Universidad Nacional Autónoma de México
Facultad de Ciencias

Complejidad Computacional

Reporte Final

Bonilla Reyes Dafne - 319089660
García Ponce José Camilo - 319210536
Juárez Ubaldo Juan Aurelio - 421095568
Moreno Castro Fernanda - 319128727

Batalla Naval (puzzle) es NP-Completo

Introducción

El puzzle de Batalla Naval, también conocido como Batalla Naval Solitario, es un juego que presenta interesantes desafíos tanto desde el punto de su resolución como de su análisis computacional. En este reporte, analizaremos Batalla Naval como un problema de decisión, basándonos en el documento "Battleships as Decision Problem.[es]crito por Merlijn Sevenster en 2004, para ver que este problema pertenece a la clase NP-Completo. Este análisis usa conceptos aprendidos durante el semestre, como las clases de complejidad, problemas NP-Completos y reducciones. También, hablaremos algo de las reducciones parsimoniosas, las cuales son importantes en la reducción usada para mostrar que Batalla Naval es NP-Completo. Y por ultimo, buscamos evidenciar como un puzzle aparentemente casual puede convertirse en un desafió significativo desde una perspectiva computacional.
Este puzzle fue inventado en Argentina y tuvo sus primeras apariciones en revistas de juegos en 1982, pero su popularidad aumento a partir de 1992, cuando fue incluido en un campeonato mundial de puzzles.

Problema de Battleship como Problema de Decisión

Battleship, además de ser un juego lógico ampliamente conocido, puede ser formalizado como un problema de decisión dentro del ámbito de la computación. Su análisis matemático ha mostrado su relación con problemas complejos como el *Bin Packing*. A continuación, definiremos las condiciones esenciales para este problema, las funciones involucradas y la estructura formal del mismo.

0.1. Condiciones del Problema

Para definir el problema de Battleship como problema de decisión, se deben considerar cuatro condiciones fundamentales:

1. **Ubicación en la cuadrícula**: Todos los barcos deben estar dentro de los límites definidos por la cuadrícula de tamaño $m \times n$.
2. **Restricciones iniciales**: Las posiciones iniciales dadas por la función I no pueden ser contradichas.
3. **No adyacencia**: Ningún par de barcos puede ocupar casillas adyacentes.
4. **Conteo por filas y columnas**: La cantidad de segmentos de barco en cada fila y columna debe coincidir con los valores predefinidos.

0.2. Definiciones Clave

El problema se representa mediante una tupla (I, C, R, F) sobre un tablero de tamaño $m \times n$:

- I: Es una función inicial que asigna a cada posición (i, j) uno de tres estados: *ship* (segmento de barco), *water* (agua) o *desconocido* (?).
- Ejemplo: Si $I(i, j) =$ water, esa celda está ocupada por agua.
- Ejemplo: Si $I(i, j) =$ ship, esa celda está ocupada por un segmento de barco.
- Ejemplo: Si $I(i, j) =$ desconocido, esa celda se le deja a la persona que va a resolver el problema .

También tenemos funciones de conteo:

- C: Es el número de segmentos de barco especificados para cada columna.
- R: Es el número de segmentos de barco especificados para cada fila.
- F: Es una función que indica la cantidad de barcos de una longitud k. Es decir, $F(k)$ devuelve el número de barcos de tamaño k en el tablero.

0.3. Función J: Solución del Problema

La función J genera una configuración del tablero que respeta las siguientes condiciones:

1. Coincide con I en las posiciones no desconocidas (?).
2. Satisface las restricciones dadas por C, R y F.

Si J cumple con estas condiciones, se considera una solución válida del problema.

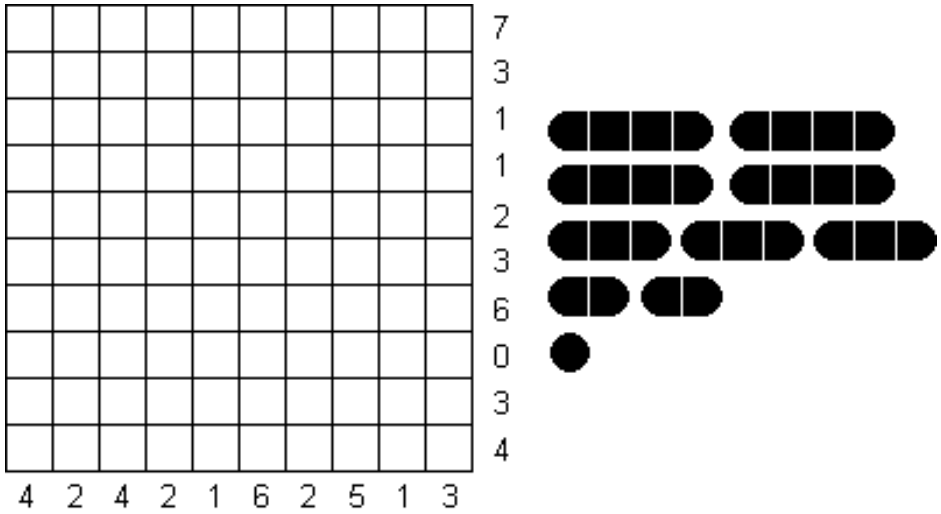
0.4. Formulación del Problema de Decisión

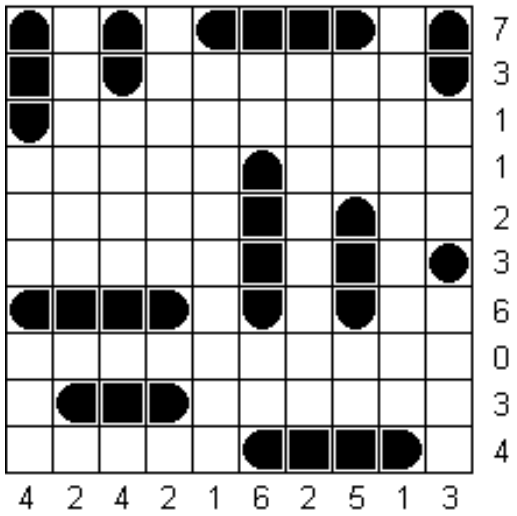
Para enunciar Battleship como problema de decisión, necesitamos especificar un ejemplar y una pregunta:

- **Ejemplar**: Una tupla (C, R, F, I) que describe el tablero inicial y las restricciones.
- **Pregunta**: ¿Existe una función J tal que cumpla con las condiciones $C1$ a $C4$ respecto a C, R y F?

Así el problema busca determinar si existe una configuración válida del tablero que satisfaga todas las restricciones dadas.

Ejemplo de puzzle





Ejemplo del puzzle solucionado

Reducciones de parsimonious

Veamos la definición formal de las reducciones de parsimonious, sean $R, R' \in PC$ y sea g ima reducción de Karp (las reducciones vistas en clase) de $S_R = \{x : R(x) \neq \theta\}$ a $S_{R'} = \{x : R'(x) \neq \theta\}$, donde $R(x) = \{y : (x, y) \in R\}$ y $R'(x) = \{y : (x, y) \in R'\}$. Decimos que g es una reducción de parsimonious si para todo x se cumple que:

$$|R(x)| = |R'(g(x))|$$

Veamos que tanto R como R', x representa una instancia del problema y y su solución, asi que de manera menos formal, podemos decir que las reducciones de parsimonious son reducciones en las cuales para cualquier instancia x de R, la cantidad de soluciones es igual a la cantidad de soluciones $g(x)$ en R'.
Vamos a ver algunas de sus caracteristicas importantes:

- **Conservación de soluciones**. Este tipo de reducciones no solamente transforman instancias de problemas, sino que mantienen intacto el número de soluciones.
- **Relacion biyectiva**. Ademas de conservar el número de soluciones, podemos establecer una relacion uno a uno entre ambos conjuntos de soluciones.
- **Problemas #P-completos**. Utilizamos estas reducciones para demostrar que un problema pertenece a la clase de problemas #P-completos.

0.5. Problemas #P-completos

Para definir esta clase de problemas, decimos que un problema es #P-completo, si:

- Pertence a #P, es decir, el problema consiste en contar soluciones de una instancia de un problema en NP.
- Cualquier otro problema en #P se reduce con una reducción de parsimonious a él.

Un ejemplo clasico es el problema $\#SAT$, recordemos que en el problema SAT visto en clase, queremos verificar si una asignacion de variables satisface una formula booleana. En esta versión, lo que queremos es contar el número de asignaciones que satisfacen una formula booleana.

Ejemplos de Reducciones Parsimoniosas

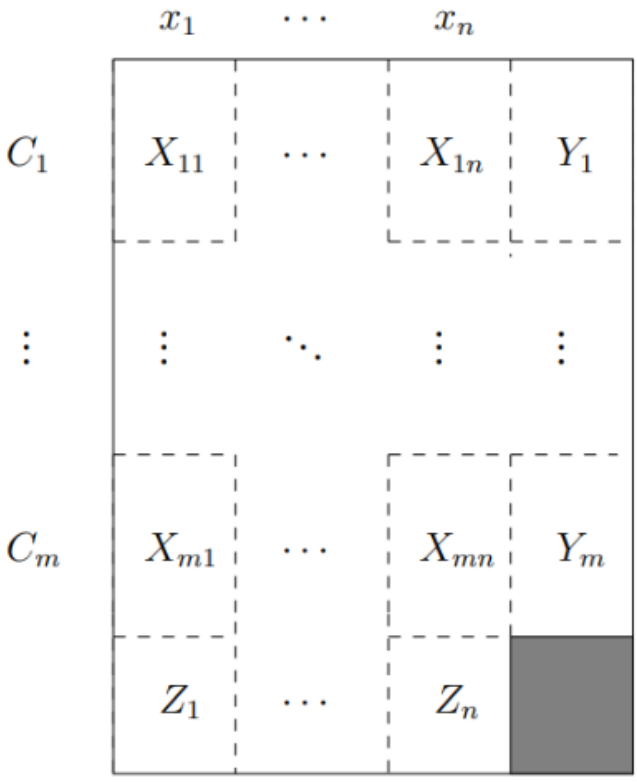
1. **De #SAT a #3SAT**: La transformación de fórmulas booleanas generales a su forma 3-CNF puede hacerse parsimoniosamente, preservando el número de soluciones, lo que demuestra que #3SAT es #P-completo.
2. **Conteo de Caminos Hamiltonianos**: La reducción de este problema a otros problemas de conteo en grafos (como conjuntos independientes) puede realizarse parsimoniosamente en algunos casos, como en el conteo de Caminos Hamiltonianos, ya que si reducimos el problema a un grafo con pesos específicos, el número total de soluciones sigue siendo el mismo.

Con estos ejemplos, podemos ver como se aplican las reducciones de parsimonious para preservar la cantidad de soluciones.

Reducción

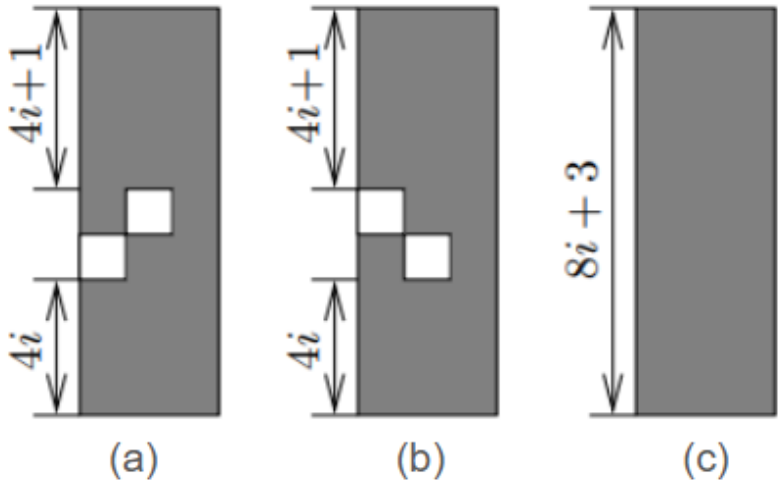
Para ver que este problema es NP-Completo, se realizó una reducción de 3-SAT (es el conjunto de fórmulas booleanas satisfacibles en forma CNF tal que tiene exactamente 3 literales por cláusula) a Batalla Naval, para facilitar el proceso se usa un problema intermedio llamado 3,{3-4}-SAT (es el subconjunto de 3-SAT tal que contiene las fórmulas booleanas satisfacibles con exactamente 3 literales y cada variable aparece tres o cuatro veces en toda la fórmula).
Una nota importante que hace el documento base/principal de este reporte es que se asume que ninguna variable aparece dos veces en la misma cláusula.
Por lo tanto, primero se realizó una reducción parsimoniosa de 3-SAT a 3,{3-4}-SAT y luego otra reducción parsimoniosa de 3,{3-4}-SAT a Batalla Naval.
La reducción de 3-SAT a 3,{3-4}-SAT la realizó Craig Tovey en un escrito llamado .ª Simplified NP-Complete Satisfiability Problem", esto lo realizó con una reducción parsimoniosa.

Ahora veamos cómo es la reducción de 3,{3-4}-SAT a Batalla Naval. Para esto vamos a ver cómo es la transformación.
Sea $\varphi = C_1 \wedge C_2 \wedge ... \wedge C_m$ sobre las variables $x_1, x_2, ..., x_n$, una fórmula booleana con m cláusulas tales que cada cláusula tiene exactamente 3 literales y n variables tales que cada variable aparece 3 o 4 veces en φ, ahora construiremos el tablero inicial para el puzzle de batalla naval, para esto tendremos el siguiente tablero.



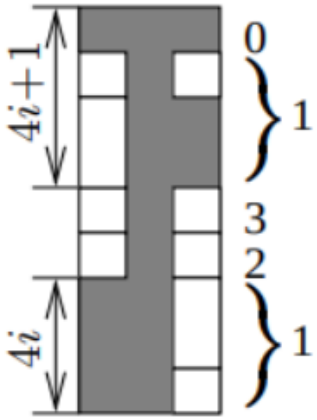
El cual tiene zonas las cuales vamos a ir llenando con piezas, notemos que la última zona (inferior derecha) del tablero solo contiene agua (en general las partes gris oscuro de las figuras/gadgets representan agua y las partes blancas son desconocido o ?).
Veamos que cada zona del tablero tiene un nombre/identificador como X_{ij}, Y_i o Z_j, veamos que van a contener estas zonas.

La casilla/zona X_{ij} está asociada con la cláusula C_i y la variable x_j, pondremos una de tres posibles figuras, si x_j aparece no negada en la cláusula C_i entonces ponemos la figura/gadget (a), si x_j aparece negada en la cláusula C_i entonces ponemos la figura/gadget (b) y si x_j no aparece en la cláusula C_i entonces ponemos la figura/gadget (c). Además se añadirá una nave de longitud 1, la cual será la $X_{ij}-$nave.



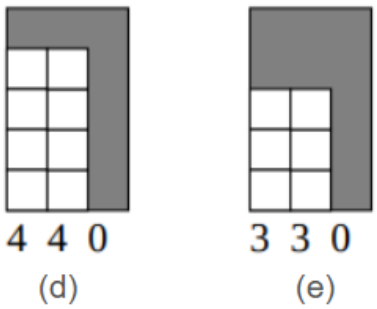
Observemos que estas figuras/gadgets tienen altura $8i + 3$ y anchura 3.

Las casilla/zona Y_i esta asociada con la cláusula C_i, aquí se usara la siguiente figura/gadget y ademas se agregaran una nave de longitud $4i + 1$, una de longitud $4i$ y una de longitud 1, serán las Y_i-naves, notemos que esta figura/gadget determinara los números de las filas que intersectan a Y_i



Observemos que esta figura/gadget tienen altura $8i + 3$ y anchura 3.

La casilla/zona Z_i esta asociada con la variable x_i, aquí usaremos una de dos posibles figuras/gadgets, si x_j aparece cuatro veces en φ entonces ponemos la figura/gadget (d) y si x_j aparece tres veces en φ entonces ponemos la figura/gadget (e). Ademas se añadirá una nave de longitud cuatro si x_j aparece cuatro veces en φ o una nave de longitud tres si x_j aparece tres veces en φ, la cual será la Z_j-nave. Notemos que esta figura/gadget determinara los números de las columnas que intersectan a Z_j



(d) (e)

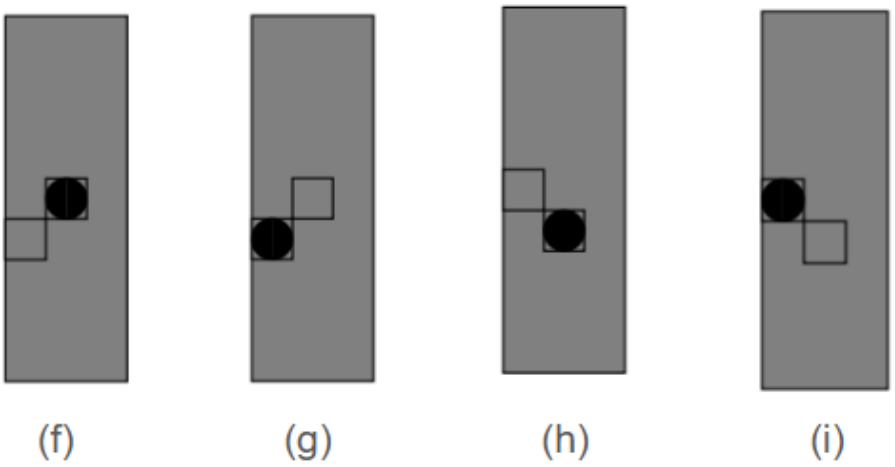
Observemos que esta figura/gadget tienen altura 5 y anchura 3.

Concluimos la transformación asignamos los números a las filas y columnas de la ultima zona (inferior derecha) del tablero, de la siguiente manera.
Para la fila $9m+1$ tiene el número 0, la fila $9m+2$ tiene el número n_4 (con n_4 siendo el número de variables que aparecen cuatro veces), la fila $9m+3$ tiene el número n, la fila $9m+4$ tiene el número n y la fila $9m+3$ tiene el número n.
Para la columna $3n+1$ tiene el número $\sum_{i=1}^{m} 4i+1$, la columna $3n+2$ tiene el número 0 y la columna $3n+3$ tiene el número $\sum_{i=1}^{m} 4i+1$.

De esta manera termina la transformación de la fórmula booleana φ a un ejemplar del problema de Batalla Naval (el tablero inicial, los barcos y los números de las filas y columnas).
Notemos que el tablero tiene una altura de $(\sum_{i=1}^{m} 8i+3)+5$ y una anchura de $3(n+1)$, por lo tanto en total tendremos $((\sum_{i=1}^{m} 8i+3)+5)*3(n+1)$ casillas en total, las cuales son $12m^2n+12m^2+21mn+21m+15n+15$ casillas.
Y el total de naves es $nm+3m+n$ (nm son las de X_{ij}s, $3m$ las de Y_is y n las de Z_is).
Entonces podemos notar que construir el tablero, las naves y los números (hacer las transformación) nos toma tiempo polinomial en relación a φ, ya que φ debe contener todas sus m cláusulas y todas sus n variables.

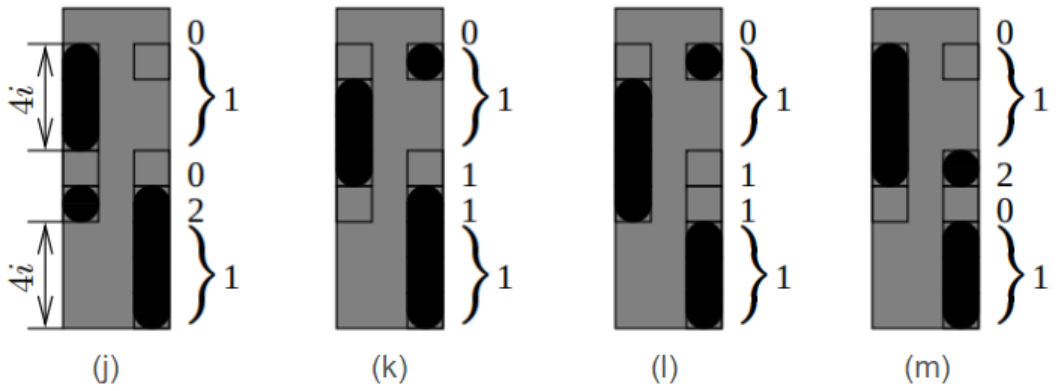
Ahora veamos que cada asignación de verdad que satisface a φ corresponde a exactamente una solución (como acomodar las naves, ya que usaremos el mismo tablero) del puzzle generado por la transformación y viceversa.
Dados valores de verdad $t : \{x_1, ..., x_m\} \to \{\text{true, false}\}$ tales que satisfacen a φ, resolveremos el puzzle de esta manera.

Para la casilla/zona X_{ij} pondremos la $X_{ij}-$nave dependiendo de la variable x_j y de la cláusula C_i.
Pondremos la nave en la posición noroeste (imagen (f)) si x_j aparece no negada en C_i y $t(x_j) = \text{true}$.
Pondremos la nave en la posición sureste (imagen (g)) si x_j aparece no negada en C_i y $t(x_j) = \text{false}$.
Pondremos la nave en la posición suroeste (imagen (h)) si x_j aparece negada en C_i y $t(x_j) = \text{true}$.
Pondremos la nave en la posición noreste (imagen (i)) si x_j aparece negada en C_i y $t(x_j) = \text{false}$.



Notemos que el valor de verdad de x_j contribuye o no en la verdad de la cláusula C_i depende de si la $X_{ij}-$nave esta en la parte norte o en la sur, si esta en la norte significa que va a contribuir (es decir la cláusula va a ser verdadera) y si esta en la sur no contribuye. Por lo tanto, como t es una asignación de verdad satisfacible, tendremos que al menos una literal en C_i es verdad, si aparece no negada, o es false, si aparece negada, por lo que en el puzzle significa que al menos una X_{ij} nave esta en la parte norte (fila superior).

Para la casilla/zona Y_i, primero notemos que para que Y_i y sus naves pueda ser satisfecho (cumplan con las condiciones de solución) por cualquier arreglo de $X_{ij}-$naves se necesita que el número de naves en la fila superior (en la parte norte) en las X_{ij} zonas que intersectan a Y_i sea al menos una y el número de las naves en la fila inferior (en la parte sur) sea a lo más dos. Dependiendo de la cantidad de naves en la fila superior, X_{ij} zonas que intersectan a Y_i, veremos como acomodar las Y_i-naves.
Si hay tres naves en la fila superior entonces el número de la fila superior ya se cumplió (necesita ser tres, es decir tres partes de naves), entonces las Y_i-naves se ponen en las posiciones indicadas por la imagen (j).
Si hay dos naves en la fila superior entonces el número de la fila superior aun no se cumple por lo cual las Y_i-naves se ponen en las posiciones indicadas por la imagen (k) o la imagen (l), pero para poder cumplir con las números de la primera y tercera columna tenemos que acomodar las naves como dice la imagen (l).
Si hay una naves en la fila superior entonces el número de la fila superior aun no se cumple pero el número de naves en la fila inferior ya se cumplió, por lo cual las Y_i-naves se ponen en las posiciones indicadas por la imagen (m).



De esta manera podemos acomodar las Y_i-naves dependiendo de las posiciones de las $X_{ij}-$naves, para

cumplir los números de las filas y se necesita que la cláusula C_i se evalúe en verdadero para cumplir los números de las filas (para cumplir el número tres).

Para la casilla/zona Z_j, notemos que t nos dice que cada variable x_j solo tiene un valor de verdad, entonces usaremos esto para acomodar la Z_j-nave, de esta manera si $t(x_j) = \text{true}$ significa que las naves relacionadas a esta variables ($X_{ij}-$naves) están en la columna derecha (por lo visto en como acomodar las naves para la zona X_{ij}), lo cual nos indica que el número de partes de naves en la columna derecha es tres (si x_j aparece tres veces en φ) o cuatro (si x_j aparece cuatro veces en φ), en ambos casos el número de la columna derecha ya se cumplió, entonces se pone la Z_j-nave en la columna de en medio para poder cumplir el número de la columna central (ya que el número de la columna izquierda siempre es cero), de esta manera cumpliendo los números en las columnas de Z_j, pero si $t(x_j) = \text{false}$, pasa algo muy similar, solo que la cantidad de piezas de naves en la columna de en medio cumple el numero de esa columna y por lo tanto la Z_j-nave se pone en la columna derecha.
De esta manera podemos acomodar la Z_j-nave dependiendo de as posiciones de las $X_{ij}-$naves y que x_j solo tiene un valor de verdad dado por t.

Ahora digamos que el puzzle generado por la transformación de una fórmula booleana φ (que en cada cláusula tiene exactamente tres literales y cada variable solo aparece tres o cuatro veces, y tiene m cláusulas y n variables) tiene una solución.
Mediante inducción podemos ver que en cada posible solución las Y_i-naves de longitud $4i+1$ y $4i$ deben estar en la zona Y_i, ya que es el único lugar donde pueden encontrar lugar, el caso base seria para las Y_m-naves de longitud $4m+1$ y $4m$ las cuales deben ir en la zona Y_m, ya que es la única zona donde pueden estar y de esta manera todas las Y_i-naves de longitud $4i+1$ y $4i$ deben estar en sus respectivas zonas Y_is.
También podemos notar que si el puzzle tiene una solución entonces cada Z_j-nave debe estar en las zonas Z_js, ya que es el único lugar donde pueden encontrar lugar y por lo tanto en las zonas X_js deben contener las X_j-naves.
La posición de cada Z_j-nave en la zona Z_j fuerza que los valores de verdad de cada variable x_j sean constantes (es decir que solo tengan un valor de verdad y de igual manera que solo están en la columna de en medio o en la derecha en las zonas X_j) para cada clausula en φ y como el puzzle tiene una soluciones, entonces los números de las filas se cumplen y eso significa que las zonas Y_i son resultas (se cumplen los números de las filas) lo que implica que cada clausula C_i es verdadera (ya que se cumple la fila con número tres, lo que significa que al menos una literal de la clausula aporta verdad). Ademas la asignación de las X_j-naves (en todas las zonas X_js) codifican una asignación de verdad, la cual hace que φ sea satisfactoria, por lo visto antes.
De esta manera podemos ver que la transformación es valida (una transformación Karp) y ademas el numero de soluciones se mantiene, es decir si es una reducción parsimoniosa.

Con lo que podemos notar que 3,{3-4}-SAT se puede reducir al Batalla Naval, y como 3-SAT se puede reducir a 3,{3-4}-SAT entonces tenemos que 3-SAT se puede reducir a Batalla Naval, y eso significa que Batalla Naval es NP-Difícil.

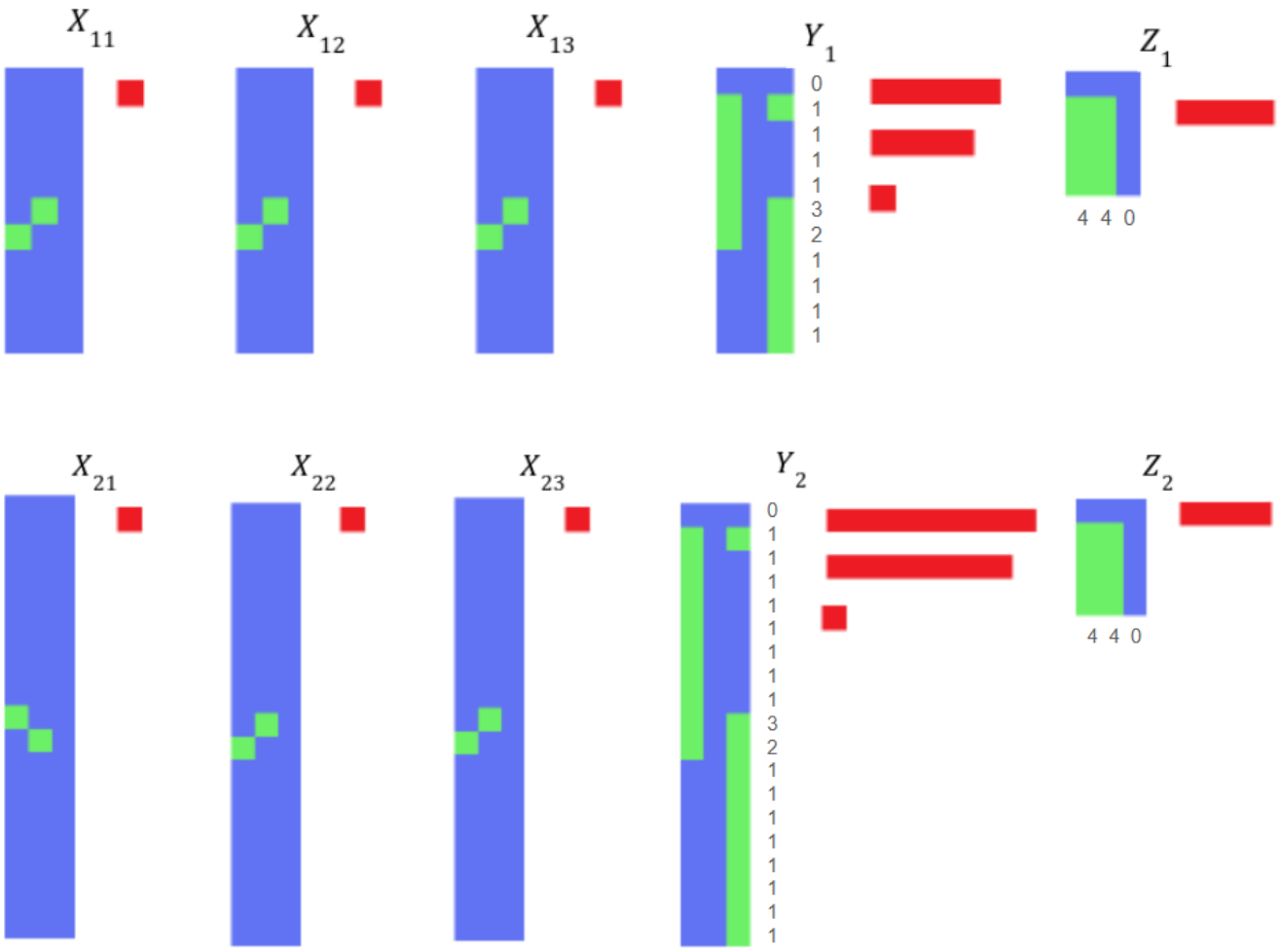
Ahora notemos que un certificado puede ser un tablero J con todas las casillas en agua o nave, es decir la función J definida antes o solo un acomodo de las naves en el tablero, entonces fácilmente podemos ver que una maquina de Turing determinista puede revisar si es una solución o no (revisando si cumple las cuatro condiciones definidas con anterioridad) con solo recorrer varias veces el tablero inicial (es decir la tupla que es el ejemplar del problema), por lo tanto nos tomaría tiempo polinomial verificar si el certificado si es una solución y ademas el certificado tendría tamaño polinomial, ya que seria del mismo tamaño del tablero del puzzle. Entonces Batalla Naval es NP.

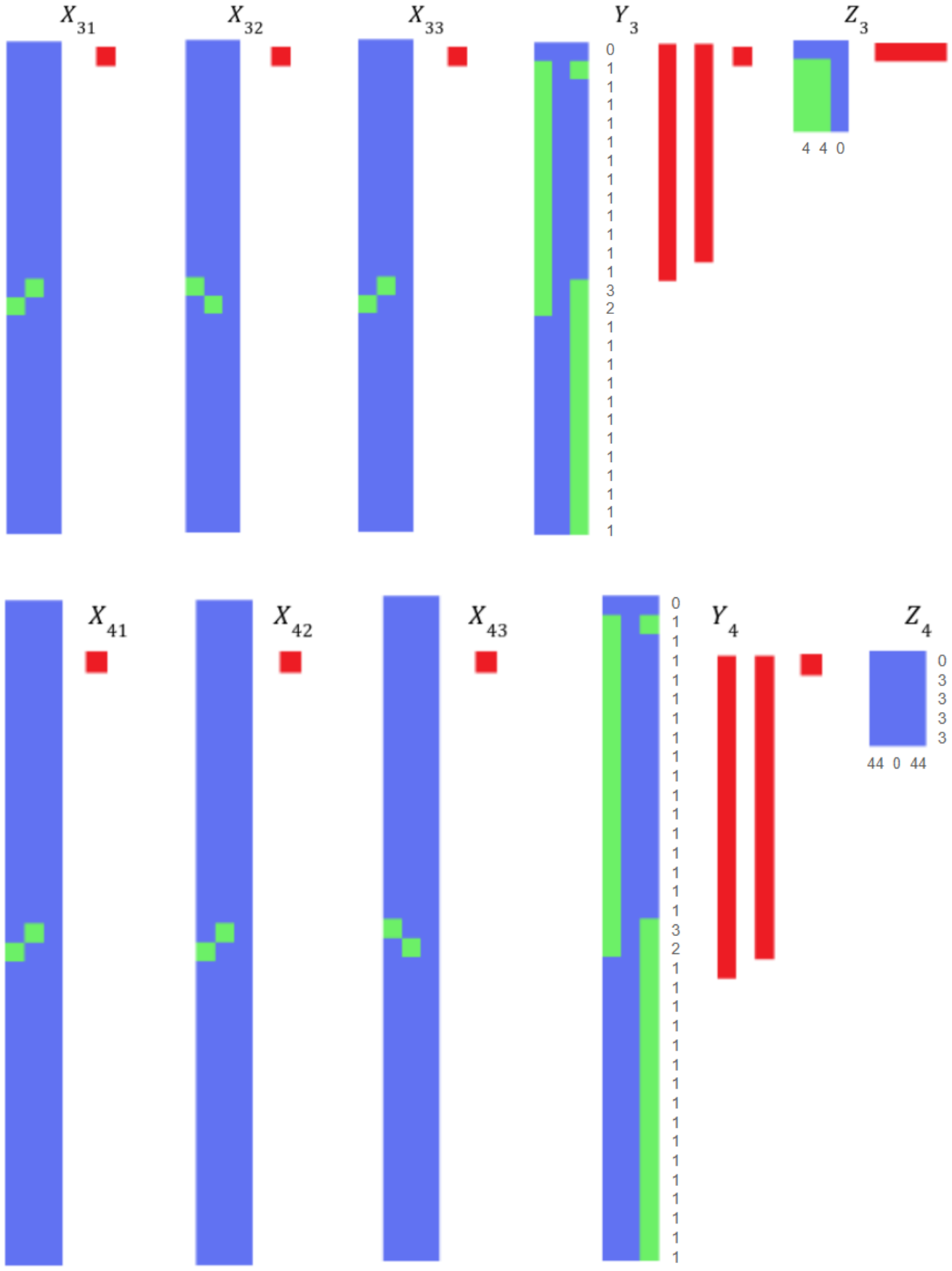
Y por ultimo, concluimos (de igual manera que el documento base/principal) que Batalla Naval es NP-Completo.

Ejemplo

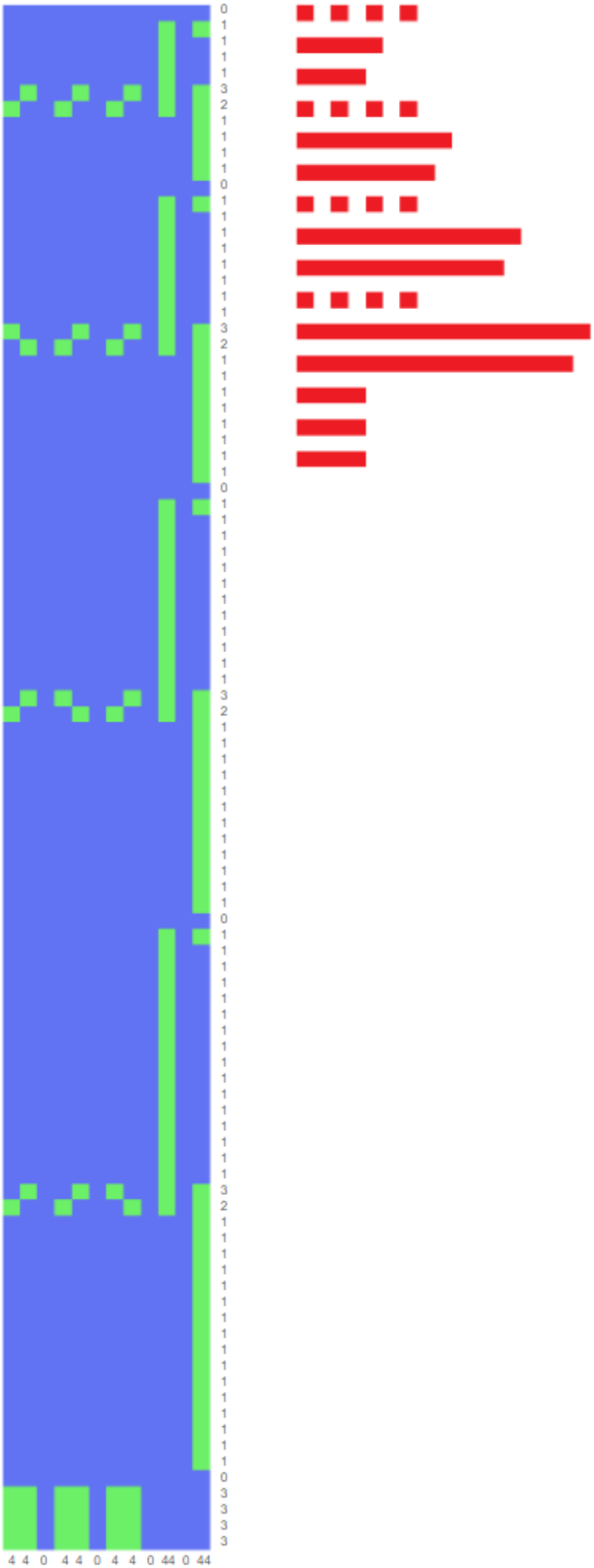
Ahora veamos un ejemplo de la transformación anterior.
Para esto usaremos que $\varphi = (a \vee b \vee c) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (a \vee b \vee \neg c)$, las clausulas son $C_1 = (a \vee b \vee c)$, $C_2 = (\neg a \vee b \vee c)$, $C_3 = (a \vee \neg b \vee c)$ y $C_4 = (a \vee b \vee \neg c)$, y las variables son $x_1 = a$, $x_2 = b$ y $x_3 = c$.

Lo primero realizado es construir las zonas que formaran el tablero final (las partes azules son el agua, las verdes son las casillas desconocidas y lo rojo son las naves).

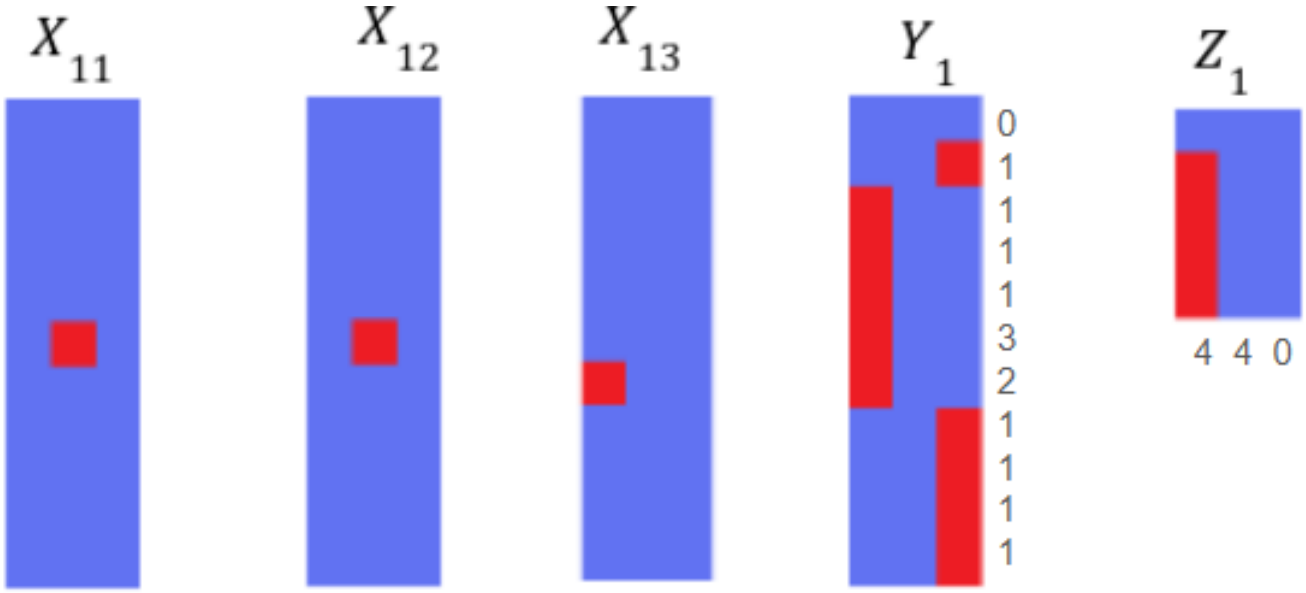


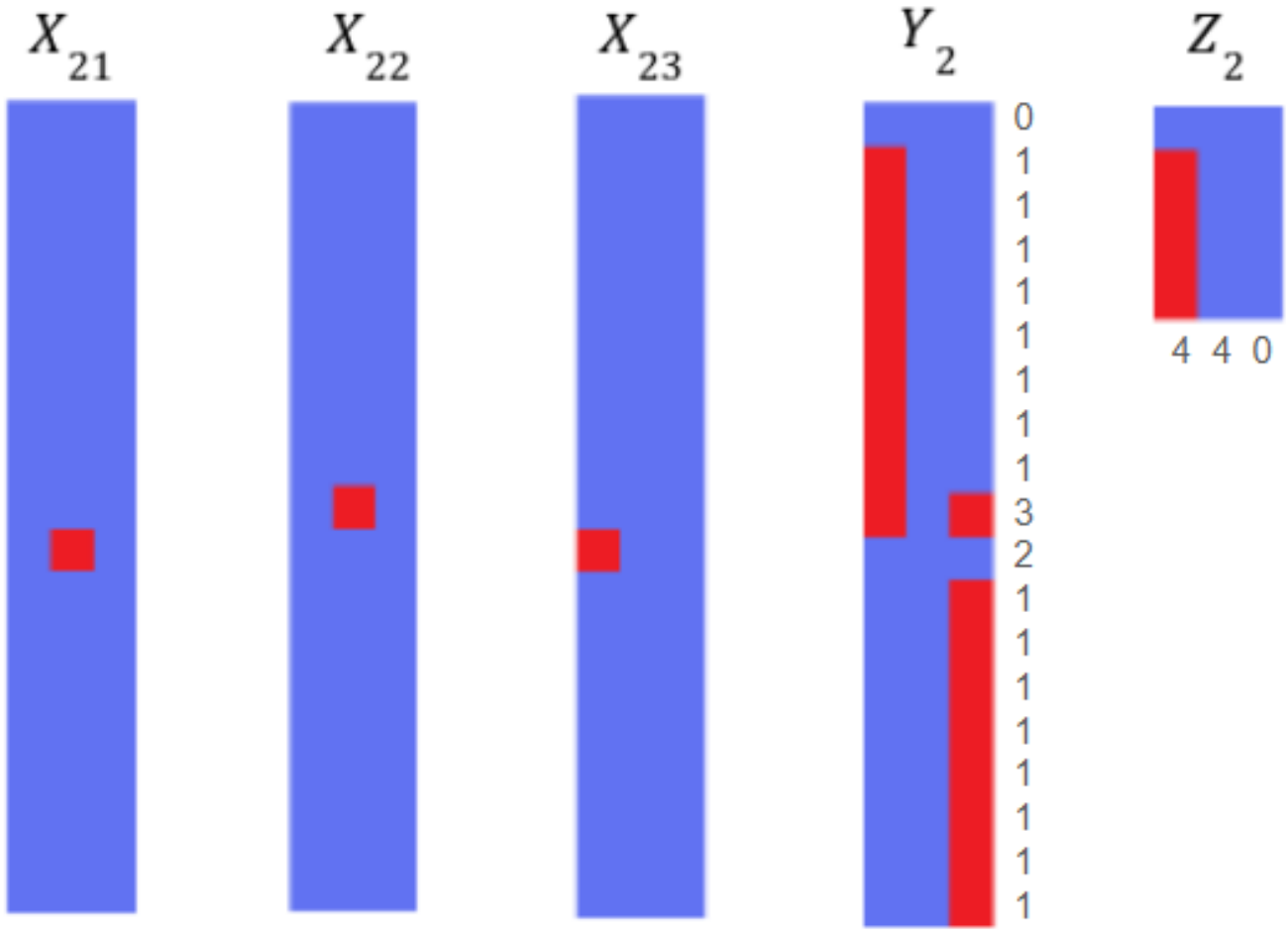

X_{31}
X_{32}
X_{33}
Y_3
Z_3
0 1 1 1 1 1 1 1 1 1 1 1 1 3 2 1 1 1 1 1 1 1 1 1 1 1 1 1
4 4 0
X_{41}
X_{42}
X_{43}
Y_4
Z_4
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 3 3 3 3
44 0 44

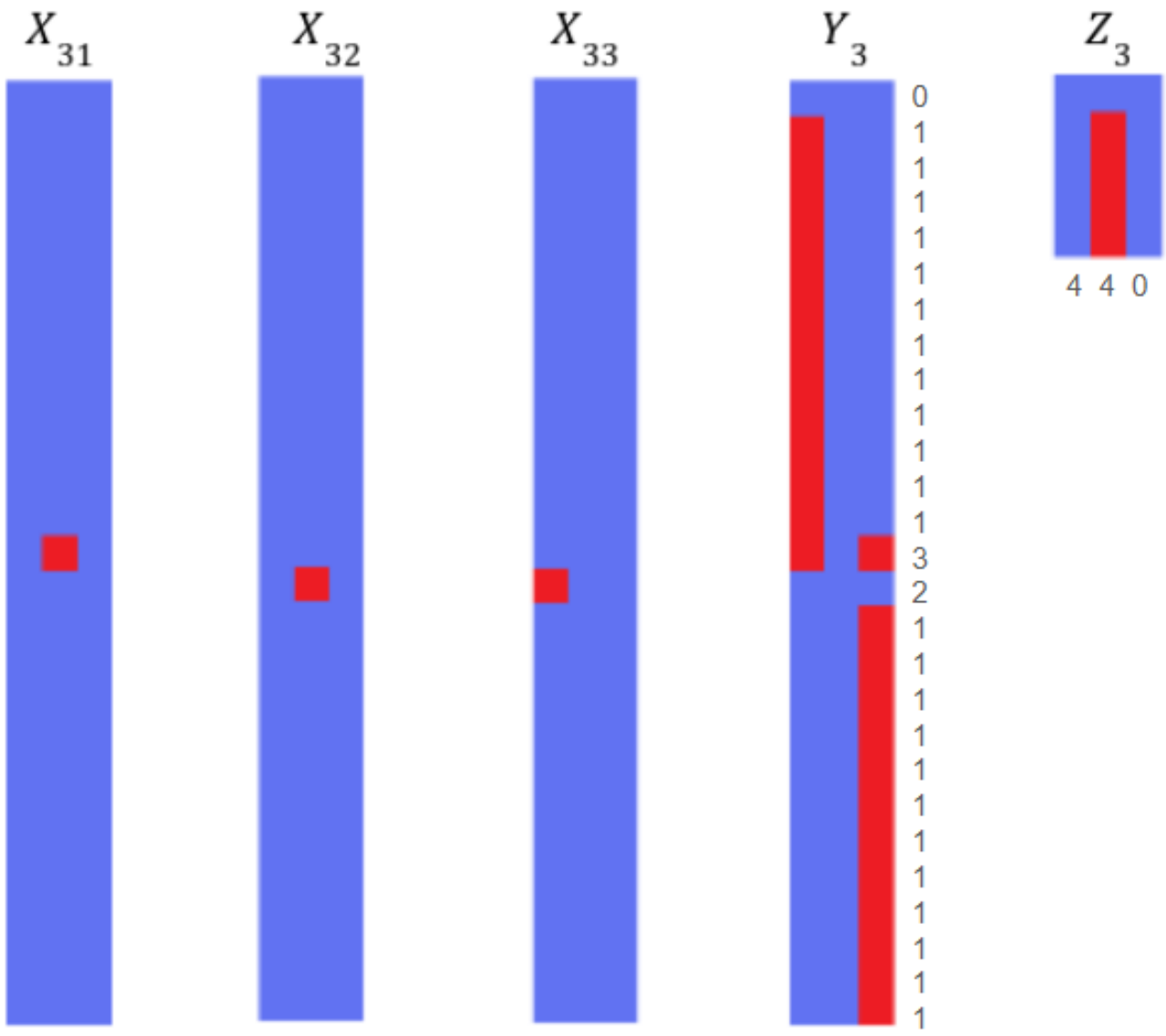
Luego juntamos todas las partes generadas y tendremos el tablero inicial, las naves y los números de las filas y columnas. Nota, los números pueden verse algo pequeños.

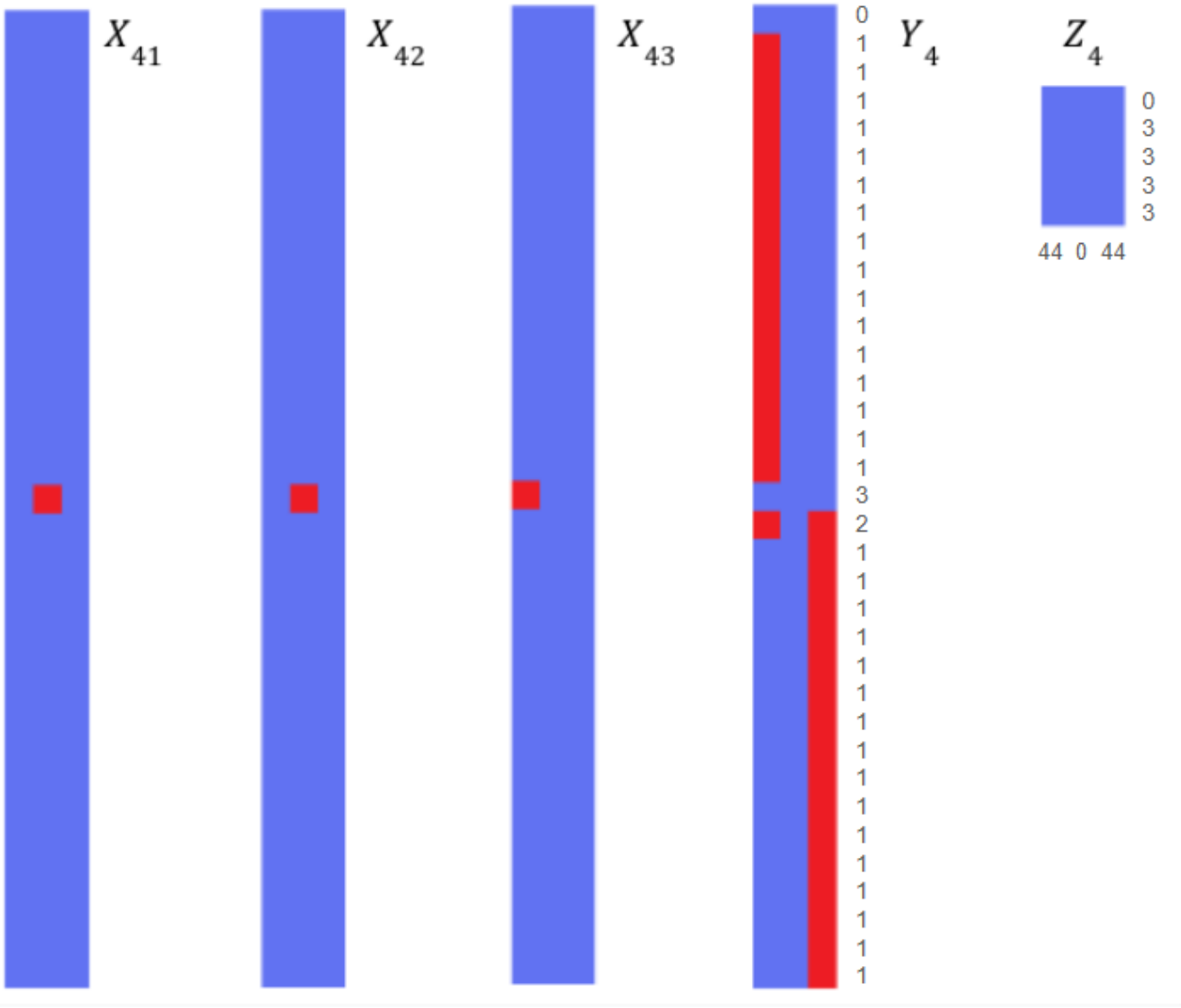


Ahora usaremos la siguiente asignación de verdad a =true, b =true y c =false, notemos que esta asignación de verdad satisface a φ. Entonces usaremos esto para poder acomodar las naves de tal forma que tengamos una solución para el puzzle.
Acomodamos las naves de cada zona.

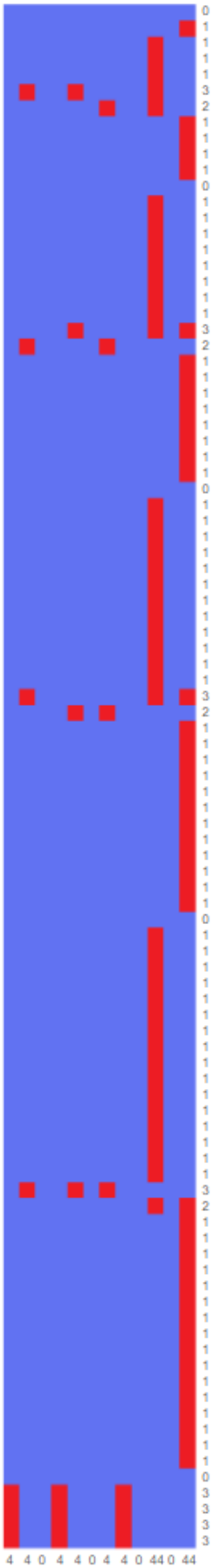



X_{21}
X_{22}
X_{23}
Y_2
Z_2
0
1
1
1
1
1
1
1
1
3
2
1
1
1
1
1
1
1
1
4 4 0


X_{31}
X_{32}
X_{33}
Y_3
Z_3
0 1 1 1 1 1 1 1 1 1 1 1 1 3 2 1 1 1 1 1 1 1 1 1 1 1 1
4 4 0



Por ultimo, juntamos todo y de esta manera tenemos la solución para el puzzle.



Conclusiones

En conclusión, el problema de *Batalla Naval Puzzle* pertenece a la clase de problemas NP-completos, lo que implica que, en general, no puede resolverse de manera eficiente mediante algoritmos deterministas para instancias de gran tamaño. Esto resalta la necesidad de explorar enfoques avanzados, como el uso de *inteligencia artificial* y técnicas heurísticas, para abordar casos más complejos o adaptados. Entre las posibles estrategias están:

- **Búsqueda A***: para explorar configuraciones prometedoras.
- **Algoritmos genéticos**: que generan soluciones iniciales y mejoran iterativamente mediante selección, cruce y mutación.
- **Redes neuronales**: capaces de predecir configuraciones de barcos con base en datos de problemas resueltos.

Además, este problema puede reducirse a otros problemas clásicos como el *coloreo de grafos* o el *empaquetado binario (bin packing)*, lo que subraya su relevancia dentro de la teoría de complejidad y su interconexión con otros problemas NP-completos.
Estudiar *Batalla Naval Puzzle* fomenta el desarrollo de técnicas algorítmicas avanzadas, como la *programación dinámica* y las *reducciones*, y pone de manifiesto las limitaciones actuales de los métodos exactos frente a problemas complejos. Más allá de la teoría, sus aplicaciones prácticas incluyen modelado en ingeniería, como el diseño de circuitos electrónicos y redes, y su dificultad lo convierte en una herramienta pedagógica ideal para enseñar algoritmos y complejidad computacional.
Por último, estas características lo hacen perfecto para diseñar juegos de lógica desafiantes y estimulantes, ofreciendo tanto retos intelectuales como oportunidades de innovación tecnológica.

Referencias

[1] Sevenster, M. (2004). BATTLESHIPS AS A DECISION PROBLEM. ICGA Journal, 27(3), 142–149. https://doi.org/10.3233/icg-2004-27303

[2] Tovey, C. A. (1984). A simplified NP-complete satisfiability problem. Discrete Applied Mathematics, 8(1), 85–89. https://doi.org/10.1016/0166-218x(84)90081-7

[3] Paulusma, D., & Szeider, S. (2019). On the parameterized complexity of (k,s)-SAT. Information Processing Letters, 143, 34–36. https://doi.org/10.1016/j.ipl.2018.11.005

[4] Goldreich, Oded (2008), Computational Complexity: A Conceptual Perspective, Cambridge University Press, pp. 203–204, ISBN 9781139472746

[5] Reductions Network. (2024). Reductions.network. https://reductions.network/parsimonious

[6] Wikipedia Contributors. (2024, November 25). Battleship (puzzle). Wikipedia; Wikimedia Foundation.