Buscapython

Álvaro Calle González
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
Correo electrónico UVUS y de contacto: alvcalgon@alum.us.es y alvarogoles@gmail.com

Antonio Nolé Anguita
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
Correo electrónico UVUS y de contacto: antnolang@alum.us.es y anolanguita@gmail.com

***Resumen*— El trabajo está basado en el juego Buscaminas, para conocer de qué trata el juego, consultar el documento aportado por el profesorado [1]. El objetivo principal del proyecto es implementar un método que sea capaz de hallar las casillas con una probabilidad más alta de no contener una mina y sugerirlas como siguiente paso a dar en el juego. Para ello, se debe usar una red bayesiana [2].**

En cuanto a las conclusiones, podemos decir que resolver la tarea planteada ha supuesto invertir mucho tiempo y esfuerzo dada la complejidad del trabajo. Adicionalmente, realizar este proyecto nos ha permitido ampliar nuestros conocimientos en diversos campos como hacer uso de un framework para el desarrollo de la interfaz gráfica: qt5 [4] y [5] o aplicar correctamente el algoritmo "Flood Fill" [3]. Por último, esta actividad nos ha permitido mejorar la gestión y organización del trabajo con respecto a la organización y gestión llevadas a cabo para tareas asociadas a otras asignaturas de la carrera como puede ser los entregables de DP, los proyecto de IISSI y AISS, etc. Finalmente, dada la ineficiencia del algoritmo de eliminación de variables, es necesario realizar una adaptación de dicho algoritmo.

Palabras claves: Buscaminas, redes bayesianas, framework, Flood Fill, eliminación de variables, inferencia exacta, rede de creencia.

I. Introducción

El proyecto presente trata sobre el juego Buscaminas. Dado un tablero de n filas y m columnas, existe un número determinado de minas. El jugador debe de descubrir todas las casillas que no poseen minas para ganar la partida. En el momento en el que se descubra una casilla que contenga mina, se pierde la partida. En el caso de que hagamos clic en una casilla que no contenga mina, obtenemos información sobre las casillas vecinas. Para más información acerca del juego, consultar el siguiente recurso: [1]. Es posible jugar al Buscaminas a través del siguiente enlace: [6].

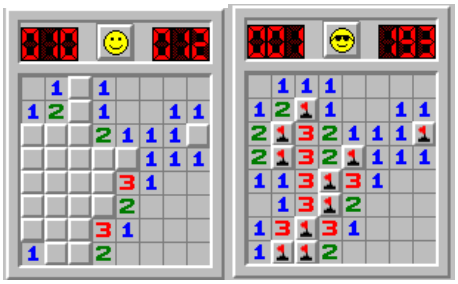

Ilustración 1. 2 capturas del juego.

Para el desarrollo del trabajo, haremos uso de las redes bayesianas. Previamente a la ejecución de la tarea, fuimos instruidos en las redes bayesianas por parte del profesorado mediante la explicación de los conceptos teóricos [2], problemas para poner en práctica la teoría [7] y una práctica [8].

El propósito de este proyecto es obtener la especificación de un método que determine las casillas con mayor probabilidad de no poseer una mina y sugerirlas como siguiente paso a tomar en la partida. A la hora de resolver el problema propuesto, se podría implementar tanto un algoritmo de inferencia exacta como es el algoritmo de eliminación de variables o un algoritmo de inferencia aproximada como muestreo (en inglés, "sampling"). Finalmente, se ha optado por aplicar el algoritmo de eliminación de variables ya que es uno de los requisitos de la actividad a realizar.

El documento se divide en 5 secciones: una introducción donde se detalla el trabajo propuesto. Seguido de la sección preliminares, en la cual se comenta los métodos y técnicas empleadas para la realización de la propuesta de trabajo y proyectos anteriores relacionados que puedan aportar nuevas ideas y/o enfoques distintos a la hora de resolver la tarea. La sección 2 corresponde a la metodología, en cuyo contenido se especifica minuciosamente cómo se ha definido la solución. La siguiente sección pertenece a los resultados obtenidos durante la experimentación. Por último, una sección con las conclusiones generales obtenidas.

II. Preliminares

A. *Métodos empleados*

Para la resolución del problema planteado se ha requerido el uso de las redes bayesianas (en inglés, bayesian network). También conocidas como redes de creencia (belief networks), pertenecen a la familia de los modelos gráficos de probabilidad. Esta estructura gráfica es usada para representar conocimiento sobre un dominio incierto. Las redes bayesianas usan grafos acíclicos dirigidos (DAG), en el que cada nodo representa a una variable aleatoria mientras que las aristas entre nodos representan las dependencias probabilísticas entre las variables aleatorias correspondientes. Si desea saber más acerca de los grafos acíclicos dirigidos y las redes bayesianas, consultar los siguientes enlaces [10] y [11] respectivamente. El sistema de sugerencia de casillas que debemos construir debe usar un algoritmo de inferencia exacta en redes bayesianas. En particular, el algoritmo eliminación de variables. Este algoritmo realiza 2 tipos de operaciones: multiplicación y agrupación. Para hallar la probabilidad de que una variable aleatoria tome un valor concreto (consulta) dada una serie de evidencias usando el algoritmo anteriormente nombrado, hay que seguir los siguientes pasos: primeramente, se calcula el factor inicial para cada variable que interviene en el proceso. Una vez obtenidos los factores iniciales, se van "eliminando" (realizar el sumatorio de la correspondiente fórmula) las variables que no son ni de consulta ni de evidencia. Cuando se eliminan todas las variables (que no sean ni de evidencia ni de consulta), sólo quedarán factores dependientes de la variable de consulta, que tendrán que ser multiplicados y finalmente normalizados. Para conocer todos los detalles del algoritmo, visitar los siguientes recursos: [2] y [12].

B. *Trabajo Relacionado*

Para familiarizarnos aún más con la tarea a completar y para conocer cómo han planteado otros autores el desarrollo del proyecto, se han consultado artículos tales como el nombrado a continuación: "Applying bayesian network in the game of minesweeper" [13]. Este documento realiza, primeramente, una breve introducción sobre las redes bayesianas. Luego, explica cómo construir la red bayesiana. También aporta una serie de recomendaciones para mejorar la eficiencia de la red bayesiana.

A su vez, este artículo hace referencia a otros documentos igual de interesantes y útiles. En especial, el documento denominado "Tensor Rank-one decomposition of probability tables" [14]. Este documento trata sobre un mecanismo que actúa sobre las tablas de probabilidad condicional y que ayuda a reducir el coste de la complejidad computacional de la inferencia probabilística del modelo de la red bayesiana.

III. Metodología

La red bayesiana consta de 2 variables distintas por cada casilla (i, j) del tablero: Xi,j e Yi,j. La variable aleatoria **X** asocia la existencia de una mina en una casilla. Por tanto, es una variable booleana. La probabilidad de que una casilla contenga una mina o no depende del tamaño del tablero y del número de minas existentes en dicho tablero. Luego, está la variable aleatoria **Y**, que indica el número de minas que hay en las casillas vecinas. Dependiendo de la casilla, la variable puede tener un rango de valores distinto. La probabilidad de que una de estas variables tome un valor concreto (estado) dependerá del número de **variables colindantes X** que contengan una mina. Es decir, cada una de estas variables depende de las variables X colindantes. Las tablas de probabilidad condicional de las variables X e Y se definen en el documento [13].

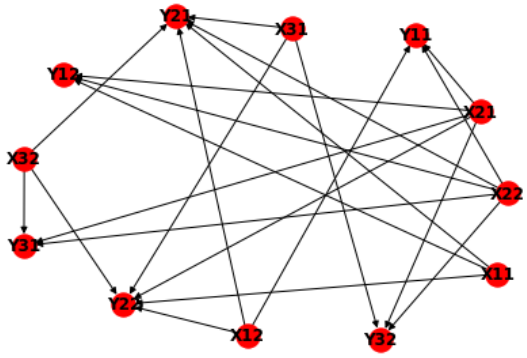

Ilustración 2. DAG correspondiente a un tablero de 3x2

A continuación, mostramos a modo de ejemplo 2 capturas de tablas de probabilidad condicional asociadas a las variables aleatorias X e Y respectivamente.

X21_0	0.666667
X21_1	0.333333

Ilustración 3. Tabla de probabilidad de la variable aleatoria X.

X21	X21_0	X21_0	X21_0	X21_0	X21_1	X21_1	X21_1	X21_1
X22	X22_0	X22_0	X22_1	X22_1	X22_0	X22_0	X22_1	X22_1
X12	X12_0	X12_1	X12_0	X12_1	X12_0	X12_1	X12_0	X12_1
Y11_0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Y11_1	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0
Y11_2	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0
Y11_3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Ilustración 4. Tabla de probabilidad de la variable aleatoria Y

Para completar el primer objetivo específico, es decir, implementar un método que construya la red bayesiana asociada a un tablero, dividimos la actividad en 2 partes.

Por un lado, definimos una serie de funciones que crearan el DAG asociado a la red bayesiana y la propia red bayesiana.

Estas funciones se encuentran en el script bayesian_network.py. Todas las funciones están diseñadas de forma parametrizada para que se pueda crear un tablero con unas dimensiones y número de minas cualquieras. La función denominada "**generate_DAG**" se encarga de crear el DAG.

Función generate_DAG
Entrada:
- height, width: int

Salida:
- Grafo que modela la red bayesiana.

Algoritmo
1. modelo_buscaminas = pgmm.BayesianModel()
2. (n, m) = (height, width)
3 Para cada casilla (i, j) del tablero, añadir las aristas entre la variable Yi,j y las variables Xk,l colindante, es decir, pa(Yi,j)
4. Devolver modelo_buscaminas

Luego, la función "**generate_BN**" define la red bayesiana. Para ello, crea una instancia de un grafo invocando al método "generate_DAG" y, luego, asocia a cada nodo una distribución de probabilidad condicional mediante la invocación al procedimiento "**createCPDs**".

Los procedimientos **create_y_CPD** y **create_x_CPD** se encargan de añadir en el grafo la tabla de probabilidad condicional del nodo que se recibe por parámetro.

Para implementar todas estas funciones, ha sido necesario hacer uso de los siguientes módulos: **pgmpy.models** y **pgmpy.factors.discrete**. El módulo pgmy.models es útil para definir los modelos gráficos de probabilidad, nuestro caso, el modelo de la red bayesiana. El otro módulo, pgmpy.factors.discrete, permite crear las tablas de probabilidad condicional y los factores asociados a dichas tablas de probabilidad. La documentación de ambos módulos se encuentra en los siguientes enlaces: [16] y [17] respectivamente.

Procedimiento createCPDs

Entrada:
- DAG, height, width, num_of_mines.

Salida:
- No devuelve nada

Algoritmo:
1. Para cada nodo del grafo DAG hacer
    a. Si nodo es una variable Y
        i. create_y_CPD(nodo, DAG)
    b. Si nodo es una variable X
        i. create_x_CPD(nodo, DAG, height, width, num_of_mines)

Por otro lado, modelamos los elementos tablero (en inglés, "board") y casilla (en inglés, "square") como 2 clases relacionadas entre sí. La clase Square se encuentra en el archivo square.py y la clase Board en board.py. La clase Square es bastante simple, posee 4 atributos:

- is_mine: indica si la casilla contiene una mina.
- neighbor_mines: número de minas que tienen las casillas vecinas.
- is_hidden: si la casilla está oculta o no.
- flagged: indica si la casilla está marcada o no.

Esta clase cuenta un constructor, una serie de "setters" para actualizar sus atributos y un método relacionado con la GUI. La clase Board tiene los siguientes atributos:

- height: número de filas del tablero.
- width: número de columnas del tablero.
- num_of_mines: número de minas que almacena el tablero.
- evidences: diccionario donde se añade, para cada variable de la casilla del tablero cuyo valor conocemos, su valor de evidencia.
- suggested_pos: es la posición de la casilla sugerida actual por el sistema. Al inicio, su valor es (0, 0).
- variable_elimination: es una instancia del algoritmo eliminación de variables.

La clase tiene implementados una serie de funciones relacionadas con la interfaz gráfica como son **initUi, init_squares, get_square, reveal_all_board, handle_left_click, display_square, handle_flag, new_game, conf_dialog, rules_dialog, y suggest_dialog**. Además, la clase integra una serie de funciones asociadas con las características y restricciones propias del juego como son la funciones y procedimientos: **place_mines, get_position, update_neighbors, invalid_position, is_end_game, reveal, reveal_information y add_evidence**.

Posteriormente, para realizar el segundo objetivo específico (implementar mecanismo de sugerencia de casillas), fue necesario añadir una serie de funciones extra tanto en la clase Board como en el script de bayesian_network.py. Estas funciones son: **calcule_prob_X, suggest_next_square y get_hidden_squares**. Estas 2 últimas funciones pertenecen a la clase Board.

El método calcule_prob_X calcula, usando el algoritmo de eliminación de variables, la probabilidad de que la casilla (i, j) no contenga una mina dada una serie de evidencias. Luego, el método suggest_next_square busca la casilla oculta que tenga una mayor probabilidad de no contener una mina y la devuelve. Para ello, invoca al método calcule_prob_X tantas veces como casillas ocultas haya y compara las probabilidades de todas esas casillas devolviendo la mayor.

A continuación, el mecanismo que desarrolla una partida completa del juego se ve reflejado en los métodos anteriormente nombrados. De manera que el usuario va haciendo clic continuamente en las casillas que el sistema le va sugiriendo hasta que, o bien consigue ganar o perder. Hay una excepción, y es que el usuario debe elegir, al empezar la partida, qué casilla descubre puesto que en el primer paso, el sistema no sugiere ninguna casilla puesto que todas los campos del tablero albergan la misma probabilidad de no contener una mina. Una vez iniciada la partida y descubierta una casilla, el sistema interactúa con el usuario sugiriendo casillas y descubriéndolas.

Como apunte final, hemos seguido a la hora de codificar en Python un estilo de código cuya referencia es [20]. El propósito de seguir esta guía de estilo es facilitar la lectura del código por parte de aquellas personas que tengan que leerlo ya sea para una posterior fase de mantenimiento o bien para su evaluación.

Mejoras

Existe otra forma de completar una partida completa del juego sin que ningún usuario intervenga. Con el procedimiento **play_game** de la clase Board, el sistema puede sugerir y hacer clic en las casillas ocultas por sí solo.

Otra mejora es que hemos implementado una interfaz gráfica de usuario (GUI) para que el usuario pueda interactuar con el sistema. Para desarrollar dicha interfaz, como hemos dicho anteriormente, se ha usado el **framework qt5** cuya documentación se puede visitar a través de los siguientes enlaces: [4] y [5]. El código asociado a este framework se encuentra en los archivos board.py y square.py.

Por otra parte, cabe destacar que Qt5 es un framework desarrollado para C++ [19]. Para poder integrarlo con python ha sido necesario usar un componente extra llamado PyQt5 [15]. Este componente es lo que se llama un "binding" de Qt5, y su objetivo es hacer de puente entre Qt5 y el lenguaje de programación Python [21].

Finalmente, dada la enorme ineficiencia inicial del algoritmo de eliminación de variables, ha sido necesario especializar dicho algoritmo del paquete pgmpy [9] para nuestro problema en concreto. Estas modificaciones serán detalladas en el siguiente apartado junto con su análisis de rendimiento.

Ilustración 5. Capturas de la interfaz gráfica.

IV. Resultados

Analizaremos los resultados desde 2 perspectivas distintas. Por un lado, valoraremos la capacidad que tiene el sistema para ganar una partida frente a las derrotas. Por otro lado, examinaremos la eficiencia del sistema.

Empecemos evaluando la destreza que presenta el sistema a la hora de obtener una victoria en una partida. Los experimentos que se han realizado se han hecho una vez aplicadas las modificaciones sobre el algoritmo de eliminación de variables. De estas modificaciones hablaremos en detalle cuando tratemos los resultados desde el enfoque del rendimiento y la eficiencia.

Para la realización de los test, se ha usado la función denominada **average_success** que se encuentra en la carpeta Buscapython/tests/test.py.

Dicho esto, podemos ver reflejados los resultados del experimento en la **tabla 1**. La primera conclusión que podemos sacar es que el sistema no es ni mucho infalible: de las 45 pruebas realizadas, 16 victorias y 29 derrotas. Es decir, la probabilidad media de ganar una partida es del 0.3556.

Huelga decir que las derrotas producidas se dan, en gran medida, en los primeros pasos (y sobre todo en el primero), ya que en estos primeros movimientos el mecanismo de sugerencia no tiene suficientes evidencias como para sugerir una buena casilla con una alta probabilidad de no contener una mina (en los primeros movimientos el algoritmo va a ciegas).

Un ejemplo claro podría ser el tablero ilustrado en la siguiente imagen. Tenemos la evidencia de una casilla, pero no es suficiente para sugerir de manera fiable una casilla sin mina.

Ilustración 6. Ausencia de evidencias

Luego, en la **tabla 2** podemos observar cómo la probabilidad según la configuración sea una u otra cambia. Se puede ver cómo, a medida que se juega en un tablero con unas dimensiones mayores y con mayor número de minas, la probabilidad de ganar una partida disminuye salvo en excepciones.

Config	Pruebas				
	1	2	3	4	5
5x5x5	Game over	Victory	Game over	Victory	Game over
5x5x6	Victory	Game over	Victory	Game over	Victory
5x5x7	Game over	Game over	Victory	Game over	Game over
8x8x13	Victory	Victory	Victory	Victory	Game over
8x8x14	Game over	Game over	Game over	Game over	Victory
8x8x15	Game over	Game over	Game over	Game over	Victory
10x10x20	Victory	Game over	Victory	Game over	Game over
10x10x22	Game over	Victory	Game over	Game over	Game over
10x10x25	Game over	Victory	Game over	Game over	Game over

Table 1. Resultados

Configuración	Probabilidad de ganar una partida
5x5x5	0.4
5x5x6	0.6
5x5x7	0.2
8x8x13	0.8
8x8x14	0.2
8x8x15	0.2
10x10x20	0.4
10x10x22	0.2
10x10x25	0.2

Table 2. Resultados

En cuanto a la eficiencia y rendimiento, el objetivo es analizar el tiempo que necesita el mecanismo para resolver correctamente una partida de principio a fin (método Board::play_game), sin tener en cuenta las veces en las que dicho mecanismo sugiere una casilla con una mina.

El algoritmo de eliminación de variables usado ("variable_elimination.py") es un clon del algoritmo del paquete pgmpy [9], al que se le ha realizado varias modificaciones para mejorar su eficiencia. Dichas modificaciones se encuentran claramente marcadas a través de comentarios precedidos por **"# --MODIFICACIÓN"**.

A continuación, analizaremos y justificaremos cada una de las 3 modificaciones y el impacto que han tenido en el rendimiento.

Para evaluar el rendimiento, consideraremos un tablero de 5x5 con 5 minas. El tablero será resuelto veinte veces (cada vez con una disposición de minas distinta) y calcularemos el tiempo de ejecución medio de cada uno de ellos. (solo se tendrán en cuenta los tiempos de ejecución en los que el sistema resuelve satisfactoriamente el tablero).

El código del test corresponde en el método **average_time** y se puede encontrar en el siguiente fichero Buscapython/tests/test.py.

IMPORTANTE: Los tiempos calculados que se muestran en la tabla 3 son relativos, ya que pueden variar dependiendo del ordenador en el que se ejecuten los tests y de los procesos activos en el momento de ejecución de los tests. Aún así, es útil para comprobar de forma orientativa si las modificaciones tienen un impacto significativo.

En primer lugar, evaluamos el rendimiento sin modificaciones, con el algoritmo de eliminación de variables tal y cómo se encuentra publicado en el paquete pgmpy.

Para este caso el test no es capaz de ejecutar ni una sola iteración para ninguno de los tres tableros, ya que el ordenador en el que se ejecuta se queda sin memoria RAM (**MemoryError**).

Tras probar varias veces de manera experimental, llegamos a la conclusión que el tablero más grande con el que podemos trabajar es un tablero de 2x3.

Para la **primera modificación**, en el concepto en el que nos basamos es el siguiente: "Toda variable que no sea antecesor (en la red) de alguna de las variables de consulta o de evidencia, es irrelevante para la consulta." (diapositiva 78 del tema de teoría, [2]). (Antecesor de una variable también hace referencia a los padres de los padres).

Para simplificar consideraremos también variables relevantes las propias variables de consultas y las variables de evidencia. Esto no tendrá ningún impacto negativo en el algoritmo.

La parte donde el algoritmo de eliminación de variables resulta tan ineficiente es a la hora de operar con los factores de los nodos, y esto empeora de forma exponencial cuando el número de nodos aumenta.

Gracias a esta modificación ganamos mucha eficiencia, ya que al saber cuáles van a ser las variables relevantes, podemos saber a priori cuáles van a ser los factores que vamos a necesitar realmente (pudiendo descartar desde el primer momento los factores que no sean relevantes) y las variables que realmente nos resulta útil eliminar para resolver la consulta dada.

Debido a la estructura concreta de la red bayesiana con la que trabajamos, el cálculo de las variables relevantes es bastante simple.

En la red tenemos dos tipos de variables, variables X y variables Y. Las variables Y solo tienen como padres variables de tipo X. Por otro lado, las variables X no tienen padres.

Teniendo en cuenta que las variables de consulta son siempre de tipo X (probabilidad de que haya una mina en una casilla), no es necesario calcular los antecesores de las variables de consulta ya que sabemos de antemano que no existe ninguno.

En cuanto a los antecesores de las variables de evidencia, solo será necesario calcular los padres de las variables de evidencias, ya que los padres de las variables de tipo Y son variables de tipo X, y estas últimas no tienen padres.

Por lo tanto, podemos deducir que las variables relevantes, en nuestro problema en específico, están formadas por los padres de las variables de evidencia y las variables de consulta y de evidencia.

Dentro de la implementación de eliminación de variables de pgmpy, hay dos variables en concreto que nos interesan para esto.

En primer lugar, la variable **"elimination_order"**. Esta variable es una lista que contiene las variables para eliminar. Simplemente inicializamos "elimination_order" con las variables relevantes calculadas y eliminamos las variables de consulta y de evidencia (estás variables no pueden estar en elimination_order debido a cómo está implementado el algoritmo).

En segundo lugar, la variable **"working_factors"**. Esta variable es una copia de todos los factores de la red bayesiana, y es la variable sobre la que trabaja el algoritmo para no modificar los factores originales. En nuestra modificación, en lugar de copiar todos los factores de la red bayesiana, solo necesitamos los factores de aquellos nodos que coincidan con las variables relevantes. De esta forma "working_factors" solo tendrá los factores relevantes para resolver la consulta.

Debido a cómo están almacenados los factores originales, es imposible saber a priori qué factor corresponde a cada variable. Sin embargo, sabemos que los factores de las variables relevantes solo contienen variables relevantes. Gracias a esto, podemos hacer la copia de los factores que nos interesan a la variable "working_factors", de forma que copiamos todos los factores originales excepto aquellos que contengan alguna variable no relevante.

En nuestro sistema generamos una partida (una red bayesiana) y se van haciendo una serie de consultas a la red mediante inferencia exacta para calcular la casilla con mayor probabilidad de no contener mina.

La **segunda modificación** está basada en el hecho de que en cada consulta que vamos haciendo siempre acabamos teniendo las mismas evidencias de la consulta anterior más dos evidencias nuevas, que corresponden con la última casilla seleccionada. La idea es aprovechar esto y aplicar de manera permanente las evidencias en los factores de la red desde el primer momento en el que se generan.

Tal y como se explicó en la modificación anterior, la variable "working_factors" es una copia de los factores originales, sobre los que trabaja el algoritmo de eliminación de variables de pgmpy. Si nos fijamos en la sección de código en la que se aplican las evidencias a los factores de la red (cuarto bloque if-else de la función "_variable_elimination", tras el comentario "#Dealing with evidence. Reducing factors over it before VE is run." de la versión del algoritmo de pgmpy) [22], podemos ver que primero inicializa la variable "working_factors" y seguidamente aplica las evidencias sobre esta misma variable.

De esta forma las evidencias nunca se aplican de manera permanente a la red, ya que cuando se realiza una nueva consulta a la red bayesiana, se vuelve a inicializar la variable "working_factors", perdiendose los cambios producidos al aplicar las evidencias.

Lo que hemos hecho ha sido modificar la sección de código en la que se aplican las evidencias (citada anteriormente) [22], y hemos sustituido la variable "working_factors" por **"self.factors"**, que es donde se almacenan los factores originales.

De esta forma en cada consulta nueva, se van aplicando a los factores originales las evidencias que no hayan sido ya aplicadas, que siempre serán dos: las que corresponden con la última casilla seleccionada.

Ahora los factores originales deberán copiarse en "working_factors" justo después de aplicar las evidencias, para que esta variable tenga los factores actualizados.

Además, las evidencias dejarán de ser variables relevantes, ya que podemos considerar que han sido "eliminadas", una vez se han aplicado a los factores originales. Por tanto, ahora las variables de evidencia solo están formadas por los padres de las variables de evidencia y las variables de consulta.

En cuanto a la **tercera modificación**, el orden de eliminación de las variables puede influir en gran medida en la eficiencia del algoritmo de la eliminación de variables. Aunque calcular el orden óptimo se considera un problema NP-completo, existen algunas heurísticas que nos pueden ayudar página 4 del artículo [13].

Estas heurísticas debemos aplicarlas en "elimination_order" (citada anteriormente) para ordenar las variables de la red a eliminar.

La heurística en la que nos basamos para esta modificación es la llamada "min-degree heuristic" [23], también llamada "minwidth heuristic" y que, en nuestro caso en particular, equivale a "minweight heuristic", si consideramos que todos los nodos tienen peso 1.

Esta heurística consiste en ordenar las variables a eliminar en función del número de vecinos que tengan los nodos. Es decir, se eliminan primero las variables cuyos nodos tengan un menor número de vecinos.

Esto también tendrá un buen impacto en el rendimiento ya que, al eliminar primero las variables con menos vecinos, se van a formar factores más pequeños que si eliminamos primero las variables con más vecinos.

Modificación	Tiempo de ejecución (sg)
Sin modificar	Sin cuantificar
1	255.1148
2	9.7735
3	3.8298

Tabla 3. Resultados de los test

Finalmente, además de las mediciones de tiempo de las mejoras aplicadas de forma acumulativa, también se ha probado a realizar las mejoras de forma aislada. De forma asilada solo la modificación 1 es la que ha conseguido poder resolver un tablero de tamaño 3x3 o superior.

Por lo tanto, podemos concluir que las tres modificaciones han tenido un impacto relevante en el rendimiento, pero la que más impacto ha tenido ha sido la modificación 1, seguida de la modificación 2.

Como última puntualización, en el apartado 3 del artículo de referencia [13], se describe una primera aproximación de la simplificación de la red bayesiana. Esta aproximación fue implementada, pero observamos que el tiempo de creación de la red bayesiana (que en este apartado también incluye la moralización y triangulación del grafo) aumentaba considerablemente respecto a la mejora de tiempo de resolver una consulta.

Este aumento significativo del tiempo de creación del grafo se debe principalmente a las transformaciones de moralización y triangulación del grafo.

Por un lado, el proceso de moralización consiste básicamente en añadir aristas no dirigidas entre cada pareja de padres de un nodo y quitar la direccionalidad a las aristas iniciales del grafo, transformándose así en un grafo no dirigido.

Por otro lado, el proceso de triangulación se basa en crear ciclos triangulares dentro del triángulo. Ambas transformaciones suponen un alto coste de ejecución por todas las operaciones que deben realizar. Debido a esto llegamos a la conclusión de descartar esta mejora.

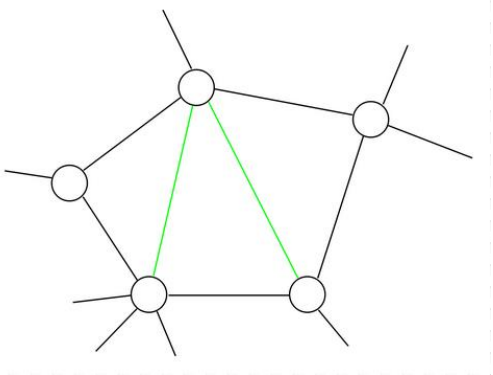

Ilustración 7. Grafo triangulado.

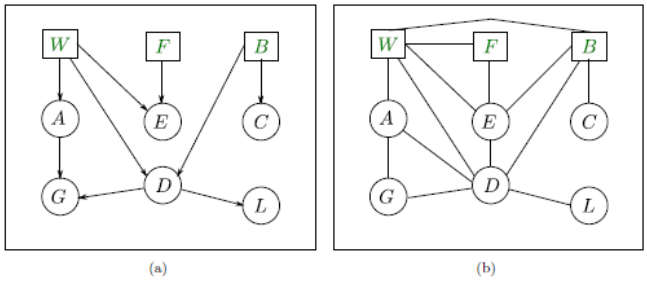

Ilustración 8. Un ejemplo de grafo dirigido (a) y de su grafo moral (b).

V. Conclusiones

El trabajo gira en torno al popular juego Buscaminas en el que, usando las redes bayesianas, debíamos conseguir implementar un mecanismo capaz de sugerir las casillas con menor probabilidad de contener una mina. Una vez desarrollado el sistema, iniciamos una fase de experimentación, en el cual observamos que el sistema era capaz siempre de completar una partida, pero donde las derrotas eran más frecuentes que las victorias.

Este trabajo nos ha permitido extender nuestros conocimientos desde aprender a usar un framework [4] como **qt5** para la interfaz gráfica hasta sacar mayor provecho del lenguaje de programación **python** con los comandos conocidos como "**magic commands**". Entre otras muchas cosas. Para más información acerca sobre estos comandos, visitar este enlace [18].

Realizar esta tarea ha sido útil para percatarnos de que el algoritmo de eliminación de variables no es eficiente (complejidad exponencial) y que si se quiere implementar dicho algoritmo en un sistema debe ser adaptado conforme a las características del sistema. Una alternativa a actualizar el algoritmo sería simplificar la red bayesiana.

Además, este proyecto nos ha hecho ganar experiencia trabajando en el ámbito de la inteligencia artificial ya que nunca antes habíamos hecho una tarea relacionada en este campo.

Esta actividad también nos ha servido para hacer progresos en la gestión y organización de un proceso de esta magnitud.

Para finalizar, de cara a una mejora del proyecto, podríamos aplicar las recomendaciones sugeridas en el punto 4 del artículo siguiente [13] y que no pudimos llevar a cabo por falta de tiempo.

Referencias

[1] Enunciado del trabajo propuesto sobre Modelización del juego Buscaminas con Redes Bayesianas. https://www.cs.us.es/cursos/iais-2017/trabajos/Buscaminas.pdf

[2] Tema de teoría sobre las Redes Bayesianas. https://www.cs.us.es/cursos/iais-2017/temas/RedesBayesianas.pdf

[3] Información sobre el algoritmo Flood Fill. https://en.wikipedia.org/wiki/Flood_fill

[4] Información sobre el framework qt. https://es.wikipedia.org/wiki/Qt_(biblioteca).

[5] Documentación oficial del framework. https://doc.qt.io/qt-5/qthelp-framework.html?hsCtaTracking=86310de7-fb82-4a3f-b320-045252048660%7Cfdd686d1-7d09-4707-89a7-417c5c297620#

[6] Juego de Buscaminas Online http://buscaminas.eu/.

[7] Enunciado de problemas y cuestiones sobre Redes Bayesianas https://www.cs.us.es/cursos/iais-2017/ejercicios/relacion-04.pdf

[8] Práctica correspondiente a Redes Bayesianas https://www.cs.us.es/cursos/iais-2017/practicas/Pr%C3%A1ctica_05.zip

[9] Módulo usado que implemente el algoritmo de Eliminación de variables. http://pgmpy.org/index.html

[10] Información sobre los DAGs. https://es.wikipedia.org/wiki/Grafo_ac%C3%ADclico_dirigido

[11] Introducción a la redes bayesianas. https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470061572.eqr089

[12] Información acerca del algoritmo de eliminación de variables: https://en.wikipedia.org/wiki/Variable_elimination

[13] Artículo. http://staff.utia.cas.cz/vomlel/vomlel-ova-cze-jap-2009.pdf

[14] Artículo derivado del primer artículo. http://staff.utia.cas.cz/vomlel/rank-one-decomposition.pdf

[15] Documentación de PyQt5. http://pyqt.sourceforge.net/Docs/PyQt5/

[16] Módulo de modelos gráficos de probabilidad. http://pgmpy.org/models.html

[17] Módulo de tablas de probabilidad condicionales y factores de probabilidad. http://pgmpy.org/factors.html

[18] Magic commands. https://ipython.readthedocs.io/en/stable/interactive/magics.html

[19] Lenguaje C++. https://es.wikipedia.org/wiki/C%2B%2B

[20] Guía estilo de código. https://www.python.org/dev/peps/pep-0008/?

[21] Sitio oficial del lenguaje Python. https://www.python.org/

[22] http://pgmpy.org/_modules/pgmpy/inference/ExactInference.html#VariableElimination

[23] Heurística. https://en.wikipedia.org/wiki/Minimum_degree_algorithm