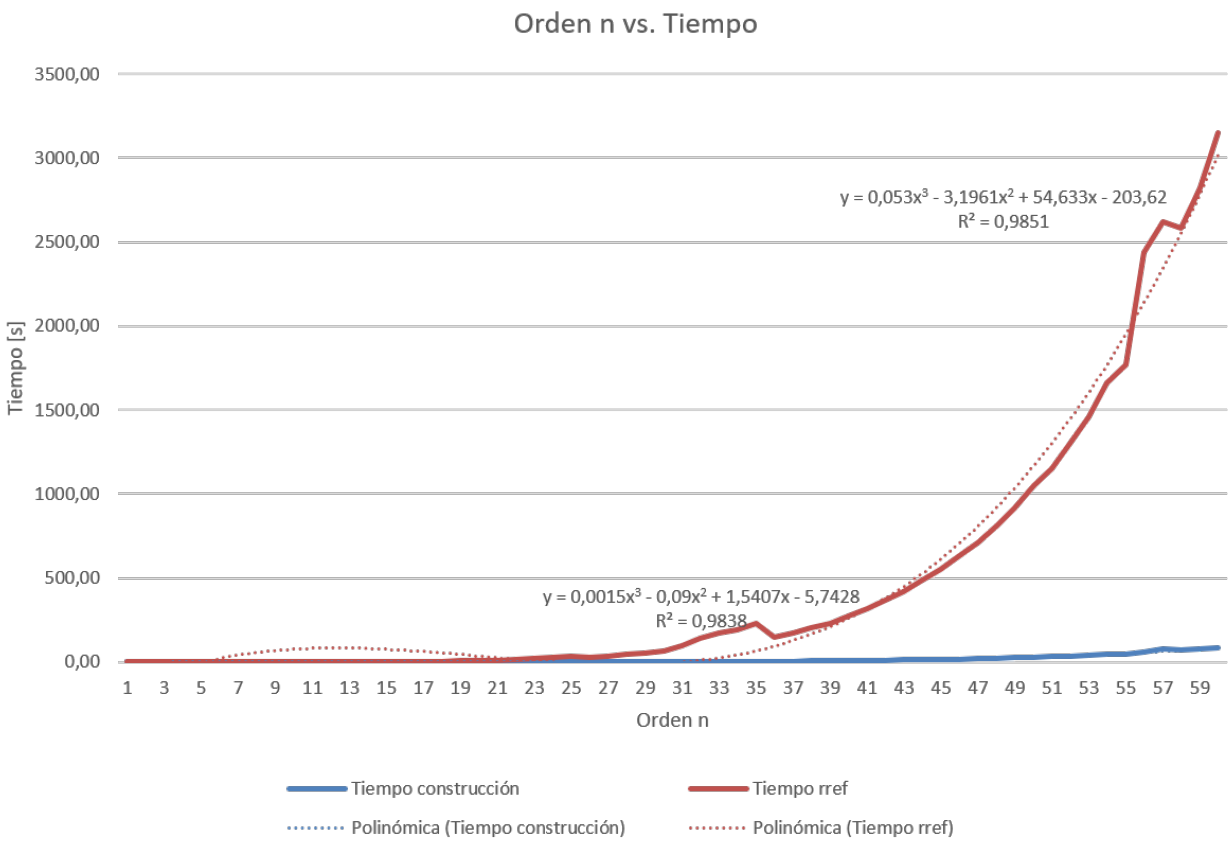


Figura D.1: Tiempos empleados por los Lights Out.

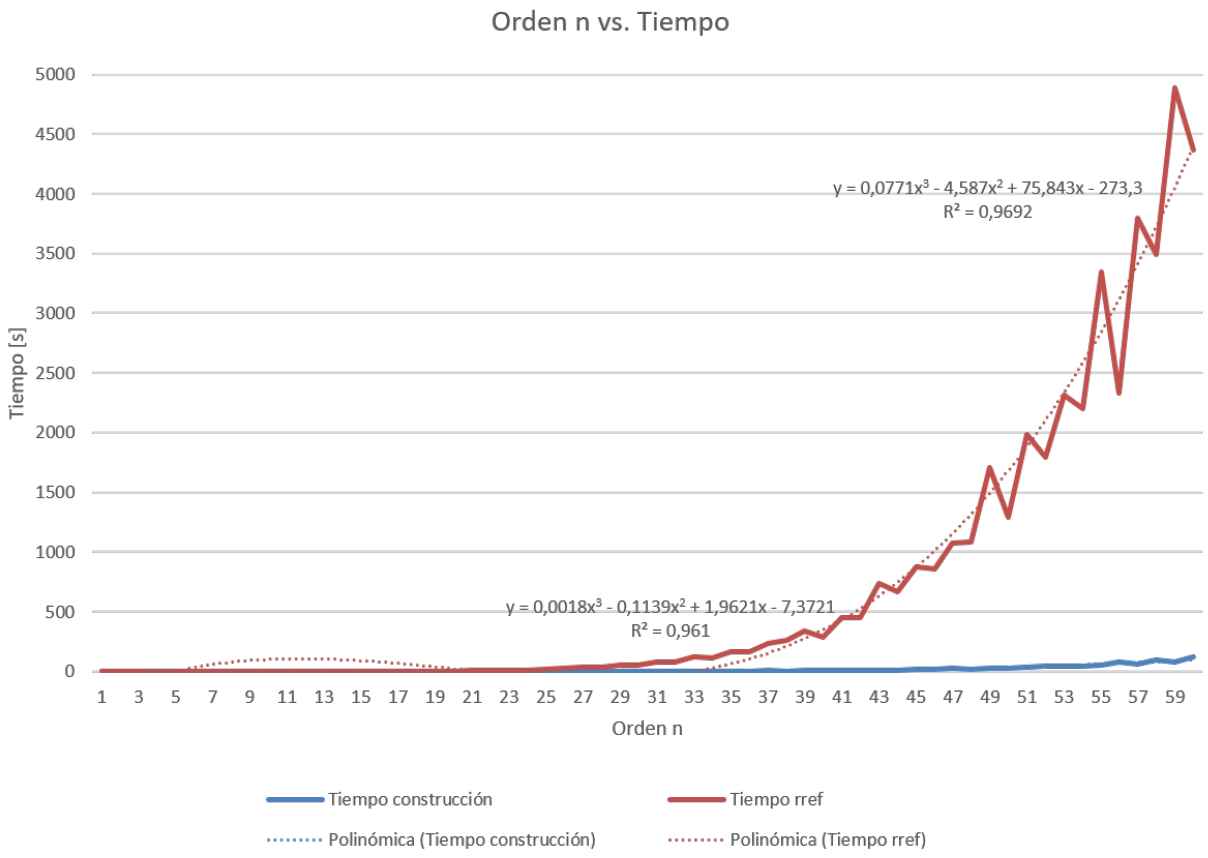


Figura D.2: Tiempos empleados por los Lights Out toroidales.

Apéndice E

Problema del triángulo

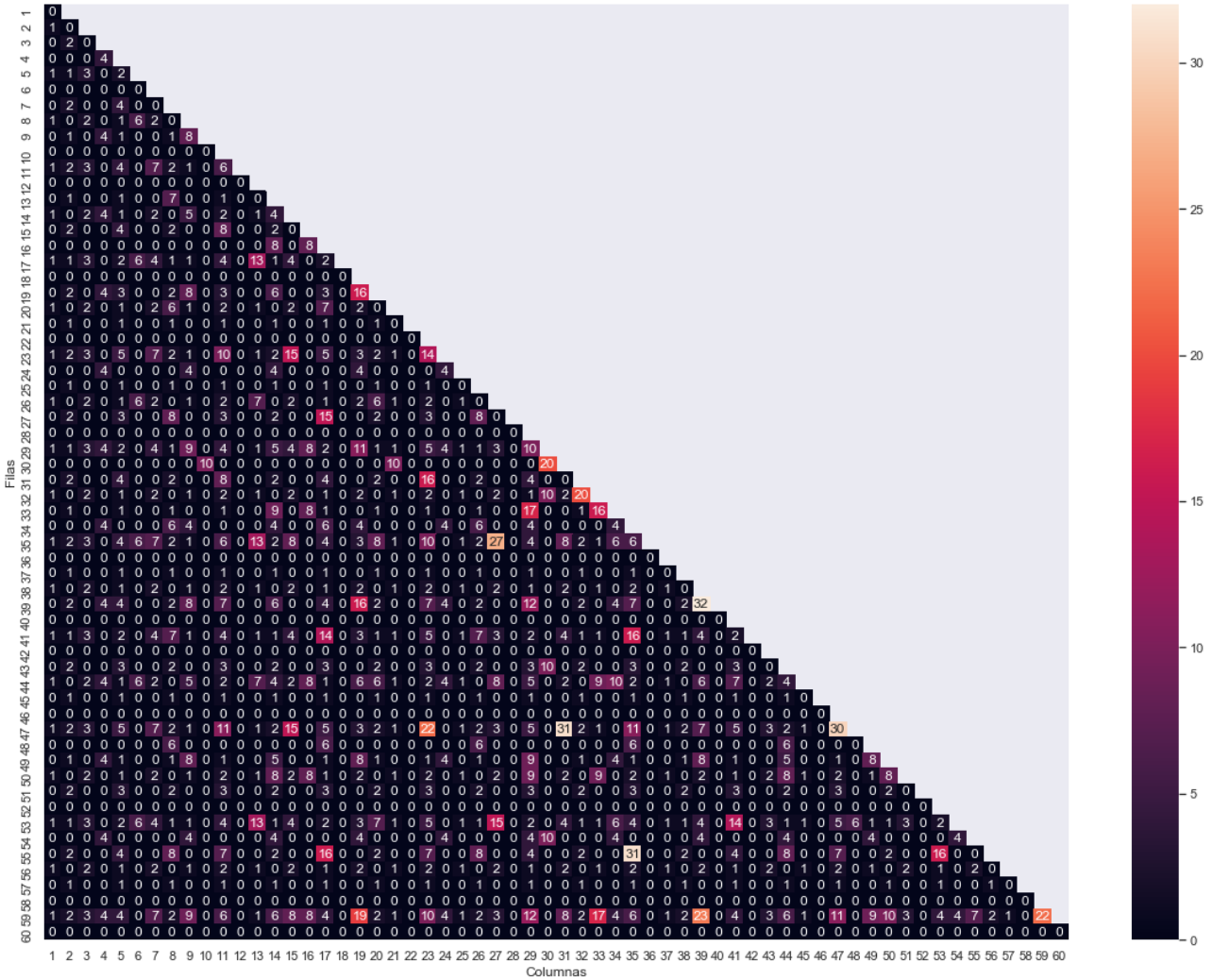


Figura E.1: Problema del triángulo para los Lights Out.

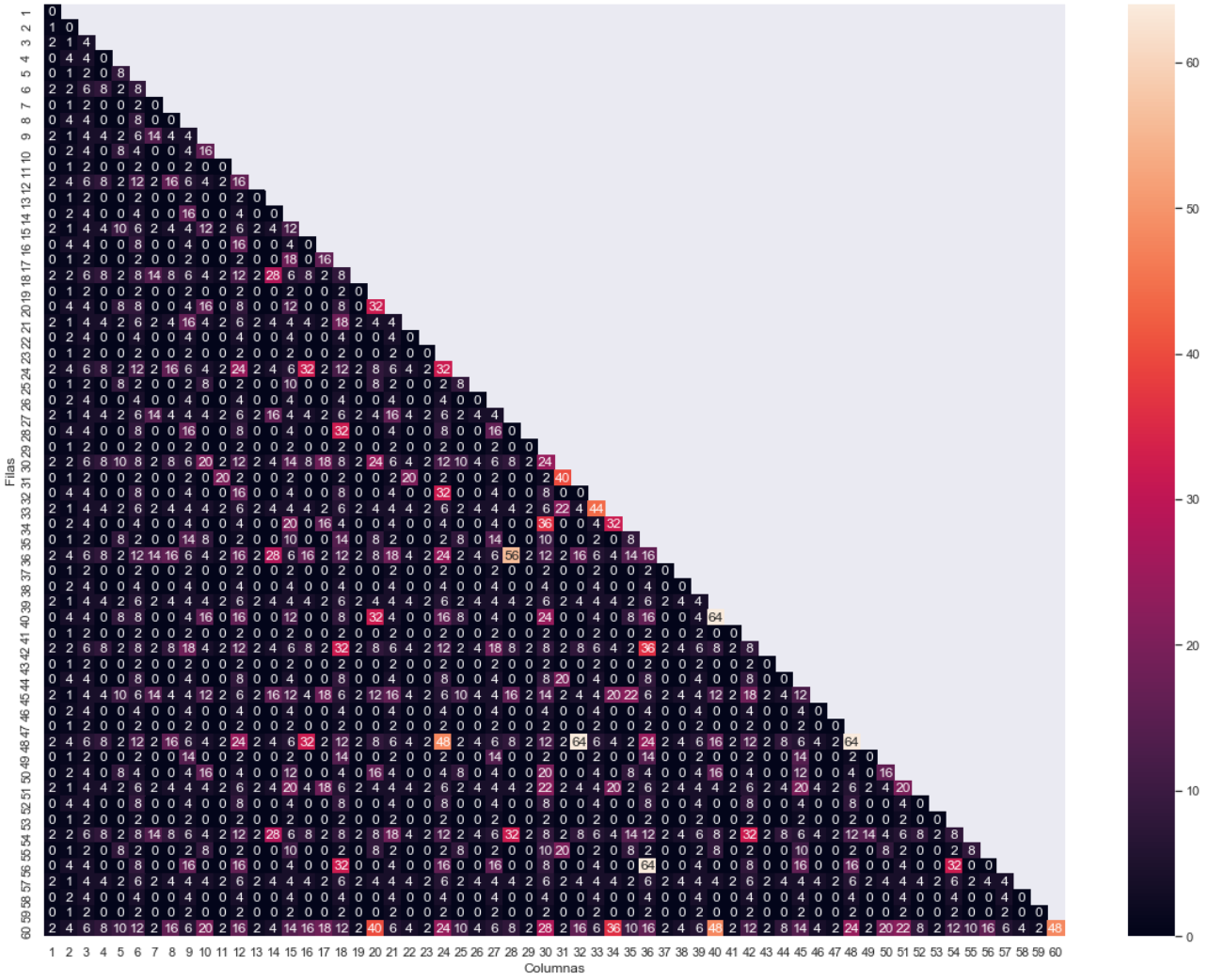


Figura E.2: Problema del triángulo para los Lights Out toroidales.