Sistema de resolución del juego "El Buscaminas" mediante el uso de redes bayesianas

Inteligencia Artificial – Curso 2017 / 2018

Miguel Ángel Antolín Bermúdez
Universidad de Sevilla
Sevilla, España
migantber@alum.us.es

David de la Torre Díaz
Universidad de Sevilla
Sevilla, España
davdedia@alum.us.es

Sumario*— *Uso de una red bayesiana para determinar con inferencia probabilística la existencia de mina en una casilla tras cada jugada.

Palabras clave*— *tablero, casillas, mina, juego, probabilidad, red
(key words)

I. INTRODUCCIÓN

"El buscaminas" (Minesweeper), es un videojuego para un jugador inventado por Robert Donner en 1989. El objetivo del juego es despejar un campo de minas sin detonar ninguna. En el juego se parte de una cuadrícula de dimensiones conocidas en la que hay escondidas una cantidad predeterminada de minas. En cada jugada, se descubre una de las casillas. Si la casilla descubierta contiene una mina, se pierde el juego. Si la casilla descubierta no contiene una mina, permite despejar una zona de la cuadrícula hasta las casillas que limitan con alguna casilla que contiene una mina. En estas últimas aparece información sobre el número de minas en las casillas colindantes. En la *figura 1* y la *figura 2* se pueden ver dos capturas del juego.

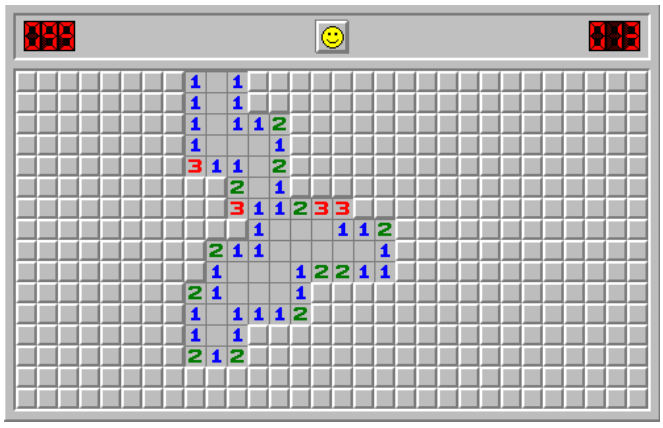

Figura 1: Tablero una vez realizado movimientos y sin haber descubierto mina. Los números contemplados da información sobre minas colindantes.

En este documento se describirán los pasos que hay que llevar a cabo para calcular, por medio de inferencia probabilística, la probabilidad que tiene cada casilla no descubierta del tablero de contener mina. Todo esto se llevará acabo construyendo una red bayesiana generada por la librería *Pgmpy*[4] que se irá actualizando con las evidencias de cada una de las casillas descubiertas en cada movimiento.

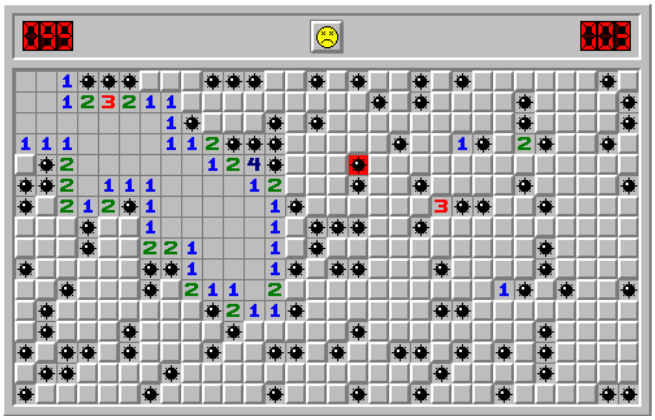

Figura 2: Situación del tablero tras descubrir una mina. Cuando esto ocurre, el juego se da por perdido.

Dada la complejidad exponencial de este algoritmo, se implementarán y propondrán métodos para la mejora de la inferencia.

II. OBJETIVO

Implementar un método que construya la red bayesiana asociada a un tablero dado, teniendo en cuenta que esta información será necesaria para construir la red.

Implementación de un método que determine las casillas con mayor probabilidad de no contener una mina y sugerirlas como siguiente paso a dar en el juego. Para ello se describirá la situación actual del juego mediante una red bayesiana, este mecanismo utilizará un sistema de inferencia exacta sobre la red (eliminación de variables) para deducir la probabilidad de que haya una mina en cada una de las casillas del tablero que no estén descubiertas usando como evidencia la información sobre las casillas del tablero descubiertas. Se sugerirán las casillas con mayor probabilidad de no contener una mina.

Implementar un mecanismo que desarrolle una partida completa del juego, eligiendo en cada paso la casilla con mayor probabilidad de no contener una mina. El resultado de descubrir dicha casilla (en caso de no finalizar el juego) se usará como entrada para dar el siguiente paso.

Alcanzar un rendimiento del sistema óptimo para distintas configuraciones del tablero a partir de la situación obtenida tras la primera jugada. Estas pruebas deben incluir los tamaños $5x5$ con 5 minas, $5x5$ con 6 minas, $5x5$ con 7 minas, $8x8$ con 13 minas, $8x8$ con 14 minas, $8x8$ con 15 minas, $10x10$ con 20 minas, $10x10$ con 22 minas, $10x10$ con 25 minas.

III. EL JUEGO DEL BUSCAMINAS

Este documento no tiene como objetivo la creación del juego en sí, por lo que se hará uso de una implementación ya realizada, "El Buscaminas"[1] hecho en Python. Sin embargo, se van a analizar detalladamente algunos aspectos de la implementación del mismo, con el objetivo de simplificar el entendimiento de este documento.

A. GENERACIÓN DEL TABLERO

Dados alto (*h)*, ancho (*w*) y número de minas (*num*), se crea un tablero en forma de matriz:

$$h \times w \quad donde \ \ h = w\,,\ (h,w) > 0$$
$$num \in [0, h \times w]$$

Para generar el mapa de minas se forma una matriz de ceros, del mismo tamaño que el tablero, por medio del paquete Numpy[2]. Para añadir las bombas (convertir algunos ceros en unos), se realizan permutaciones aleatorias y se cambian los valores de las casillas seleccionadas. Se sustituirán los valores de tantas casillas como número de minas se haya indicado en la inicialización del tablero.

Esta implementación brinda una forma de poder comprobar en cualquier momento del juego el estado del tablero. Esta información se suministra a través de una matriz de tamaño igual al tablero donde en cada casilla (i, j) se obtiene la información relativa a:

$$info(X_{i,j}) \begin{cases} 9 & si\ x\ está\ marcada \\ 10 & si\ x\ es\ incógnita \\ 11 & si\ x\ no\ está\ descubierta \\ 12 & si\ x\ es\ una\ mina \\ 0 \le x \le 8 & si\ x\ ninguna\ de\ las\ anteriores \end{cases}$$

El mapa obtenido tendrá una información contenida diferente en cada iteración del juego, a medida que el jugador vaya haciendo movimientos.

B. MOVIMIENTOS CONTEMPLADOS

El autor del proyecto contempla una serie de movimientos para la realización del juego. Algunos de ellos serán mencionados a posteriori en este documento.

Todos los movimientos están definidos por una posición $x \in h$ y $j \in w$ para poder ser manipulables:

a. Click[1]*:* Revela la información de una casilla. Si la casilla seleccionada contiene una mina, el juego acaba. Si por el contrario la casilla seleccionada no contiene una mina, se descubrirá esa casilla y la zona del mapa adyacente, si así procede.

b. Flag: Coloca en la posición dada un identificador visual que sirve de ayuda al jugador a tomar decisiones futuras. Este movimiento en ninguna circunstancia revela información del mapa. El único objetivo de este movimiento es permitir al jugador determinar, de forma subjetiva, la posición de una posible mina.

c. Unflag: Elimina de una posición dada un identificador colocado previamente por el jugador. Este movimiento en ninguna circunstancia revela información del mapa. Si la información de esa casillas es revelada por un movimiento *click*, el identificador se eliminará de dicha posición.

d. Question: Coloca en la posición dada un identificador visual que sirve de ayuda al jugador a tomar decisiones futuras. Este movimiento en ninguna circunstancia revela información del mapa. Al diferencia del movimiento *Flag*, el jugador hará uso de este movimiento cuando desconozca la existencia de una mina en la casilla dada.

C. OTRAS CONSIDERACIONES

Dado que el proyecto seleccionado se encuentra bajo licencia MIT, la cual permite modificaciones del mismo, se ha procedido a eliminar o modificar algunas funciones de ellas.

Se han eliminado conexiones TCP y otras definiciones de funciones que no resultaban relevantes para nuestra tarea.

1. Hacer el movimiento "click" equivale a descubrir el valor de la casilla y por consiguiente, se puede descubrir una bomba y perder el juego.

Se ha añadido código para obtener información directamente del juego y poder construir la red que calculará la inferencia. La implementación de dicho código se expondrá en los próximos apartados.

IV. DEL TABLERO A LA RED BAYESIANA

En la red que construiremos, hay dos tipos de variables aleatorias para cada casilla:

- Variables aleatorias *X* asociadas a la presencia de una mina en una casilla. Estas variables son de tipo entero, donde el valor 1 indica la presencia de la mina en la casilla, y el valor 0 indica que la casilla está libre. Todas estas variables se distribuyen de la misma forma y la probabilidad de que tomen el valor 1 dependerá del número de minas escondidas y del número de casillas del tablero. En el estado inicial sería:

$$P(X_{i,j}) = \frac{numMinas}{h \times w}$$

- Variables aleatorias *Y* asociadas a la cantidad de minas que hay en las casillas colindantes (vecinos) a una casilla. Estas variables son de tipo discreto y pueden tomar valores desde 0 hasta 8. En particular las variables aleatorias Y asociadas a las casillas de las esquinas sólo pueden tomar valores desde 0 hasta 3, y las asociadas a las casillas del perímetro sólo pueden tomar valores desde 0 hasta 5. Cada una de estas variables depende de las variables *X* colindantes. La probabilidad de que una de estas variables tome un valor dependerá del número de variables colindantes *X* que contengan una mina:

$$P(Y_{i,j} = y \mid padres(Y_{i,j}) = x) = \begin{cases} 1 & si\ y = \sum_{x \in\ x} X \\ 0 & en\ otro\ caso \end{cases}$$

Además, todas las combinaciones de los estados *x* de los padres de *Y* tienen que cumplir lo siguiente:

$$\sum_{y} P(Y_{k,l} = y \mid padres(Y_{i,j}) = x) = 1$$

Como ya se especifica en el apartado anterior, han sido necesarios ciertos métodos que nos ofrezcan información sobre el tablero con el fin de poder obtener la probabilidad descrita anteriormente, además de la información necesaria para construir la red bayesiana con su correspondiente grafo y tablas de probabilidad condicionales (CPDs).

Para poder obtener la lista de nodos que conformarán el grafo, que contendrá la información conocida sobre las casillas y sus relaciones de adyacencias con sus vecinos, hemos seguido los siguientes pasos:

a. Recorrer el tablero con dos índices (uno para cada dimensión) pasando por todas las casillas, almacenado su posición y creando una lista con sus casillas adyacentes:

```
nodos_nombrados = []
    para i de 0 a w:
        para j de 0 a h:
            nodos_nombrados.añadir("X"+ i + j)
```

b. Obtener los vecinos de una posición (i, j) *verificando su posición relativa en el tablero, ya que es condicionante del número de vecinos de una casilla, siendo vecinos de una casilla todos aquellos que cumplan las siguientes afirmaciones:*

a. Para cualquier casilla central:

- es_vecino_horizontal_izquierda sii. $x[g-1]$[2]
- es_vecino_horizontal_derecha sii. $x[g+1]$
- es_vecino_vertical_arriba sii. $x[g-w]$
- es_vecino_vertical_abajo sii. $x[g+w]$
- es_vecino_diagonal_izquierda_arriba sii. $x[g-w-1]$
- es_vecino_diagonal_izquierda_abajo sii. $x[g+w-1]$
- es_vecino_diagonal_derecha_arriba sii. $x[g-w+1]$
- es_vecino_diagonal_derecha_abajo sii. $x[g+w+1]$

b. Para las esquinas:

- Si ***i == 0 ^ j == 0***, es la esquina superior izquierda:
  - es_vecino_horizontal_derecha sii. $x[g+1]$
  - es_vecino_vertical_abajo sii. $x[g+w]$
  - es_vecino_diagonal_izquierda_abajo sii. $x[g+w-1]$
- Si ***i == w-1 ^ j == 0***, es la esquina superior derecha:
  - es_vecino_horizontal_izquierda sii. $x[g-1]$
  - es_vecino_vertical_abajo sii. $x[g+w]$
  - es_vecino_diagonal_derecha_abajo sii. $x[g+w+1]$
- Si ***(i == w-1) ^ (j == h-1)***, es la esquina inferior derecha:
  - es_vecino_vertical_arriba sii. $x[g-w]$
  - es_vecino_horizontal_derecha sii. $x[g+1]$
  - es_vecino_diagonal_izquierda_arriba sii. $x[g-w-1]$
- Si ***i == 0 ^ (j == h-1)***, es la esquina inferior izquierda:
  - es_vecino_vertical_arriba sii. $x[g-w]$
  - es_vecino_horizontal_izquierda sii. $x[g-1]$
  - es_vecino_diagonal_derecha_arriba sii. $x[g-w+1]$

c. Para el perímetro del tablero:

- Si ***i == 0***, es la parte izquierda del perímetro del tablero:
  - es_vecino_vertical_arriba sii. $x[g-w]$
  - es_vecino_vertical_abajo sii. $x[g+w]$
  - es_vecino_horizontal_derecha sii. $x[g+1]$
  - es_vecino_diagonal_derecha_arriba sii. $x[g-w+1]$
  - es_vecino_diagonal_derecha_abajo sii. $x[g+w+1]$
- Si ***i == w-1***, es la parte derecha del perímetro del tablero:
  - es_vecino_vertical_arriba sii. $x[g-w]$
  - es_vecino_vertical_abajo sii. $x[g+w]$

2. Mediante el uso de la variable *g=(j+1)+((i+1)-1)*w-1*, damos un nombre significativo a cada una de ellas que las mantiene relacionadas con su posición en la matriz del juego.

- es_vecino_horizontal_izquierda sii. $x[g-1]$
- es_vecino_diagonal_izquierda_arriba sii. $x[g-w-1]$
- es_vecino_diagonal_izquierda_abajo sii. $x[g+w-1]$

• Si ***j* = = *0***, es la parte inferior del perímetro del tablero.
- es_vecino_horizontal_izquierda sii. $x[g-1]$
- es_vecino_horizontal_derecha sii. $x[g+1]$
- es_vecino_vertical_abajo sii. $x[g+w]$
- es_vecino_diagonal_izquierda_abajo sii. $x[g+w-1]$
- es_vecino_diagonal_derecha_abajo sii. $x[g+w+1]$

• Si ***j* = = *h-1***, es la parte superior del perímetro del tablero.
- es_vecino_horizontal_izquierda sii. $x[g-1]$
- es_vecino_horizontal_derecha sii. $x[g+1]$
- es_vecino_vertical_arriba sii. $x[g-w]$
- es_vecino_diagonal_izquierda_arriba sii. $x[g-w-1]$
- es_vecino_diagonal_derecha_arriba sii. $x[g-w+1]$

c. Dados los vecinos de cada casilla, creamos un conjunto con todos los nodos y las aristas relacionadas, para añadirlas posteriormente al grafo a modelar:

```
para i de 0 a w:
    para j de 0 a w:
        vecinos = vecinos_de_posicion(i, j)
        para cada vecino x en vecinos:
            grafo.añadir(vecinos[x],"Y"+i+j)
```

d. Añadimos al modelo bayesiano la información obtenida anteriormente junto con las funciones de probabilidad de X e Y, para obtener las CPDs de la variable X, se hace uso de la probabilidad inicial dada por la formula de probabilidad inicial de las X vista anteriormente. A partir del resultado obtenido creamos una CPD por cada X del grafo asignando los valores correspondientes.

```
    resX = [Todos los nodos X]
para todo resX:
cpd_Xi j= CPD(resX[e],2,[[probabNoBomba,probabBomba]])

modelo.añadirCPD(cpd_Xi j)
```

Para las CPDs de la variable Y, recorremos la lista de casillas, obtenemos los vecinos de la casilla que se está iterando en ese momento y se hacen permutaciones sobre el número de vecinos y se calcula así la probabilidad del número de minas en casillas vecinas.

```
    resY = [s para todos los nodos_nombrados si 'Y'
contenido en s]
        para todos los valores l de resY:
            i
            j
                                        (i, j)

            esA

            para todos los valores v de vecinos:
                probabilidades_calculadas = []
                para todos los valores a de resA:
                    lista = resA[a]
                    contador = counterPermutations(lista)
                    si contador es v:
                        probabilidades_calculadas.añadir(1)
                    si no:
                        probabilidades_calculadas.añadir(0)

            probabilidades_unidas.añadir(probabilidades_calcul
adas)

            cpd_Y = CPD(casilla l a iterar de resY,
tamaño(vecinos)+1, probabilidades_unidas, vecinos,
2*tamaño(vecinos))

            modelo.añadirCPD(cpd_Y)
```

V. SISTEMA DE SUGERENCIA DE CASILLAS

Tras la construcción de nuestro modelo bayesiano, diseñamos un algoritmo que realice consultas a éste para obtener, después de un movimiento aleatorio en el tablero, una sugerencia de próximo movimiento. Siendo esta última una de las casillas no descubiertas colindantes a una isla con menor probabilidad de contener una bomba:

El primer movimiento se realiza de manera aleatoria generando cada uno de sus indices de la siguiente forma.

$$i = randInt(0, w-1)$$
$$j = randInt(0, w-1)$$

Tras saber cual es la casilla seleccionada para hacer *click*, y previo a realizar el movimiento *click* sobre esa casilla, se ha elaborado un algoritmo que despeje una zona como mínimo de nueve casillas desplazando las minas encontradas a las esquinas o a los laterales del tablero. Esta acción se encuentra contemplada en el código decompilado del buscaminas original[3], por lo que no se esta llevando a cabo ninguna acción "ilegal". El algoritmo contempla lo siguiente:

```
definir mover_minas_alrededor(i, j):
    si tablero.mapaDeMinas[j, i] == 1:
        mover_mina_a_esquina(j, i)
        vecinos = vecinos_de_posicion(i, j)
        para todos los vecinos en vecinos:
            i = posición i de vecino
            j = posición j de vecino
            si tablero.mapaDeMinas[j, i] == 1:
                mover_mina_a_esquina(j, i)

definir mover_mina_a_esquina(x, i):
    si tablero.mapaDeMinas[w-1, h-1] == 0:
        tablero.mapaDeMinas[j, i]
        tablero.mapaDeMinas[w-1, h-1] = 1
    si tablero.mapaDeMinas[0, 0] == 0:
        tablero.mapaDeMinas[x, i] = 0
        tablero.mapaDeMinas[0, 0] = 1
    si tablero.mapaDeMinas[w-1, 0] == 0:
        tablero.mapaDeMinas[x, i] = 0
```

```
        tablero.mapaDeMinas[w-1, 0] = 1
    si tablero.mapaDeMinas[0, h-1] == 0:
        tablero.mapaDeMinas[x ,i] = 0
        tablero.mapaDeMinas[0, h-1] = 1
```

Una vez colocadas las minas en sus nuevas posiciones, se procede a hacer despejar una casilla por medio del movimiento *click* con la orden:

```
game.play_move("click", i,  j)
```

Tras este movimiento, el estado del tablero cambia y se obtienen nuevas evidencias. Esta nueva información se usará para calcular el siguiente movimiento de la forma que sigue:

Volvemos a iterar el tablero del juego para observar las evidencias descubiertas tras el movimiento realizado:

```
para todas las i en anchura del tablero:
    para todas las j en anchura del tablero:
        estado_casilla = información del mapa en (i, j)
        si el estado_casilla está comprendido entre 0 y 8:
            añadir evidencia Xij = 0
            añadir evidencia Yij = estado_casilla
        si estado_casilla es 0:
            añadir evidencia Xij = 0
            añadir evidencia Yij = 0
        si estado_casilla es 11:
            añade Yij a lista de evidencias
            añade Xij a casillas no descubiertas
        si estado_casilla es 9:
            añadir evidencia Xij = 1
```

Ahora hemos de obtener aquellas casillas que son susceptibles a ser consultadas por su probabilidad de contener mina, estas son; aquellas casillas no descubiertas que sean vecinas de casillas ya descubierta, es decir, obtenemos todas las casillas colindantes a una isla. La selección de estas casillas en concreto se debe a que dichas casillas son las que todavía están sin descubrir, pero son las que tienen más información útil para poder hacer una consulta veraz y rápida.

El sistema iterará cada casilla de ese conjunto y realizará las siguientes operaciones sobre ellas:

a. Se eliminarán todas aquellas casillas que no sean ni de consulta ni de evidencia y que además no sean hojas del árbol. Cabe destacar que las casillas que se eliminan varían en cada iteración del conjunto que estamos recorriendo.

b. Una vez hecho esto, se hará uso del método *VariableElimination* para llevar a cabo la consulta de dicha casilla.

c. Para llevar a cabo la consulta, se genera un nuevo modelo específico para la eliminación de variables al que se le realizará una consulta pasándole como parámetros la casilla que estamos consultando y una lista con las evidencias relacionadas.

```
Modelo para eliminación = pgmi.VariableElimination(modelo)
consulta = Cosulta a Modelo para eliminación dado una casilla sin descubrir y una lista de evidencias.
```

d. Una vez obtenido el valor correspondiente a la probabilidad de mina existente en esa casilla y con el objetivo de optimizar el algoritmo, se pasa por un filtro establecido previamente (la elección del filtro se detalla en la sección "aumentando la eficiencia") en el que se decide si esa casilla contiene una bomba o no, el filtro sería el siguiente:

$$mina(X_{i,j}) \begin{cases} 1 & \text{si } 1 > P(X_{i,j}) > 0.790 \\ P(X_{i,j}) & \text{en otro caso} \end{cases}$$

$$\neg mina(X_{i,j}) \begin{cases} 1 & \text{si } 1 > P(X_{i,j}) > 0.90 \\ P(X_{i,j}) & \text{en otro caso} \end{cases}$$

Como se puede observar a aquí, si una casilla tiene una probabilidad mayor *0,90* de no contener una bomba, se romperá el bucle y se realizará un movimiento *click* sobre esa casilla pasando a la siguiente consulta. Del mismo modo, si esa casilla tiene una probabilidad mayor que *0'790* de contener una mina, se pasa a la siguiente iteración del bucle y se marca esa casilla como mina con el movimiento *Flag*.

e. Por último, si ninguno de los elementos iterados ha entrado en la condición de *no mina*, de entre todos ellos se seleccionará para hacer *click* aquel que mayor probabilidad tenga de no contener una mina. A igualdad de probabilidades, se seleccionará el primero.

```
lista_probabilidades = []
para todas las consultas realizadas:
    Añadir resultado de la consulta a lista_probabilidades
probabiliades_no_mina = Primeros elementos de la lista de probabilidades.
sugerencia = el índice con el máximo valor de la lista de probabilidades de no contener una mina
```

VI. AUMENTANDO LA EFICIENCIA

En el ámbito del problema planteado, se entiende por eficiente aquella implementación que sea capaz de resolver el tablero de mayor tamaño 8x8 con 25 minas

en menos de una hora, haciendo uso único de un núcleo de la CPU de como mínimo *3Ghz* de velocidad de reloj y en ningún momento la GPU.

Teniendo en cuenta esta premisa y dado la implementación base de la librería utilizada, llevar a cabo la acción de resolución en el tiempo anteriormente descrito se convierte en una tarea imposible ya que en el 100% de los casos, la ingente cantidad de memoria utilizada provoca un desbordamiento apenas varios minutos después de comenzar.

Este desbordamiento se debe principalmente a la falta de eficiencia de la librería *Pgmpy* que lejos de intentar optimizar las variables relevantes en cada consulta, intenta realizar el algoritmo de eliminación de variables en todos los casos con todas las variables del grafo, tenga o no tenga evidencias de ellas, sean o no sean hojas del grafo, por ejemplo:

Se tiene un tablero de 5X5 con 5 minas sin ningún tipo de evidencia, se hace una consulta sobre la probabilidad de una casilla aleatoria. En este momento el algoritmo tiene en cuenta todas las variables del grafo, es decir todas las *X* y todas las *Y* para dar una respuesta que en teoría debería de ser instantánea. Dicha respuesta se encuentra expuesta más arriba y se calcula con el número de minas y el número de casillas del tablero. Lo que debería tardar unos pocos milisegundos, puede llegar a tardar horas por no haber eliminado todas las variables irrelevantes en el dominio del problema.

Para llevar a cabo el subsanado de este problema, se han puesto en funcionamiento una serie de mecanismos que mejoran enormemente la eficiencia del algoritmo:

a. En primer lugar, se exime al algoritmo de tener que calcular las probabilidades iniciales de las X haciendo uso de la fórmula que involucra el número de casillas y el número de minas.

b. En segundo lugar, se implementa un método que en el primer movimiento del juego, en el *100%* de los casos genere una isla de al menos 9 elementos tal y como estaba implementando en "El Buscaminas" original. Esto hace que el algoritmo siempre tenga un número considerable de evidencias en el primer movimiento

c. En tercer lugar, para deducir el siguiente movimiento del juego, se hace la consulta a las casillas colindantes a una isla y no a todas las casillas ocultas del tablero, lo que reduce en un gran número la cantidad de consultas a realizar

d. En cuarto lugar, en cada una de las iteraciones de las casillas consultadas, se crea un submodelo resultante de eliminar del modelo original aquellas variables irrelevantes para la consulta, es decir, todas las variables que no sean ni de consulta ni de evidencia, y ademas sean hojas del árbol. Lo que resulta de un modelo mucho más simple y menos complejo.

e. En quinto lugar, se establece una cota por la cual se interpreta la existencia de una mina en una casilla, esto es; Tras realizar la consulta de la probabilidad de la existencia de mina en una casilla se comprueba si el número resultante está contenido entre la cota:

Si la probabilidad de existencia de mina supera dicha cota, el sistema marcará automáticamente dicha casilla como mina y pasará a la siguiente sin que esta se convierta en candidata sobre la que hacer *click* en el siguiente movimiento. Del mismo modo, si el número obtenido se encuentra por debajo de la cota, el sistema interpretará automáticamente que en esa casilla no existe una mina y hará *click* inmediatamente, rompiendo el bucle y pasando al siguiente movimiento.

La elección de la cota no es algo trivial ya que una cota de la inexistencia de mina demasiado baja provocaría que la probabilidad de hacer *click* por error aumentara. Del mismo modo, la existencia de mina demasiado baja podría provocar que se marcara como mina una casilla que no lo es.

Tras muchas comprobaciones, hemos llegado a la conclusión de que una cota como la descrita anteriormente se encuentra en el punto medio entre lo óptimo y lo permisible a la hora de proporcionar resultados veraces.

Estas mejoras descritas hacen que sea posible la ejecución de un tablero de 10X10 con 25 minas en una hora, cosa que antes era imposible resolver dado su tamaño.

VII. PRUEBAS DE RENDIMIENTO

Para las pruebas de rendimiento de la red bayesiana, se ha tenido en cuenta el tiempo que tarda el algoritmo en sugerir una casilla y resolver el tablero satisfactoriamente, frente al tamaño del tablero y el número de minas contenidas en éste.

Tras realizar varias pruebas con los tamaños requeridos se han obtenido los siguientes datos de duración:

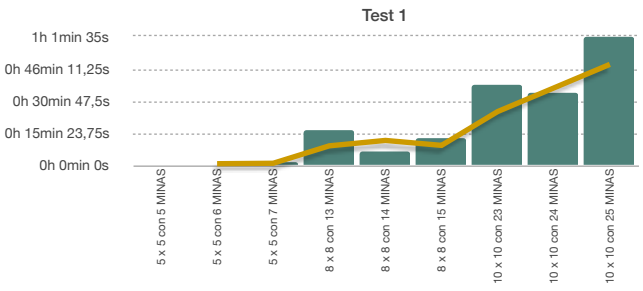

Figura 3: Gráfica que relaciona cada uno de los tamaños en el test1 con la duración de la resolución completa del tablero por parte del algoritmo junto con la recta promedio.

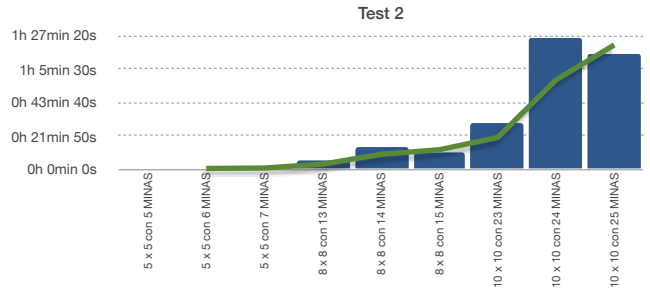

Figura 4: Gráfica que relaciona cada uno de los tamaños en el test2 con la duración de la resolución completa del tablero por parte del algoritmo junto con la recta promedio.

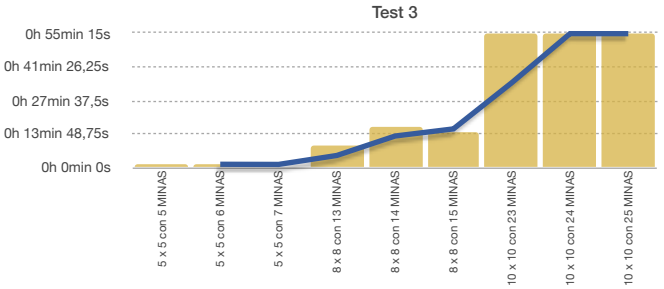

Figura 5: Gráfica que relaciona cada uno de los tamaños en el test3 con la duración de la resolución completa del tablero por parte del algoritmo junto con la recta promedio.

Se aprecia en las tres gráficas que con tamaños de tablero pequeños (5x5), la complejidad de la red es pequeña, ya que hay una gran diferencia en la proporción número de evidencias - número de casillas a descubrir, tras un *click* en el tablero se desvela gran parte del contenido de éste. La probabilidad inicial de encontrar minas con 5, 6 y 7 minas sería de *0'2*, *0'24* y *0'28* respectivamente. El tiempo de resolución hasta la victoria por parte de nuestro algoritmo de inferencia de es de 1 minuto.

Con tableros de tamaño 8x8, aumenta la complejidad al haber un mayor número de casillas en el tablero, pero la proporción de bomba se mantiene relativamente ya que la probabilidad inicial de encontrar mina con 13, 14 y 15 minas sería de *0'20*, *0'218*, *0'234* respectivamente. El tiempo de resolución hasta la victoria es de unos 10 minutos.

Para los tableros de mayor tamaño (10x10), la probabilidad inicial de encontrar una mina para tableros con 23, 24 y 25 minas son de *0'23*, *0'24*, *0'25* respectivamente, son cifras similares a las anteriores, pero con este tamaño de tablero, el tiempo de resolución hasta la victoria por parte de nuestro algoritmo de inferencia llega a rondar la hora.

Sin embargo, las mediciones fluctúan mucho entre tableros del mismo tamaño con igual número de bombas dependiendo del caso del test. Esto se debe a que influye mucho el azar dentro de la complejidad final y del tiempo de resolución total del tablero, ya que el número de evidencias tras el primer *click* condiciona mucho a nuestro tablero.

Un primer *click* con pocas evidencias haría que nuestro algoritmo se demore más en darnos una respuesta segura sobre el siguiente movimiento, o en el caso contrario, un primer *click* en el que se genere una isla con muchas casillas descubiertas y mucha información sobre el número de minas en las casillas colindantes haría que nuestro algoritmo trabaje y proporciones respuestas más veraces y rápidas.

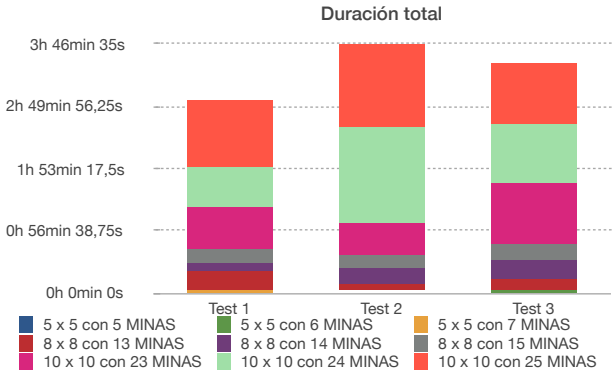

Figura 6: Gráfica que engloba la duración de cada una de las pruebas dentro de los tests realizados.

Si calculamos el promedio de tiempo de resolución de tableros hasta la victoria por parte de nuestra red con todos los tamaños usados anteriormente y las variaciones obtenidas en las 3 pruebas, tenemos un tiempo promedio de 23 minutos.

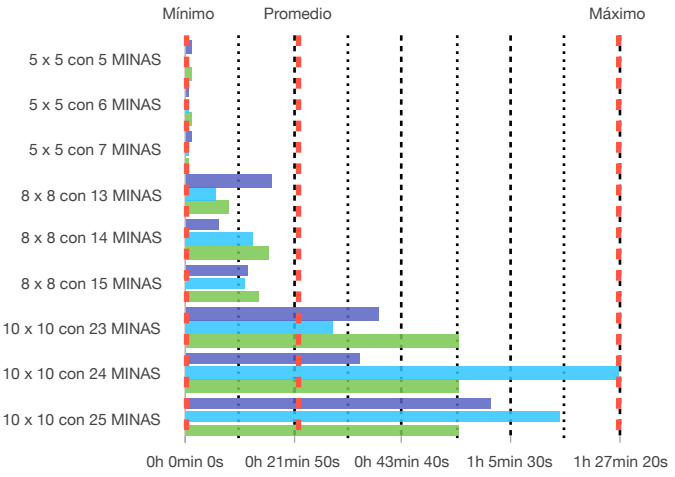

Figura 7: Gráfica que relaciona cada uno de los tamaños con la duración de la resolución completa del tablero por parte del algoritmo y el valor mínimo de tiempo, el promedio y el máximo.

VIII. CONCLUSIONES

Como se ha podido observar, el juego "El Buscaminas" es un ejemplo útil de como se puede utilizar una red bayesiana para resolver un problema complejo.

Tras la búsqueda de cómo poder hacer una consulta eficiente a la red, y después de haber realizado un gran número de las simplificaciones posibles a ésta, se consigue reducir el problema a un tamaño medianamente aceptable para una máquina contemporánea, ya

que *Pgmpy* realiza un cálculo demasiado exacto sobre los nodos de la red y se producen desbordamientos de memoria durante los cálculos, o simplemente por el hecho de tener un grafo muy grande con mucha información que ha de ser reducido para hacer una consulta sobre éste.

Han sido clave las utilidades creadas para crear islas de casillas tras el primer *click* del juego, ya que ayuda a reorganizar la información sobre las minas en el tablero y que el jugador, o en este caso nuestro algoritmo, tenga mucha más información al principio del juego para poder resolverlo exitosamente, además ha sido primordial el hecho de crear un submodelo en cada consulta para poder reducir grandes redes de inferencia a pequeños grafos con la información relativa a la consulta de la casilla que queremos consultar en ese momento, es decir, deshacernos de la información no relevante para la operación actual. También ha aportado velocidad a nuestro algoritmo la implementación de la cota con la se permite, si no se tiene la seguridad absoluta sobre la existencia de bomba en las casillas que se está consultando, que nuestro algoritmo marque como bomba o haga *click* sobre esa casilla si los valores de probabilidad están dentro de esa cota.

Sin estas mejoras, no podría haberse resuelto el problema dado su tamaño y complejidad.

Para comprobar empíricamente todo lo descrito en este documento, se puede acceder a el repositorio de la red bayesiana[5] y ejecutar los archivos Python *nextStepMinesweeperSuggestor.py,* que dados un tamaño de tablero y un número de minas resuelve la partida paso a paso pidiendo al usuario en todo momento el consentimiento para realizar otro paso dentro del juego, o el archivo *testRepeater.py* que ejecuta el algoritmo que contiene la red de inferencia bayesiana que hemos construido de forma recursiva hasta la finalización de todos los casos de prueba descritos anteriormente.

REFERENCIAS

1. duguyue100, “A python Minesweeper with interfaces for RL.,” Github (https://github.com/duguyue100/minesweeper), Diciembre 2016.
2. “NumPy - fundamental package for scientific computing with Python”, (http://www.numpy.org)
3. Código decompilado del buscaminas original (http://www.techuser.net/files/code/minesweeper/stepsquare.asm)
4. “Pgmpy - python library for working with Probabilistic Graphical Models”, (https://github.com/pgmpy/pgmpy#installation)
5. M.A. Antolín y D. De la Torre, “Sistema de resolución del juego El Buscaminas mediante el uso de redes bayesianas ” Github (https://github.com/Pacoliva/MinesweeperBayesianNetwork.git), Junio 2018.
6. Marta Vomlelová y Jirí Vomlel “Applying Bayesian networks in the game of Minesweeper”, Czech Republic (http://staff.utia.cas.cz/vomlel/vomlel-tichavsky-pgm2014.pdf), 2012
7. J.L. Ruiz Reina, F.J. Martín Mateos et al. Redes bayesianas. Dpto. Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla, 2017-2018