Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Síntesis de Explicaciones en Redes Bayesianas

Autor(a): Bartomeu Ramis Tarragó
Tutor(a): Juan Antonio Fernández del Pozo de Salamanca

Madrid, Junio - 2023

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Ingeniería Informática

Título: Síntesis de Explicaciones en Redes Bayesianas

Junio - 2023

Autor(a): Bartomeu Ramis Tarragó
Tutor(a): Juan Antonio Fernández del Pozo de Salamanca
Grupo de Inteligencia Computacional
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Durante los últimos años hemos estado viviendo una explosión en la utilización y popularización de las tecnologías basadas en Inteligencia Artificial. A día de hoy el termino IA se ha convertido en sinónimo de avance tecnológico, novedad y progreso.

A pesar de esto, mucho de los sistemas que actualmente dominan el mercado de la IA son tecnologías de Machine Learning basadas en modelos no-explicables. Los modelos no-explicables son aquellos que requieren de herramientas externas para poder ser entendidos por humanos. En contraposición, los modelos explicables, o también llamados interpretables, presentan estructuras comprensibles para los humanos sin necesidad de utilizar herramientas externas. Este segundo conjunto de técnicas de IA (las llamadas XAI, Explainable Artificial Intelligence) presentan claras ventajas frente a los modelos de "caja negra" en cuanto a trazabilidad, interpretabilidad y la confianza que aportan estos modelos en la toma de decisiones.

Uno de los modelos interpretables más utilizados son las redes Bayesianas. Las redes Bayesianas son grafos dirigidos que proporcionan una representación gráfica para un conjunto de variables aleatorias y para las relaciones existentes entre ellas. La estructura de la red permite especificar la función de probabilidad conjunta de estas variables como el producto de funciones de probabilidad condicionadas. Este enfoque representa una buena estrategia para hacer frente a problemas relacionados con la incertidumbre, donde las conclusiones no pueden ser construidas sólo a partir de un conocimiento previo sobre el problema.

Es sobre este modelo de redes Bayesianas sobre el que trata este TFM. En concreto, la propuesta de este trabajo es producir un paquete de código en R que sea capaz de generar explicaciones a partir de modelos de redes Bayesianas y encontrar optimizaciones sobre la estructura de listas de reglas que permitan simplificar el proceso de toma de decisiones.

Para un problema de toma de decisiones modelizado en una red Bayesiana podemos representar la decisión que se tomará en cada caso en formato matricial. Sobre estas estructuras de datos aplicaremos algoritmos de optimización y transformación para obtener las "listas óptimas" que presentarán en un espacio reducido todas las reglas necesarias para la creación de un sistema de soporte para la toma de decisiones.

El objetivo final de este proyecto es proporcionar un paquete funcional en R [1] que permita, a partir de una red Bayesiana, generar dichas estructuras matriciales de reglas, optimizarlas espacialmente mediante varios algoritmos alternativos, estudiar y comparar la eficiencia de cada uno de esos algoritmos, y finalmente, generar explicaciones útiles de las decisiones tomadas por el modelo.

Abstract

Over the last few years we have been living through an explosion in the popularity of AI-based technology. Currently, the term AI has been transformed into a synonym for technological advancement, novelty, and progress.

However, many of the most popular systems in the AI sector rely on Machine Learning technology based on nonexplainable models. Nonexplainable models are those defined by the need for external tools to be understood by humans. In direct contrast, explainable, or interpretable, models are those that are humanly comprehensible without the need for external tools. This second set of AI techniques (also known as XIA) show clear advantages against "black box" models in terms of accountability, interpretability, and trust given to the user.

One of the most used interpretable models are Bayesian Networks. Bayesian Networks are defined as directed graphs of a set of random variables and the relations between them. The network structure allows us to specify the joint probability function of these variables as the product of conditional probability functions. This approach represents a good strategy for dealing with uncertainty-related problems, where conclusions can not be built solely from prior knowledge about the problem.

It is this Bayesian network model that is the subject of this TFM. Specifically, the proposal of this work is to produce a code package in R that is able to generate explanations from Bayesian network models and to find optimizations on the structure of the list of rules that allow to simplify the problem spatially.

For a decision making problem modeled in a Bayesian network we can represent the decision to be made in each case in matrix format. On these data structures we will apply optimization and transformation algorithms to obtain the "optimal lists" that will present in a reduced space all the necessary rules for the creation of a decision support system.

The final objective of this project is to provide a functional package in R that allows, from a Bayesian network, to generate such matrix structures of rules, to optimize them spatially by means of several alternative algorithms, to study and compare the efficiency of each one of these algorithms, and finally, to generate useful explanations of the decisions taken by the model.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

Actualmente estamos viviendo una explosión en la innovación, investigación y utilización de sistemas de IA. Cada día más y más industrias adoptan soluciones basadas en IA a medida que los modelos más potentes de este campo se hacen también más accesibles y conocidos por el público general.

Los sistemas de IA y aprendizaje automático pueden ser muy poderosos usados de forma correcta, pero a menudo nos encontramos con cajas negras, es decir, sistemas opacos que ofrecen respuestas pero no explicaciones. Estos sistemas son llamados modelos no-explicables, es decir, sistemas de IA los cuales no son fácilmente comprensibles para los humanos, y que por tanto, requieren de herramientas o métodos externos para poder "entender" el proceso de razonamiento [3]. Esto es problemático, ya que la complejidad de estos sistemas puede esconder prejuicios en los datos de entrada o errores en el razonamiento del sistema o en los algoritmos, que pueden llevar a la toma de decisiones injustas o incorrectas [4].

Es por esto que cada vez encontramos una creciente necesidad de poder contar con sistemas de IA que no solo nos recomienden las mejores decisiones, sino que sean capaces de ofrecer un vistazo al "porqué" de las decisiones tomadas [5].

Los métodos interpretables, o explicables, para la toma de decisiones se centran en desarrollar algoritmos y sistemas que puedan explicar cómo se llega a una decisión particular. Esto puede incluir la identificación de las variables más influyentes en una decisión, el seguimiento de la cadena de razonamiento detrás de una decisión o la identificación de los datos que se utilizaron para tomar dicha decisión. La toma de decisiones en modelos explicables resulta imprescindible en muchos sectores críticos dónde la trazabilidad o "accountability" resulta una obligación [6]. Esto es importante en aplicaciones en las que las decisiones pueden tener consecuencias significativas para las personas, como en la medicina, el derecho o la ingeniería. Además, la transparencia en estos sistemas puede ayudar a identificar y corregir errores que puedan existir en los datos o en los propios algoritmos. Así pues, las redes Bayesianas o en general, los sistemas de IA interpretables (habitualmente abreviado XAI) representan una oportunidad para poder aplicar las herramientas de IA y aprendizaje automático en entornos donde hasta ahora resultaba demasiado arriesgado utilizar sistemas de "caja negra".

1.2. Estado del arte

1.2.1. Redes Bayesianas

Las redes Bayesianas son herramientas probabilísticas surgidas en el campo de la IA que permiten afrontar situaciones de investigación con incertidumbre. Las redes Bayesianas proporcionan una representación gráfica para un conjunto de variables aleatorias y para las relaciones probabilísticas existentes entre ellas. La estructura de la red permite especificar la función de probabilidad conjunta de estas variables como el producto de funciones de probabilidad condicionadas. Este enfoque representa una buena estrategia para hacer frente a problemas relacionados con la incertidumbre, donde las conclusiones no pueden ser construidas sólo a partir de un conocimiento previo sobre el problema.

Las redes Bayesianas se encuentran entre los modelos gráficos más populares. La principal diferencia, con respecto a otros modelos, está en que sus arcos son dirigidos y representan dependencia condicional entre las variables.

El término red Bayesiana fue utilizado por primera vez por Judea Pearl en su papaer titulado *Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning*, publicado en 1985 [7]. En este artículo, Pearl define las redes Bayesianas como grafos dirigidos acíclicos, donde los nodos representan variables y los arcos las relaciones de causalidad entre estos nodos. Utilizando esta estructura es posible entonces, no solo almacenar las complejas relaciones de casualidad de un dominio conocidas solo por expertos, sino también para activar este conocimiento y trabajar con él.

El razonamiento probabilístico sobre las redes Bayesianas consiste en propagar los efectos de las evidencias (variables conocidas) a través de la red para conocer las probabilidades a posteriori de las variables desconocidas. De esta forma se puede determinar un valor estimado para dichas variables en función de los valores de probabilidad obtenidos.

El mecanismo de inferencia sobre redes Bayesianas permite utilizarlas para construir modelos muy generales de interacción y correlación de causalidad entre las variables de un problema bajo incertidumbre [8]. Un ejemplo interesante son los clasificadores. Estos modelos, se construyen a partir de una red Bayesiana en la que clase y atributos sean las variables interrelacionadas en el grafo. La clase corresponderá a la variable desconocida, objetivo de la inferencia. Dada una instancia cualquiera para la que se conozcan todos sus atributos, la clasificación se efectuará infiriendo sobre el grafo la probabilidad posterior de cada uno de los valores de la clase, y seleccionando aquel valor que maximice dicha probabilidad.

Así pues, se pueden construir clasificadores bayesianos a partir del conocimiento de expertos. Estos modelos construidos "manualmente" son capaces de aprovechar la experiencia de profesionales en sus respectivos campos para definir las probabilidades condicionadas en todos los puntos del grafo. De esta forma, realizando preguntas concretas a los expertos de cierto campo se pueden construir modelos capaces de clasificar y ayudar en la toma de decisiones importantes tomando como entrada cantidades muy grandes de datos y/o conocimiento experto.

Los procesos de inferencia producidos por los clasificadores debe ser explicados para comprender y justificar los resultados y contextualizar y categorizar los atributos

según relevancia y redundancia. Así pues, se requiere métodos adicionales para poder tomar las clasificaciones de las redes Bayesianas y generar explicaciones lógicas que justifiquen las decisiones tomadas por el propio modelo [9].

1.2.2. Generación de explicaciones

Las redes Bayesianas, en el contexto de sistemas expertos, presentan una gran ventaja respecto a otros modelos: la interpretabilidad. La interpretabilidad es la capacidad de un modelo de IA a ser comprendido en términos humanos. Así pues, mientras las redes neuronales son básicamente cajas negras, imposibles de entender sin un exhaustivo análisis, las redes Bayesianas utilizan una cadena de razonamiento fácilmente comprensible [10]

Al trabajar con un modelo gráfico, los mecanismo de inferencia quedan claramente representados en el grafo del modelo. La dirección de la propagación de las evidencias y las probabilidades a posteriori resulta de fácil comprensión para individuos que no tienen porqué ser expertos en el campo de la IA. De esta forma, un médico puede sentirse seguro en el razonamiento del modelo y en que el diagnóstico recibido es lógico, comprensible y consistente con el conocimiento y la experiencia en este campo. Para esto, los procesos internos del modelo deben ser explicables en lenguaje natural, de una forma en que profesionales de otros mundos aparte de la informática o la estadística sean capaces de comprenderlos.

En la actualidad, esta necesidad de contar con sistemas de IA explicables y confiables está siendo identificada y cada vez más vemos proyectos que tratan de aportar soluciones [11].

En general, todos estos proyectos comparten las mismas consideraciones a la hora de generar explicaciones en lenguaje natural. Podemos identificar 3 grandes grupos de parámetros de los tipos de explicaciones para las redes Bayesianas [12]: el contenido de la explicación, la comunicación y la adaptación.

1. El contenido referencia todo aquello dentro de la explicación de una decisión.

   Foco: qué elementos proporcionados destacaremos para generar las explicaciones, la evidencia, el propio modelo y sus relaciones o el proceso de inferencia y razonamiento usado para llegar a la conclusión.

   Objetivo: qué se pretende conseguir con las explicaciones, una descripción del problema y/o de la decisión tomada o una compresión de los motivos de dicha decisión.

   Método de explicación: qué estilo se utilizará micro o macro. Iniciaremos las explicaciones a partir de la evidencia individual e iremos construyendo la explicación general o partiremos de una explicaciones a gran nivel y progresivamente iremos bajando al nivel de datos individuales.

2. La comunicación hace referencia al modo de transmitir dichas explicaciones. Como interactúa el usuario final con el sistema experto puede afectar a la comprensión de este mismo. Por eso queremos considerar:

   Interacción usuario-sistema: puede ser estilo menú, con siempre las mismas opciones predefinidas; un sistema de consulta al modelo que se adapte a

las pregunta hechas; o un sistema basado en lenguaje natural con el que se pueda conversar como si fuera un experto humano.

Muestra de explicaciones: puede estar basada en texto, en sistemas gráficos o pueden ser sistemas multimodales que combinan explicaciones escritas con figuras gráficas.

Expresiones de probabilidad: resultan especialmente importantes para transmitir el proceso de inferencia al usuario final. Las probabilidades pueden expresarse en forma de porcentaje, de forma comparativa o utilizar cualquier combinación basada en números y expresiones lingüísticas.

Adaptación: hace referencia a la capacidad del modelo a adaptarse a las necesidades del usuario.

3. Por último, la adaptación hace referencia a la capacidad del modelo a adaptarse a las necesidades del usuario.

   Adaptación al conocimiento: puede estar basada en niveles de conocimiento (nivel bajo, medio, alto), puede ser dinámico y cambiar de explicación a explicación o puede no ser adaptable.

   Estilo de razonamiento: puede también adaptarse a las necesidades del usuario. Se pueden generar explicaciones a bajo nivel, siguiente el proceso del modelo, o a más alto nivel, tratando de alejarse de los métodos probabilísticos.

   Nivel de detalle: también puede adaptarse o no. Cuanto más detalle, más parámetros y más etapas se incluyen en las explicaciones.

Cabe destacar, que uno de los parámetros más importantes en la lista anterior es el foco de la explicación. Este elemento resulta uno de los más interesantes de estudiar y de identificar para un sistema de toma de decisiones. Como ya hemos mencionado, el foco de la explicación puede ser sobre el modelo, el método de razonamiento, la evidencia obtenida o incluso, en la decisión tomada [13].

1.2.3. Representación de tablas de decisiones óptimas

Otro de los puntos en los que nos centraremos en este trabajo son las tablas de decisiones óptimas. Las tabla de decisiones óptimas (TDO, Knowledge Base o KB) son estructuras de datos utilizadas para dar soluciones a problemas de toma de decisiones bajo incertidumbre. Estas tablas recogen todos los posibles casos y variaciones de los datos de entrada desconocidos e identifican la decisión que se tomaría en cada uno de esos casos.

Las tablas de decisiones óptimas resultan una herramienta muy valiosa a la hora de tratar con problemas en el mundo real, ya que estos presentan un gran nivel de incertidumbre, operan con información incompleta y gestionar redes Bayesianas complejas. Aún así, estas estructuras resultan muy costosas computacional y espacialmente, ralentizando el trabajo y las operaciones sobre ellas.

Esta problemática es la que trata el paper *A list-based compact representation for large decision tables management* [14]. El artículo discute los desafíos de construir tablas espacialmente eficientes que sigan proporcionando explicaciones claras, concisas,

consistentes y completas a partir de las mencionadas TDO. La solución propuesta se basa en la gestión de matrices multidimensionales que minimicen el espacio utilizado.

La idea detrás de esta estructura es la siguiente: Las TDO contienen todas las posibles combinaciones de los atributos de entrada con la decisión que se toma en cada uno de esos casos. Si imponemos un orden predefinido en los atributos, un cambio en la posición de las variables cambia la estructura y orden de los elementos de la tabla, pero no de su contenido. Así pues, si somos capaces de maximizar la cantidad de elementos contiguos de la tabla que resultan en la misma decisión, podremos reducir el tamaño de la tabla. A este proceso de reducción se le denomina coalescencia.

Consecuentemente, si nuestro objetivo es la optimización combinatoria (encontrar el orden concreto de atributos de la tabla que maximiza los elementos contiguos con la misma decisión) debemos saber que tratamos con un problema NP-difícil. Por tanto, se deben hacer uso de heurísticas que sean capaces de dirigir la búsqueda. En el paper se sugieren y se comparan dos heurísticas y varios algoritmos de búsqueda para nuestro problema.

1.2.4. Algoritmos de optimización

Como ya hemos explicado, una vez que se obtiene un lista de decisiones para un problema real, la estructura de datos necesaria tiene un tamaño intratable. Las grandes matrices utilizadas para guardar dicha información son poco prácticas y muy costosas. Es por esto que es imprescindible que optimicemos dichas listas para poder trabajar sobre ellas de forma ágil y rápida. Es aquí dónde será necesario aplicar técnicas de optimización espacial basadas en heurísticas con el objetivo de reducir el tamaño de las listas KBM2L.

Antes de pasar a los algoritmos, vamos a describir rápidamente el problema de la optimización y la heurística definida para dirigir la búsqueda.

El problema de optimización de las listas de decisiones se trata de una búsqueda combinatoria sobre todas las posibles ordenaciones de las columnas que forman la base de conocimiento (KB). Nuestro objetivo es encontrar aquella matriz con el menor número de reglas que sea comprensible para un humano. Esto requiere reducir el número de reglas de los cientos, miles o millones a una cantidad gestionable, más cercana a las decenas (a partir de las 20 reglas la complejidad empieza a ser demasiada incluso para un experto). Aún así, no solo es importante el número absoluto de reglas, sino la complejidad de estas. Supongamos que disponemos de un modelo con 100 variables aleatorias, es preferente obtener 15 reglas sencillas, que trabajen con 2 o 3 variables cada una, que con un conjunto muy reducido de reglas (entre 2 y 5), pero que cada una de ellas tenga 10, 15 o 20 variables aleatorias involucradas. Así pues, necesitamos encontrar un equilibrio entre las reglas y su complejidad (el número de variables aleatorias relevantes presentes en cada regla).

Entrando en detalle en el método de optimización, lo que queremos es encontrar la ordenación específica de las variables aleatorias que forman la red tal que se produzcan el mínimo de cambios entre decisiones contiguas.

En el ejemplo sencillo de la Tabla 1.1 disponemos de una tabla muy reducida de tan solo 2 variables aleatorias. Cada una de las variables, A y B, dispone tan solo de 2 estados (a_0, a_1 y b_0, b_1 respectivamente). De la misma forma, la decisión solo tiene 2

A	B	Decisión
a_0	b_0	D_0
a_0	b_1	D_1
a_1	b_0	D_0
a_1	b_1	D_1

Cuadro 1.1: TDO básica

B	A	Decisión
b_0	a_0	D_0
b_0	a_1	D_0
b_1	a_0	D_1
b_1	a_1	D_1

Cuadro 1.2: TDO básica optimizada

estados, D_0 y D_1. Al orden de las variables aleatorias en la tabla se llama la **base** de la lista. En este caso, la base actual es {A, B}

En el primer caso, la Tabla 1.1 presenta 4 reglas, es decir, 4 cambios entre las decisiones contiguas ya que cada fila tiene una decisión distinta a la anterior. Esto es debido al orden de las variables aleatorias en la tabla. No resulta difícil ver que la variable B es la única que es relevante a la hora de elegir la decisión. Es por esto que B debería ser la variable de más peso en la lista y por tanto, estar situada a la izquierda.

El procedimiento de optimización debe buscar entre las posibles reordenaciones de las columnas de la tabla (las posibles bases) y encontrar aquella que reduzca el número de reglas. En nuestro ejemplo trivial solo hay una posible base alternativa, la cual se muestra en la Tabla 1.2. La nueva tabla con base {B, A} tiene únicamente 2 reglas.

Para la representación reducida de las tablas utilizaremos la nomenclatura basada en índices. Esta representación parte de una tabla de decisiones y forma la KBM2L (Knowledge Base Matrix to List) a partir de los elementos de la tabla relevantes, en nuestro caso, ya que la tabla se encuentra siempre ordenada, solo aquellos casos anteriores a un cambio de decisión son relevantes. Así pues, para representar una regla se debe indicar el valor de los atributos ordenados según la base del último caso de una regla seguido de la decisión tomada. De esta forma se define un rango de casos con la misma decisión: desde el ítem inmediatamente siguiente de la regla anterior (o desde el caso inicial si se trata de la primera regla), hasta el caso indicado por la regla.

Utilizando la Tabla 1.1 como ejemplo, la KBM2L en notación en índices es la siguiente:

$$< (a_0, b_0), D_0| < (a_0, b_1), D_1| < (a_1, b_0), D_0| < (a_1, b_1), D_1|$$

Como vemos, disponemos de 4 reglas. Una para el caso inicial y 3 para los cambios entre decisiones inmediatos. Si generamos la KBM2L de la Tabla 1.2 veremos que la lista solo tiene 2 elementos, la decisión inicial y un cambio de decisión.

$$< (b_0, a_1), D_0| < (b_1, a_1), D_1|$$

Esta lista nos aporta la misma información que la tabla, ya que sabemos que para todos los casos anteriores a (b_0, a_1) la decisión es la misma (D_0), y para todos los casos a partir de (b_0, a_1) y hasta (b_1, a_1) la decisión es D_0. Así pues, nuestras búsquedas deben encontrar aquellas bases que generen las KBM2L con menos elementos y, a poder ser, con el menor número de atributos relevantes. En nuestro caso, se han elegido 3 algoritmos los cuales varían en complejidad y rendimiento.

1.2.4.1. Algoritmo de Taboo search

La búsqueda tabú o "taboo search" es un mecanismo de búsqueda metaheurístico que utiliza la estrategia de búsqueda local o por vecindario. La idea principal tras este algoritmo es evitar quedar estancado en mínimos locales marcando las posibles soluciones como "tabú", prohibiendo volver a visitarlas.

El algoritmo fue propuesto por primera vez por Fred Glover en 1986 [15] y formalizado en un artículo el 1989 [16]. La propuesta de Glover con este algoritmo era el desarrollo de un método de optimización combinatoria enmarcado en el campo de la IA. Aún así, este método resultó ampliamente popular y versátil, siendo capaz de dar buenos resultados en problemas que van desde el "scheduling" y equilibrio de canales, hasta el análisis de clusters e incluso en el tratamiento del problema del viajante.

Este algoritmo hace uso de una heurística para poder dirigir la búsqueda. En nuestro caso, queremos encontrar aquella base que ordene las variables aleatorias de una red Bayesiana de tal forma que la KBM2L derivada sea mínima. Entonces la heurística a utilizar resulta intuitiva, se ha de buscar aquella base con el menor número de reglas, con el objetivo de minimizar la longitud de la KBM2L. Esta primera solución presenta un problema: la complejidad combinatoria y el coste computacional. En nuestros ejemplos anteriores, habíamos usado una base con tan solo 2 variables, cada una con 2 estados. Esto rara vez es así, puesto que la mayoría de redes Bayesianas presentan muchas mas variables y un número de estados muchísimo mayor.

Como sabemos, el número de posibles ordenaciones de una lista donde los elementos no pueden repetirse son:

$$!n = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 3 \cdot 2 \cdot 1$$

Siendo n el cardinal de del conjunto de variables del modelo. En nuestro caso, una red Bayesiana puede tener más de cientos de variables aleatorias. Así pues, con tan solo una base de 10 elementos, el número de posibles permutaciones son 3.628.800. Por tanto, para una red Bayesiana sencilla, de tan solo 10 variables, debemos buscar entre más de 3 millones y medio de bases.

Por otro lado, la longitud de cada tabla de decisión depende también del número de estados que tiene cada variable. La fórmula que siguen las longitudes de las tablas es la siguiente:

$$L_t = \prod_{i=1}^{n} e_i$$

Siendo n el número de variables de la red, y e_i el número de estados de la variable i.

En nuestro ejemplo solo teníamos dos variables {A, B} y estas podía tomar solo dos valores, pero en un caso real, se pueden presentar mas de 100 variables, cada una

con hasta 5 estados. Así pues, si tenemos en cuenta que para realizar una reordenación de nuestra tabla original el coste computacional es lineal al número de ítems de la tabla y que el el espacio de soluciones (las posibles ordenaciones) de una base es exponencial con la cantidad de variables, rápidamente nos damos cuenta que nuestra solución anterior es inviable. La cantidad de posibles ordenaciones y el coste computacional de reordenar la tabla para cada una de ellas es inasequible.

De esta forma, tomaremos dos decisiones para simplificar el problema y dirigir la búsqueda de las bases óptimas. En primer lugar, definiremos el "vecindario" de una base como todas aquellas bases que se pueden obtener al realizar una permutación de dos elementos de la base original.

Por ejemplo , dada una base $B_0 = \{A, B, C, D\}$, sus bases vecinas no son todas las posibles ordenaciones ($4! = 24$), sino aquellas obtenidas a partir de intercambiar la posición de dos elementos, es decir:

- $B_1 = \{B, A, C, D\}$
- $B_2 = \{C, B, A, D\}$
- $B_3 = \{D, B, C, A\}$
- $B_4 = \{A, C, B, D\}$
- $B_5 = \{A, D, C, B\}$
- $B_6 = \{A, B, D, C\}$

Con esta nueva definición de vecindad entre bases, ya no trabajamos sobre todo el espacio de soluciones, sino un subconjunto de ellas mucho más reducido. Dada una base B_0 con n variables aleatorias, el número de vecinos es:

$$N(n) = \frac{n!}{2 \cdot (n-2)} = \frac{n \cdot (n-1)}{2}$$

De esta forma, para una base con 10 elementos, el vecindario es de tan solo 45 bases.

Por otro lado, para evitar realizar una costosa reordenación de la tabla para cada uno de los vecinos, pasaremos a gestionar una nueva función G [14]. El objetivo de esta función será darnos una idea aproximada de la "similitud" entre dos tablas a partir de sus respectivas bases.

$$G(B, B') = r_{left} + \left(\sum_{i=0}^{n-1} r_i + H(B, B') - 1 \right)$$

Donde B y B' son la base original y la nueva base a visitar respectivamente, r_{left} es el número de atributos no permutados a la izquierda de la variable permutada con más peso, n es el número de atributos, r_i es el número de variable entre la posición inicial y la posición final de la variable i y $H(B, B')$ es la medida de Hamming entre las dos bases. La distancia de Hamming se define como el número de caracteres distintos entre dos palabras de la misma longitud [17] .

La medida G está basada en la distancia de Hamming, pero no se trata de una medida de distancia, ya que no cumple la desigualdad triangular $G(B, B') \leqslant G(B, B'') + G(B'', B') \forall B, B', B''$

Esta medida nos servirá para dirigir la búsqueda y compara bases entre ellas sin necesidad de hacer una reordenación de las tablas para cada vecino. La función G es una heurística que funciona a modo de aproximación a la diferencia entre dos tablas a partir de sus bases. Para entender mejor esta medida hay que entender las consecuencias de permutar los atributos de una base. Si permutamos atributos de mucho peso, es decir, aquellos que se encuentran más a la izquierda de la base, obtenemos una reordenación de grandes bloques de la propia tabla, resultando en una configuración "similar" a la tabla previa. Mientras que si se realizan cambios en los atributos de poco peso, la reordenación de las filas es en pequeños grupos, realizando muchos cambios locales, resultando en una tabla más distinta a la original.

Por ejemplo, dada la Tabla 1.3 si aplicamos una transformación de mucho peso, veremos que se produce una reordenación de grandes bloques, quedando intactos secuencias de 4 filas, como se puede ver en la Tabla 1.4. En este caso, la tabla resultante es poco óptima, con una regla por fila.

Por otro lado, si aplicamos una permutación sobre las variables de menor peso, veremos que se produce una reordenación a nivel de filas individuales de la tabla. Como se ve en la Tabla 1.5, el resultado es una KBM2L con la mitad de reglas y por tanto, más óptima espacialmente.

A	B	C	D	Decisión
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_0
0	0	1	1	D_1
0	1	0	0	D_0
0	1	0	1	D_1
0	1	1	0	D_0
0	1	1	1	D_1
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_0
1	0	1	1	D_1
1	1	0	0	D_0
1	1	0	1	D_1
1	1	1	0	D_0
1	1	1	1	D_1

Cuadro 1.3: Tabla T base

De esta forma, nuestra medida G nos permite identificar de manera numérica aquellas transformaciones que modifican en mayor o menor medida la tabla original. Una permutación de atributos de B a B' dónde $G(B, B')$ sea un número menor, indica una reordenación de los atributos de mayor peso y por tanto, cambios de grandes bloques, lo que implica una nueva tabla más bien similar a la original. Mientras que una $G(B, B')$ de mayor número indica una mayor reordenación de filas a nivel individual, resultando, por lo general, en una tabla más distinta.

Siguiendo el ejemplo anterior siendo B la base de la Tabla 1.3, B' la base de la

B	A	C	D	Decisión
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_0
0	0	1	1	D_1
0	1	0	0	D_0
0	1	0	1	D_1
0	1	1	0	D_0
0	1	1	1	D_1
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_0
1	0	1	1	D_1
1	1	0	0	D_0
1	1	0	1	D_1
1	1	1	0	D_0
1	1	1	1	D_1

Cuadro 1.4: Tabla T' resultado de una permutación de los atributos de más peso

A	B	D	C	Decisión
0	0	0	0	D_0
0	0	0	1	D_0
0	0	1	0	D_1
0	0	1	1	D_1
0	1	0	0	D_0
0	1	0	1	D_0
0	1	1	0	D_1
0	1	1	1	D_1
1	0	0	0	D_0
1	0	0	1	D_0
1	0	1	0	D_1
1	0	1	1	D_1
1	1	0	0	D_0
1	1	0	1	D_0
1	1	1	0	D_1
1	1	1	1	D_1

Cuadro 1.5: Tabla T" resultado de una permutación de los atributos de menor peso

Tabla 1.4 y B'' la base de la Tabla 1.5, vemos que:

$$G(B, B') = 0 + ((0) + 2 - 1) = 1$$
$$G(B, B'') = 2 + ((0) + 2 - 1 = 3$$

Ahora que entendemos las herramientas podemos volver a definir el método que usaremos para resolver este problema de búsqueda. Así pues, nos encontramos frente a un problema de búsqueda combinatoria en el cual usaremos una metaheruísitca G para optimizar espacialmente las tablas, buscando aquella base que minimice el número de reglas de una KBM2L derivada de una tabla de decisiones.

1.2.4.2. Simulated annealing

El algoritmo de simulated annealing, también llamado algoritmo de recocido simulado, es un algoritmo de búsqueda global metaheurística basado en el proceso de enfriamiento de los metales.

El nombre de recocido simulado fue propuesto por Kirkpatrick, Gelatt Jr., Vecchi en 1983 [18]. A pesar de esto, este planeamiento había sido descrito de forma independiente por varios autores anteriores y posteriores a dicho artículo, como Pincus en 1970 [19], Khachaturyan en 1979 [20] y Cerny en 1985 [21] [22].

La idea tras este algoritmo es la búsqueda de la optimización de una función utilizando la misma lógica de enfriado de metales. En la metalurgia, el recocido de metales es una técnico donde el metal se enfría lenta y controladamente con el objetivo de eliminar tensiones internas en el metal y obtener configuraciones de baja energía. Equivalentemente, en informática, este algoritmo sugiere la utilización de una variable T, la cual se asimila a la temperatura, que empieza en un valor muy elevado (cercano a infinito) y poco a poco va descendiendo hasta $T = 0$ indicando el final del enfriamiento [23].

El algoritmo parte de una solución inicial factible, y transita a una solución mejor del del entorno, mejorando iterativamente. La trayectoria de soluciones generadas se aproximan a la solución óptima.

La variación de T desde una temperatura alta a temperatura 0 significa entonces el afinamiento del sistema de búsqueda de soluciones a medida que el algoritmo avanza. Inicialmente, con la temperatura máxima, el algoritmo trata de buscar de forma general en todo el espacio de soluciones óptimos globales. Pero a medida que T reduce su valor, la búsqueda se va ajustando progresivamente a óptimos locales hasta terminar en $T = 0$.

De la misma forma que en el algoritmo anterior vamos a utilizar las mismas estrategias para planear el problema de una forma realista. La búsqueda de vecinos se seguirá realizando mediante las permutaciones de dos atributos en una base. Y la medida G se seguirá usando como heurística para identificar cada uno de los vecinos.

1.2.4.3. Algoritmo genético

El algoritmos genético es un sistema de búsqueda poblacional basado en el proceso de la evolución en el contexto de la biología. Este algoritmo trata de simular la selección,

cruce y mutación de los genomas en poblaciones de seres vivos encontrando así al final los individuos "mejor adaptados".

Los algoritmos genéticos se remontan hasta los años 60. En 1965 Ingo Rechenberg propone la llamada "Estrategia Evolutiva", una versión primitiva de los algoritmos genéticos actuales que carece de algunas de las características esenciales como la mutación o el cruce [24].

Es en 1975 cuando todas las ideas relacionadas con estos sistemas son culminadas en el libro *Adaptation in Natural and Artificial Systems* de John Holland [25]. Fué aquí donde se definieron estos algoritmos como sistemas digitales adaptativos basados en los mecanismos de selección, cruce y mutación. Ese mismo año, Jong demostró el potencial de estos algoritmos en una gran variedad de problemas [26].

El mecanismo principal de este algoritmo itera sobre una población inicial de individuos 3 etapas, resultando en una nueva generación de individuos ligeramente distintos a los originales.

El paso inicial consiste en la creación de una población aleatoria. Cada individuo (es decir, solución) ha de ser lo más único posible, de esta forma se consigue "repartir" la población por todo el espacio de soluciones, tratando de situarse cerca de los puntos óptimos.

A partir de esta población inicial se inicia el proceso iterativo donde para cada población se:

Ordenación Inicialmente, cada nuevo individuo es evaluado y asignado un valor de "fitness". Este valor será usado para ordenar a los nuevos individuos de mejor a peor e iniciar el proceso de selección para el cruce.

Cruce Los individuos se seleccionan utilizando una función de selección basada en la "fitness" de cada individuo y se van cruzando por parejas entre ellos. Aquellos individuos mejor adaptados son más probables a ser seleccionados y por tanto, tienen un mayor número de descendientes. Los "hijos" de una pareja deben de estar formados por las dos mitades de sus progenitores, heredando así de forma aleatoria las características.

Mutación Cada vez que se genera un nuevo individuo este puede sufrir mutaciones. Las mutaciones implican cambios aleatorios en el genoma heredado. De esta forma, los descendientes de una misma pareja no tienen porque ser todos iguales. Esto aporta cambios aleatorios en la nueva generación que pueden mejorar, empeorar o mantener igual la "fitness" de la población.

Selección Aquellos individuos "mejor adaptados" de cada generación son seleccionados y pasan a la siguiente generación sin sufrir cambios. Durante el proceso iterativo es posible que se llegue a un óptimo global, es decir, que se genere al mejor individuo posible o a un individuo muy cercano a la perfección. Es por esto que las "elites" de cada generación serán seleccionados para seguir formando parte de la siguiente generación y así no perder posibles soluciones óptimas.

1.3. Objetivos y alcance

El objetivo principal es desarrollar un paquete en R que trabaje sobre redes Bayesianas. El código debe ser capaz de evaluar modelos y general listas de explicaciones para los resultados obtenidos. Estas explicaciones se basan en listas KBM2L, y deben implementar algoritmos para su optimización espacial.

El objetivo final es ser capaces de generar explicaciones basadas en reglas para la toma de decisiones usando modelos de redes Bayesianas. El TFM incluye un trabajo de investigación sobre el estado del arte de las tecnologías, algoritmos y métodos actuales para trabajar con redes Bayesianas. Un estudio del problema y de sus posibles soluciones, además de un documento recogiendo los resultados obtenidos, comparando métodos y probando el código con conjuntos de datos tanto reales como sintéticos.

Así pues, los objetivos son los siguientes:

- Investigación del estado del arte en redes Bayesianas, generación de listas de reglas y algoritmos de optimización.
- Realizar un estudio del problema y de sus posibles soluciones.
- Desarrollar un paquete en R que trabaje con redes Bayesianas.
- Implementar un algoritmo de evaluación de modelos en el paquete en R.
- Desarrollar un algoritmo de generación de listas KBM2L en el paquete en R.
- Implementar varios algoritmos de optimización espacial para las listas KBM2L en el paquete en R.
- Generar explicaciones basadas en reglas para la toma de decisiones usando modelos de redes Bayesianas.
- Generar diálogos sencillos que permitan la comunicación y extracción de explicaciones entre las listas de reglas obtenidas a partir de las redes Bayesianas y los usuarios
- Documentar los resultados obtenidos y comparar los distintos métodos.
- Probar el código desarrollado con conjuntos de datos tanto reales como sintéticos.
- Presentar una memoria detallada sobre el trabajo realizado que incluya todos los objetivos, las decisiones y los resultados obtenidos durante el desarrollo del proyecto.

Capítulo 2

Desarrollo

2.1. Propuesta inicial

En este primer apartado explicaremos la propuesta inicial del proyecto de Trabajo de Fin de Máster que se realizó al inicio del proyecto. Esta propuesta incluye una breve descripción del trabajo a realizar, con los conceptos básicos, la motivación, los objetivos e incluso una lista de tareas divididas en 7 aparatos y explicadas brevemente.

2.1.1. Descripción general del trabajo

Se pretende desarrollar un paquete en R que trabaje sobre redes Bayesianas. El código debe ser capaz de evaluar modelos y general listas de explicaciones para los resultados obtenidos. Estas explicaciones se basan en listas KBM2L, y deben implementar algoritmos para su optimización espacial. El objetivo final es ser capaces de generar explicaciones basadas en reglas para la toma de decisiones usando modelos de redes Bayesianas. El trabajo incluye un trabajo de investigación sobre el estado del arte de las tecnologías, algoritmos y métodos actuales para trabajar con redes Bayesianas. Un estudio del problema y de sus posibles soluciones, además de un documento recogiendo los resultados obtenidos, comparando métodos y probando el código con conjuntos de datos tanto reales como sintéticos.

Los objetivos principales desglosados se encuentran en el Capítulo 1.3.

2.1.2. Lista de tareas

Las tareas se han dividido en 7 grandes grupos divisibles en las siguientes tareas individuales:

1. Planificación

   Consiste en la identificación de tareas, aproximar el esfuerzo requerido y plasmar los objetivos, los hitos y las tareas en un eje temporal (diagrama de Gantt). Además de generar el primer entregable, el Plan de Trabajo.

   1.1. Planificación inicial

   1.2. Plan de trabajo

2. Estado del arte

Consiste en la investigación y búsqueda de los métodos, algoritmos y tecnologías actualmente usadas y que servirán para elaborar e implementar la solución a nuestro proyecto.

2.1. Redes Bayesianas

2.2. Redes Bayesianas Clasificadoras

2.3. Algoritmos de simulación

2.4. Representación de soluciones en redes Bayesianas y KBM2L

2.5. Algoritmos de optimización espacial

2.6. Paquetes en R

3. Estudio del problema y soluciones

   Consistirá en describir el problema y proponer varias soluciones que lo resuelvan desde varios puntos de vista, para así poder hacer un análisis comparativo.

4. Codificación y pruebas

   Consistirá en la creación del paquete en R para poder llevar a cabo el análisis de redes Bayesianas y la generación de explicaciones.

   4.1. Implementación de algoritmos de simulación

   4.2. Generación de listas de explicaciones

   4.3. Implementación de algoritmos de optimización

   4.4. Procedimientos de explicación del conocimiento

   4.5. Pruebas del paquete

5. Documento de resultados

   Consistirá en un escrito que recoja los resultados obtenidos de las comparativas entre los distintos métodos codificados para la solución del problema.

   5.1. Resultados de simulación de redes Bayesianas

   5.2. Resultados de optimizaciones

   5.3. Resultados de generación de explicaciones

6. Memoria final

   Consiste en la redacción de una memoria completa de toda la información que se considere necesaria y útil para la descripción y justificación del trabajo desarrollado y de los resultados obtenidos.

7. Seguimiento y control

   Tomará la forma de reuniones semanales con el tutor para orientas en la elaboración del trabajo, además de controlar posibles desvíos de la planificación.

2.1.3. Diagrama de Gannt

2.2. Planteamiento y soluciones

2.2.1. Algoritmos de optimización

Una vez que se tuvo el problema identificado, se estudiaron varias soluciones para tratar de resolver la cuestión de optimización espacial de las TDO. Las propuestas fueron los 3 algoritmos de optimización explicados en el Capítulo 1.2.4:

- Algoritmo de búsqueda tabú
- Algoritmo de recocido simulado
- Algoritmo genético

Estos 3 algoritmos, presentan planteamientos, complejidades, ventajas y desventajas únicas que permitirán explorar la mejor solución. Los 3 algoritmos si comparten el hecho de ser métodos de optimización no exacta.

El algoritmo de **Taboo Search**, o búsqueda tabú, se eligió como una primera aproximación. Este algoritmo, como ya hemos explicado, marca las soluciones visitadas como tabú para así evitar repetirlas y caer en bucles dentro de óptimos locales. Este planeamiento es ligeramente más complejo que una búsqueda heurística básica (siempre visitar el mejor vecino) ya que introduce este vector tabú que almacena posibles soluciones y fuerza un poco más el visitado de todo el espacio de soluciones.

El algoritmo de **Simulated Annealing**, o algoritmo de recocido simulado, resulta especialmente atractivo en nuestro problema y nuestra heurística. Como se ha explicado, este algoritmo utiliza la variable T como indicador de la precisión de la búsqueda, a mayor temperatura menos preciso pero más variedad de soluciones, al reducir la temperatura las soluciones son mas homogéneas pero se busca con mayor precisión. Esto casa muy adecuadamente con nuestra función de similitud G definida en la Ecuación 1.2.4.1. Un valor mayor de G indica una diferencia mayor entre las tablas de decisiones, mientras que una G menor indica mayor similitud.

Así pues, se ha diseñado una función f que define una función lineal entre la medida de temperatura T y la medida de diferencia entre tablas G.

$$f(T_{max}, T_{min}, T_c, G_{max}) = \left(\frac{T_c - T_{min}}{T_{max} - T_{min}}\right) G_{max}$$

Dónde T_{max} y T_{min} son el máximo y mínimo de temperatura, T_c es la temperatura actual y G_{max} es el valor máximo de G entre los posibles vecinos. Con esta función se obtiene una relación entre la temperatura actual y la G a elegir entre los vecinos. Así pues, dado un vector de vecinos cada uno con su G asignada, podemos saber cual sería la medida de diferenciación ideal para corresponder a la temperatura actual del sistema.

Por ejemplo supongamos un vector de bases vecinas B', siendo B_0 la base original, 750 la temperatura actual, una temperatura máxima de 1000º y mínima de 0, y $max(G(B_0, B')) = 10$. Con estos datos podemos calcular la G vecina que deberíamos seleccionar:

$$f(1000, 0, 750, 10) = \left(\frac{750}{1000}\right) 10 = 7,5$$

Por tanto, buscaremos dentro del vector B' aquella base que sea más próxima a $G(B_0, B') = 7,5$.

Por último, el **Algoritmo Genético** se eligió como una alternativa a las búsqueda metaheurísticas, utilizando un paradigma distinto basado en poblaciones de individuos. Este algoritmo es especialmente interesante gracias a la gestión de de grupos de soluciones, permitiendo explorar el espacio de soluciones desde varios puntos al mismo tiempo. Además, los algoritmos genéticos son una buena primera aproximación en este tipo de problemas basados en la optimización combinatoria de permutaciones [14].

En nuestro caso, definimos un individuo como una posible base de una TDO, adaptándose perfectamente a la forma clásica de los individuos en algoritmos genéticos, los cuales suelen definirse como una cadena binaria donde cada atributo representa un gen.

Por otra parte, la **función de selección** se definió de la siguiente forma:

$$P(r) = q(1 - q)^{r-1}$$

Siendo r la posición del individuo en la lista ordenada por "fitness", q un número elegido entre 0 y 1. De esta forma, los mejores individuos tienen mayores probabilidades de ser elegidos, sin monopolizar la función de selección (el mejor individuo será aquel con $r = 1$).

Otro aspecto importante del algoritmo genético es la **función de cruce** entre los individuos. Una vez seleccionada una pareja de individuos, se debe generar un descendiente formado a partir de los genes de sus padres. En un problema tradicional utilizando listas binarias, los hijos son creados directamente usando la mitad de los atributos del padre y la mitad de la madre (si el padre es $P = (1, 1, 1, 1)$ y la madre $M = (0, 0, 0, 0)$, el hijo será $H = (1, 1, 0, 0)$ o $H = (0, 0, 1, 1)$). En nuestro caso, no es tan sencillo, los individuos están formados a partir de aplicar una serie de permutaciones de 2 atributos sobre una base original. De esta forma, existen algunos individuos que

podrían no ser válidos al combinar las dos mitades de los padres. Debemos entonces asegurarnos de que los descendiente generados cumplan la condición de ser bases validas generadas a partir de la base original.

Para lograr el cruce correcto se ha utilizado la función siguiente [14]:

$$a_i = \frac{j * L_m + k * L_p}{L_m * L_p}$$

Para cada tributo (a_i) se calcula su peso, a partir de j y k siendo la posición del atributo a_i en la base del padre B_p y de la madre B_m respectivamente, y L_p y L_m las "fitness" del padre y madre respectivamente.

De esta forma, para cada atributo se obtiene una valor el cual se usará para ordenar los atributos del descendiente.

Por ejemplo, supongamos dos bases padre tal que $B_p = \{A, D, C, B\}$ y $B_m = \{C, B, A, D\}$, $L_p = 15$ y L_{23}. Así pues, el orden de atributos del descendiente se calcula tal que:

$$a_A = \frac{0 * 23 + 2 * 15}{15 * 23} = 0.1538$$
$$a_B = \frac{3 * 23 + 1 * 15}{15 * 23} = 0.4308$$
$$a_C = \frac{2 * 23 + 0 * 15}{15 * 23} = 0.2359$$
$$a_D = \frac{1 * 23 + 3 * 15}{15 * 23} = 0.3487$$

Recordemos, que nuestra "fitness" (L) es la longitud de la KBM2L, por tanto, cuanto menor sea mejor. Así pues, el orden de menor a mayor define la base descendiente, que será: $B_h = \{A, C, D, B\}$.

Otro aspecto importante del algoritmo genético es la función de "fitness" utilizada anteriormente. Como sabemos, la función de distancia G definida en la Ecuación 1.2.4.1 no representa la "optimidad" de una base, sino la similitud entre dos bases. Por esto, se ha de desarrollar algún otro sistema para poder evaluar de forma numérica las bases y así ordenarlas. En este caso, lo buena o mala que es una función se refleja en la cantidad de reglas que tiene la KBM2L formada a partir de una base. Pero realizar el calculo de la KBM2L para cada individuo es inviable (véase Capítulo 1.2.4.1). Así pues, para tratar de reducir el coste computacional utilizaremos una aproximación a este método: para evitar reordenar una tabla con miles o cientos de miles de filas, extraeremos las filas mas representativas de cada tabla y trabajaremos con una simplificación de la tabla.

En concreto, se ha decidido extraer la última fila de cada una de las reglas que forman la KBM2L, es decir, trabajaremos sobre la propia KBM2L. Así pues, reducimos en gran manera el tamaño de la tabla original y podemos trabajar con una aproximación mucho más manejable.

Por ejemplo, supongamos la TDO formada por 3 atributos, cada uno de ellos con 2 estados, y 2 posibles decisiones. Esto resulta en una tabla de 3 columnas y 8 filas. En lugar de reordenar la tabla para cada uno de los individuos, observaremos la KBM2L de la base original y formaremos la nueva tabla con una fila de cada regla de la siguiente KBM2L:

$$< (a_0, b_1, c_0), D_0| < (a_0, b_1, c_1), D_1| < (a_1, b_1, c_1), D_0|$$

A	B	C	Decisión
a_0	b_1	c_0	D_0
a_0	b_1	c_1	D_1
a_1	b_1	c_1	D_0

Cuadro 2.1: Tabla de decisión aproximada

Así pues, la tabla con la que calcularemos la "fitness" de los individuos no tendrá 8 filas, sino 3; véase la Tabla 2.1. En este caso trivial computacionalmente, no tiene sentido trabajar con una aproximación reducida, pero en casos reales, donde podemos tener miles de filas, este sistema reduce considerablemente el tamaño de las tablas.

Un detalle importante que es necesario mencionar es la falta de precisión. Como toda aproximación, favorecemos velocidad sobre precisión y por tanto existe la posibilidad real que el número de reglas que resultan de aplicar una permutación sobre una tabla aproximada no se corresponda con el número de reglas que resultan de aplicar la misma permutación sobre la tabla real.

Para mitigar este problema hay varias alternativas. La más sencilla es seleccionar más filas. Si observamos malos resultados en las aproximaciones de la "fitness" de una base, podemos aumentar el número de filas que se seleccionan por regla, de esta forma, tenemos una imagen más real de la tabla original con un coste computacional superior.

Así pues, el algoritmo genético seguirá la misma estructura que se ha explicado en el Capítulo 1.2.4.3 con 2 modificaciones.

El primer cambio se produce al ordenar los individuos de la población. Ahora, para asignar un valor de "fitness" a cada individuo debemos antes, crear la tabla aproximada a partir de la tabla original, y sobre esta aproximación realizar las permutaciones correspondientes a cada una de las bases a evaluar.

Y en segundo lugar, al final de cada iteración generacional, una vez se ha obtenido una nueva generación, se debe actualizar la tabla original con la mejor de las bases de la generación anterior. Este paso es muy importante, ya que si no actualizamos la tabla original, el algoritmo optimizará el número de reglas de la aproximación, no del problema real. Por tanto, con el objetivo de mantener las aproximaciones más cercanas a la realidad, es necesario actualizar la tabla original cada iteración con el "mejor" de los individuos.

Por último, la **función de mutación** funciona de forma completamente aleatoria. Cada vez que se genera un nuevo individuo, cada uno de los atributos tienen una probabilidad de mutar, es decir, realizar una permutación con otro atributo elegido aleatoriamente. Esta probabilidad viene dada por un parámetro de entrada adaptable. Es importante considerar que esta probabilidad debería ser relativamente baja (cercano a 0.05), ya que si hay demasiadas mutaciones, se puede perder fácilmente los "buenos" atributos de los padres.

2.2.2. Generación de KBM2L y explicaciones

Para la parte de la generación de listas KBM2L se han tomado una serie de consideraciones importantes referentes a la estructura y su funcionalidad.

Primeramente, vamos a añadir un nuevo campo a nuestra base de conocimiento. Dada la naturaleza de las redes Bayesianas, es posible no solo obtener una predicción del estado del nodo "target" sino también la distribución de probabilidades de éste. Es decir, al realizar la predicción de un nodo dada una lista de evidencias, obtendremos dos valores: la decisión tomada, y la probabilidad de dicha decisión.

Esta información adicional resulta muy interesante para nuestros usuarios, ya que la probabilidad asociada a cada decisión es uno de los factores más necesarios en un sistema de soporte. Esta probabilidad es la que nos permite separarnos de los modelos explicables (también llamados de "caja negra") y situarnos claramente en los modelos interpretables.

Resulta muy claro el motivo de la importancia de este valor probabilístico, no es lo mismo una decisión tomada con el $98\,\%$ de certeza, que una tomada con tan solo un $50\,\%$.

Así pues, nuestra estructura de Base de Conocimiento (o Knowledge Base) tendrán una estructura similar a la Tabla 2.2: n columnas con las variables aleatorias del modelo, una columna para la decisión tomada y una última columna con la probabilidad de dicha decisión.

De la misma forma, nuestras estructuras de lista (KBM2L), ahora deberán incorporar esta información. Así pues, cada regla incluirá también un valor de probabilidad.

Otro aspecto a considerar en la conversión de la Base de Conocimiento a Lista de Reglas es la pérdida de información. Dada una KB se debe poder generar su KBM2L, y dada una KBM2L se debe poder reconstruir la KB original.

A pesar de esto, va a ser necesario sacrificar cierta información de la KB. En concreto, la probabilidad asociada a una decisión resulta un dato que inevitablemente vamos a perder. Como ya hemos explicado anteriormente, la estructura de lista permite optimizar espacialmente las tablas de decisiones [14]. Lamentablemente, esta estructura originalmente no contempla la dimensión de probabilidad que se ha decidido añadir a nuestras bases de conocimiento. Así pues, al transformar la KB a lista, vamos a perder las probabilidades únicas de cada ítem de la tabla original.

Con el objetivo de reducir esta pérdida de información se ha implementado una solución parcial: calcular la media de las probabilidades de cada decisión y asignar a las reglas ese valor. Así pues, si queremos optimizar la Tabla 2.2, primero debemos calcular la media de la columna "Prob" por cada regla (en este caso, la probabilidad media de las 3 primeras reglas es $mean(0.7, 0.9, 1) = 0.866$ aproximadamente).

Obteniendo las siguientes reglas:

$$(1, 0) > D_0, 0.866 | (1, 1) > D_1, 1$$

Este hecho resulta especialmente relevante cuando se considera que para trabajar con un KBM2L, uno de los pasos es la reconstrucción de la KB original.

A	B	Decisión	Prob
0	0	D_0	0.7
0	1	D_0	0.9
1	0	D_0	1
1	1	D_1	1

Cuadro 2.2: Tabla ejemplo con probabilidad

A	B	C	D	E	Decisión	Prob
0	0	0	0	0	D_0	0.7
0	0	0	0	1	D_1	0.8
0	0	0	1	0	D_1	0.9
0	0	0	1	1	D_0	0.7

Cuadro 2.3: Tabla devuelta por el método de búsqueda

Uno de los objetivos de este proyecto es la creación de un sistema de "dialógo" con los modelos con el cual el usuario puede obtener información de las Listas de Reglas. En concreto, se han decidido implementar varios métodos:

- El primer método ha de ser capaz de dado un ítem, es decir, una evidencia completa, devolver aquellos atributos más relevantes. Estos se conocen como la parte fija de una explicación. Es esta parte fija, la cual explica la decisión tomada en cualquier instancia [14].

  Por ejemplo, supongamos que queremos conocer porque se ha toma la decisión D_1 en la instancia $(0, 1, 0, 0, 1)$, suponiendo un modelo con 5 nodos binarios. Nuestro método ha de extraer la parte fija de dicha instancia, lo cual implica, devolver aquellos atributos que explican la decisión D_1.

- El segundo método ha de ser capaz de recibir como entrada una evidencia parcial y devolver todos aquellos ítem que coincidan con la evidencia dada.

  Por ejemplo, dada la entrada $(0, 0, 0)$ y la KBM2L de un modelo con 5 nodos, el método devolverá los 4 ítem de la KB que coinciden con esos 3 atributos iniciales. Es decir, se devolverá una tabla similar a la Tabla 2.3.

La implementación de estos métodos requerirá una reconstrucción de la KB. A pesar de esto, como ya hemos explicado, al perder una pequeña parte de información en el proceso de transformación de matriz a lista, la KB reconstruida no será idéntica a la KB original. Así pues, consideramos esta pérdida de información un sacrificio necesario para lograr una reducción espacial de las KB de varios órdenes de magnitud.

Capítulo 3

Resultados

3.1. Algoritmos y eficiencia

En este apartado trataremos los distintos algoritmos implementados, experimentaremos con diferentes combinaciones de parámetros y compararemos los resultados obtenidos. Mencionar que todos los experimentos se han realizado utilizando los métodos del paquete desarrollado, llamado SERB (Síntesis de Explicaciones en Redes Bayesianas).

Realizaremos varios experimentos con el modelo canónico conocido como "asia". Esta red Bayesiana es un ejemplo clásico de modelo discreto y suele ser de los más usados en la literatura de este género. La red está formada por 8 nodos, que en conjunto representan un sistema de decisión para el diagnóstico de cáncer de pulmón y tuberculosis [27].

En primer lugar, tratemos el aspecto de la eficiencia de los algoritmos respecto al número de iteraciones.

Como vemos en las Gráficas 3.1 y 3.1 se comparan el número de reglas obtenidas al incrementar el número de iteraciones. Se ha utilizado un rango de iteraciones desde 1 hasta 1000, con incrementos de 100 y 25 iteraciones respectivamente. Resulta fácil apreciar la naturaleza aleatoria que presentan los algoritmos genético y simulated annealing, lo cual implica que algunas ejecuciones producen picos en el número de reglas.

Cabe destacar que el algoritmo Simulated Annealing ("sima" en nuestro paquete) no tiene un parámetro que defina el número de iteraciones (como podemos ver en A). Para tratar de comparar de esta forma este algoritmo se ha optado por modificar el "cooling rate" o la velocidad de enfriado, de forma que a mayor número de iteraciones, menor sea el parámetro "c". De esta forma se consigue aumentar el número de iteraciones aunque no de forma exacta (resulta complejo definir una temperatura inicial y una velocidad de enfriado que resulte en, exactamente, 1000 iteraciones, por ejemplo).

En concreto, se han utilizado los siguientes parámetros para las funciones:

```
SERB::tabu(KB, i, 10)
SERB::sima(KB, 1000, 0.05, (1000-i)/1000)
```

```
SERB::genetic(KB, 10, 0.05, 0.9, i)
```

Donde i es el número de iteraciones a realizar. Como ya se ha explicado, en el algoritmo de Simulated Annealing se utiliza una transformación en dicho número de iteraciones para aumentar o reducir el parámetro c según se necesite.

En cuanto a la aleatoriedad del algoritmo Simulated Annealing, debemos prestar atención al parámetro k. Este valor, en el ejemplo $k = 0.05$, define la probabilidad de seleccionar una solución parcial no-óptima, con el objetivo de escapar de mínimos locales.

Al algoritmo genético es necesario definir los parámetros que definen la población, la probabilidad de mutación y la función de selección. Tanto la probabilidad de mutación, m, como la probabilidad de selección, q, introducen cierta aleatoriedad al algoritmo. Esto produce, como ya hemos explicado, un comportamiento poco predecible, resultando en los mencionados picos que vemos en las Gráfica 3.1.

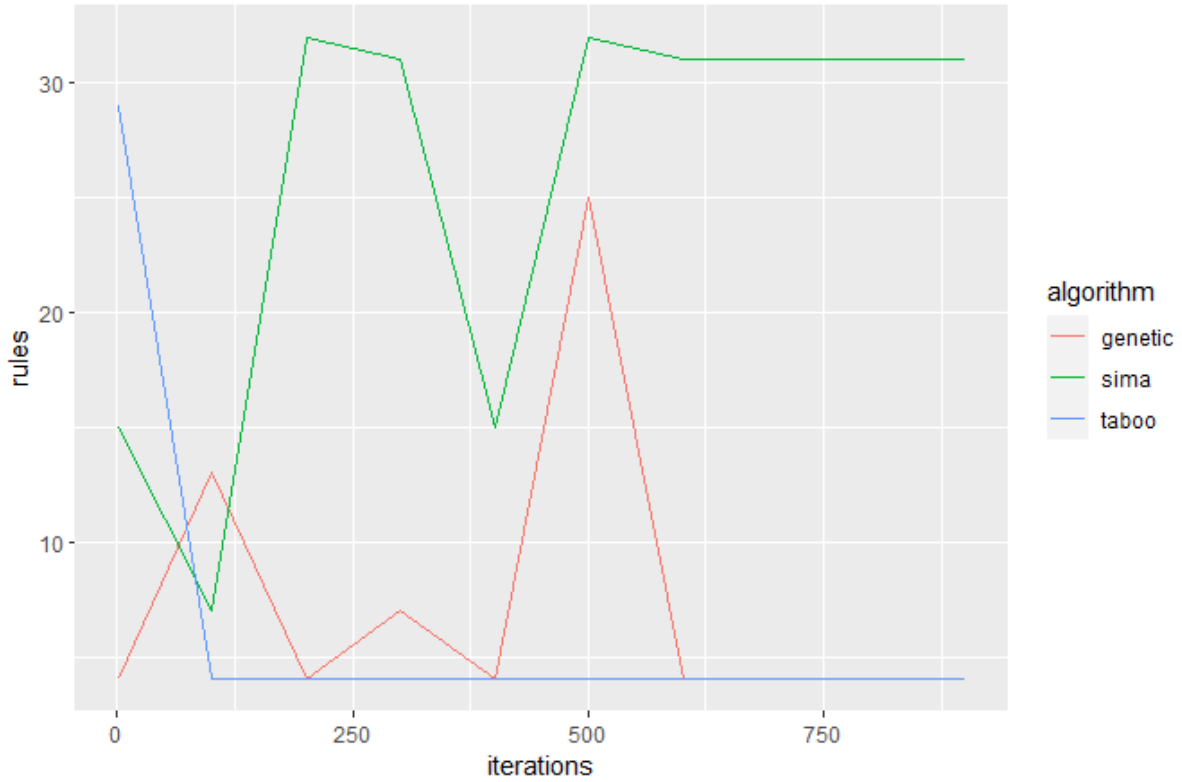

Figura 3.1: Evolución del número de reglas al aumentar las iteraciones en incrementos de 100

3.1.1. Taboo Algorithm

Vamos a estudiar el efecto de los parámetros del algoritmo Tabú.

Como ya hemos podido ver en las Gráficas 3.1 y 3.1, el algoritmo Tabú da buenos resultados de forma consistente. Esto se debe a la combinación de no tener naturaleza aleatoria y a estar trabajando con un modelo relativamente sencillo. El algoritmo Tabú es, objetivamente, el más sencillo de los 3 que se han implementado. Esto

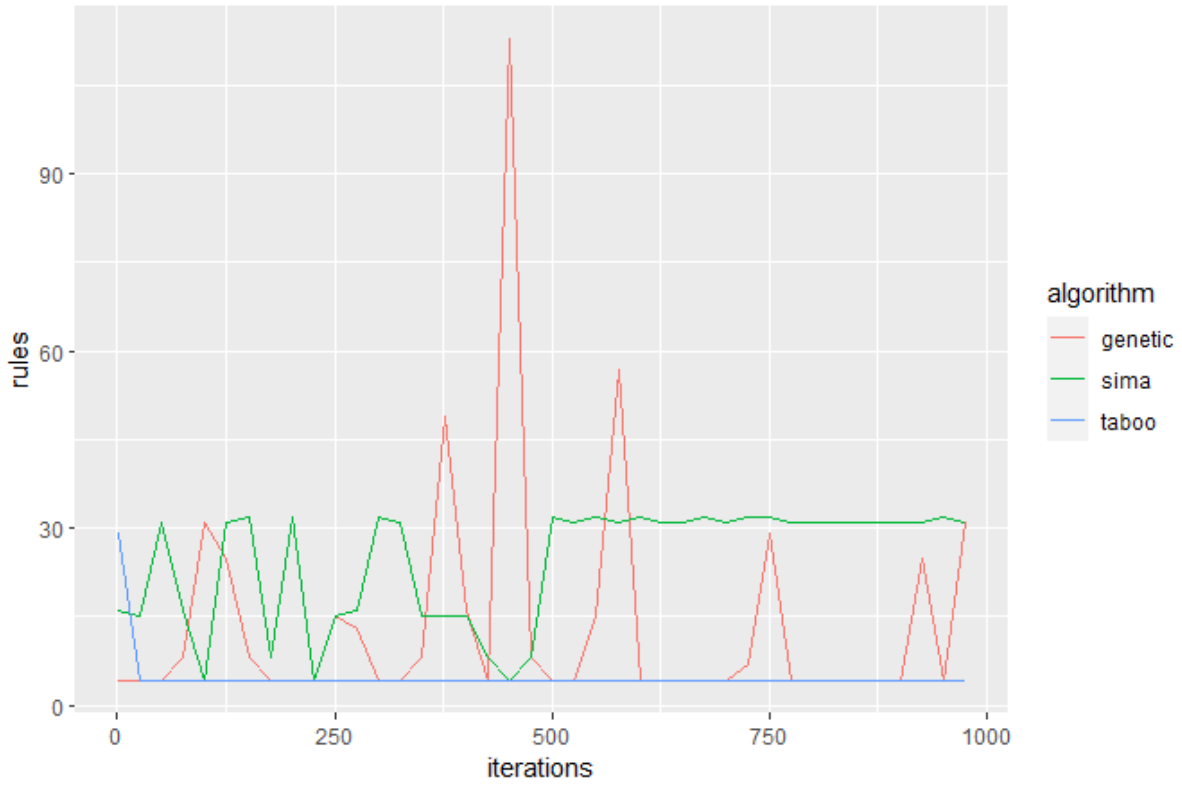

Figura 3.2: Evolución del número de reglas al aumentar las iteraciones en incrementos de 25

hace que para problemas básicos, como es la optimización de las bases del modelo "Asia", funcione muy bien en comparación a los otros algoritmos más complejos. Esto es natural, puesto que para problemas sencillos es apropiado utilizar herramientas sencillas. Pero al mismo tiempo, es de esperar que al aumentar la complejidad de los modelos y el espacio de búsqueda, este sea de los algoritmos que peor actúan.

Aún así, estudiaremos el efecto del parámetro $maxTabu$, que define la longitud máxima de la lista tabú. el funcionamiento de esta lista ya ha sido descrito en el Capítulo 1.2.4.1.

Como se aprecia en la Gráfica 3.1.1, el algoritmo tabú es consistente a lo largo de múltiples ejecuciones ya que no tiene ningún tipo de aleatoriedad. Así pues, resulta fácil concluir que la lista tabú óptima debe ser superior a 6.

Cabe notar, que un problema conocido con este algoritmo es que la longitud de la lista tabú tiene un límite superior un tanto difícil de determinar. Si la lista tabú alcanza una longitud demasiado alta, es posible que se produzca la situación en que todos los vecinos de una base sean considerados tabú y por tanto el algoritmo deba pararse. En nuestro caso, ese límite se encuentra en una longitud superior a 20. Pero esto dependerá del número de vecinos que puede tener una base, y del recorrido por el espacio de soluciones que tome el algoritmo, ya que no tiene porque darse la situación descrita.

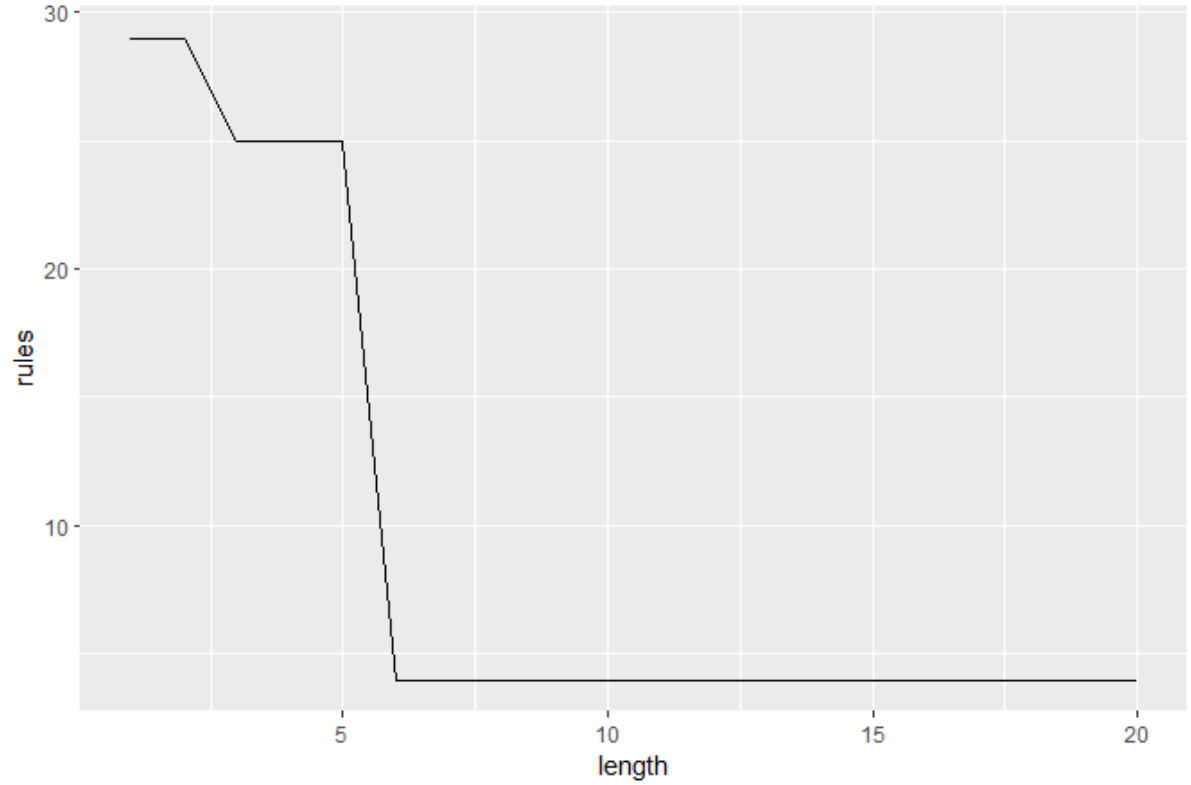

Figura 3.3: Evolución del número de reglas al incrementar la longitud de la lista tabú

3.1.2. Simulated Annealing

Estudiemos el efecto de los parámetros del algoritmo Simulated Annealing.

En primer lugar, vamos a experimentar con el "cooling rate". Este parámetro, como ya hemos explicado en apartado 1.2.4, hace referencia a la velocidad a la que el sistema se enfría.

Como podemos ver en la Gráfica 3.1.2, el efecto que tiene este parámetro es significativo en nuestro algoritmo. Resulta claro que un valor de cooling superior a 0.5 parece dar mejores resultados (un menor número de reglas). Aún así, siguen apareciendo la naturaleza aleatoria del parámetro k, que provoca algunos picos inesperados en la franja entre $0.5 - 0.99$. Notemos que c, nunca puede ser $>= 1$, ya que multiplicar la temperatura actual por 1, resulta en la misma temperatura, y por tanto, el algoritmo no terminaría nunca.

Por otro lado, estudiemos el efecto del parámetro k, la probabilidad aleatoria de seleccionar vecinos no-óptimos. Como se muestra en la Gráfica 3.1.2, k no parece tener un efecto significativo sobre el número de reglas. En concreto, solo parece resultar negativo cuando este parámetro toma valores muy reducidos, cercanos a 0. Esto revela la importancia "oculta" de este parámetro. En aquellos casos dónde (prácticamente) nunca seleccionamos vecinos no-óptimos encontramos malos resultados.

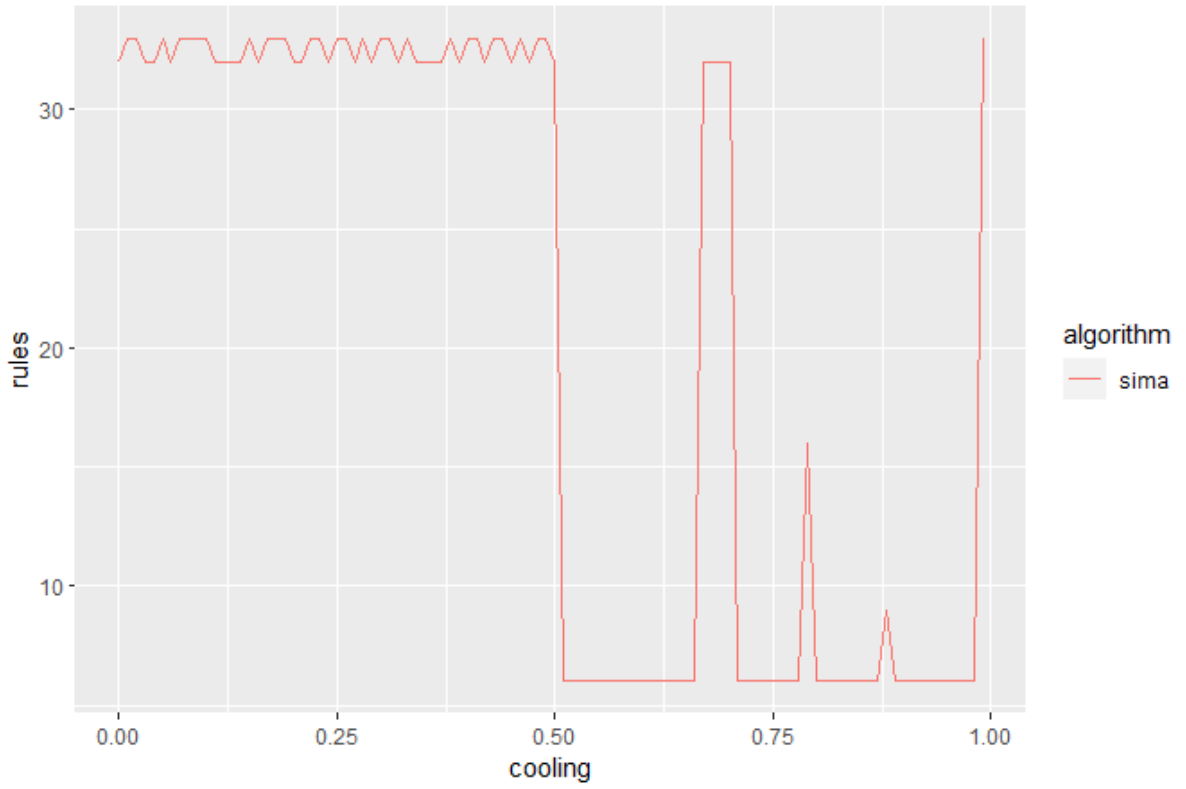

Figura 3.4: Evolución del número de reglas al incrementar el parámetro c

3.1.3. Genetic algorithm

Por otra parte, el algoritmo genético presenta tres parámetros que influyen en el comportamiento del algoritmo.

En primer lugar trataremos con el parámetro de probabilidad de la mutación. Este valor afecta la probabilidad de un atributo cualquiera de un individuo al "nacer" de sufrir una mutación (en este caso, una permutación entre dos atributos aleatorios).

Como se ve en la gráfica 3.1.3, una probabilidad de mutación reducida (entre 0.1 y 0.4) parece devolver mejores resultados. Para tratar de minimizar la aleatoriedad y normalizar los resultados, para cada valor del eje "mutation chance" se han realizado varias ejecuciones y se ha hecho la media del número de reglas devuelto por cada ejecución. Es decir, para $m = 0.05$ se ha ejecutado 5 veces el algoritmo, devolviendo los siguientes número de reglas: $[7, 13, 4, 4, 8]$. De este forma, para el valor de $mutation_chance = 0.05$ marcamos $rules = mean(7, 13, 4, 4, 8) = 7,2$.

En cuanto al parámetro de selección, cuyo funcionamiento se ha descrito con la Sección 2.2.1, podemos ver que con q reducidas (< 0.4) encontramos un gran factor de aleatoriedad, con muchos picos al mismo tiempo que mínimos globales. En cambio, con $q > 0.5$ vemos que los resultados son más consistentes (igual que con la probabilidad de mutación, se ha usado la media de varias ejecuciones para minimizar los efectos de la aleatoriedad).

Por último, el tamaño de la población es un parámetro muy influyente. Debemos entender que, cuanto mas grande sea la población más sencillo resulta que un in-

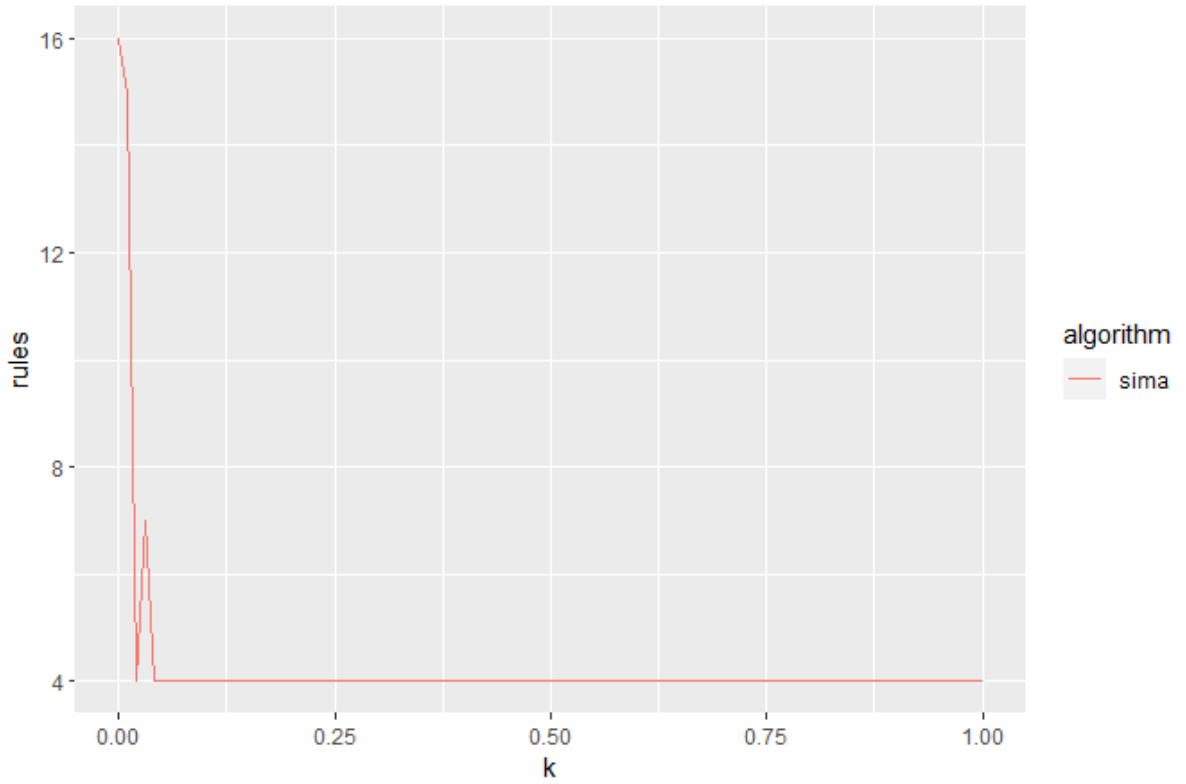

Figura 3.5: Evolución del número de reglas al incrementar el parámetro k

dividuo de la población inicial se encuentre cerca de un mínimo global, y por tanto, sea mucho más sencillo encontrar esa solución. Idealmente utilizaríamos poblaciones inmensas para maximizar la eficiencia, pero en nuestro caso, no contamos ni con la potencia computacional ni el espacio para poder gestionar grandes poblaciones.

Como se ve en la Gráfica 3.1.3, con un modelo sencillo, el algoritmo no requiere de una gran población para buenas soluciones. Es importante destacar que con poblaciones muy reducidas (< 5), el algoritmo parece desempeñar notablemente mal.

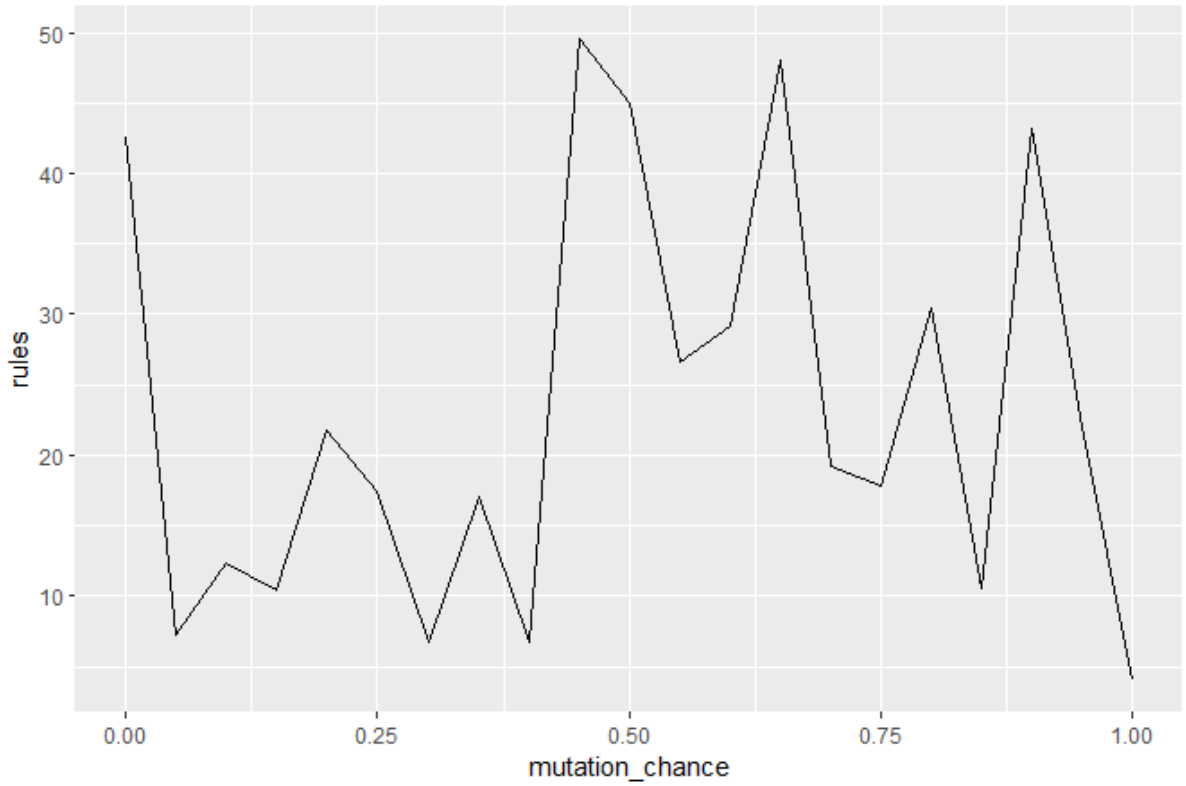

Figura 3.6: Evolución del número de reglas al incrementar la probabilidad de mutación

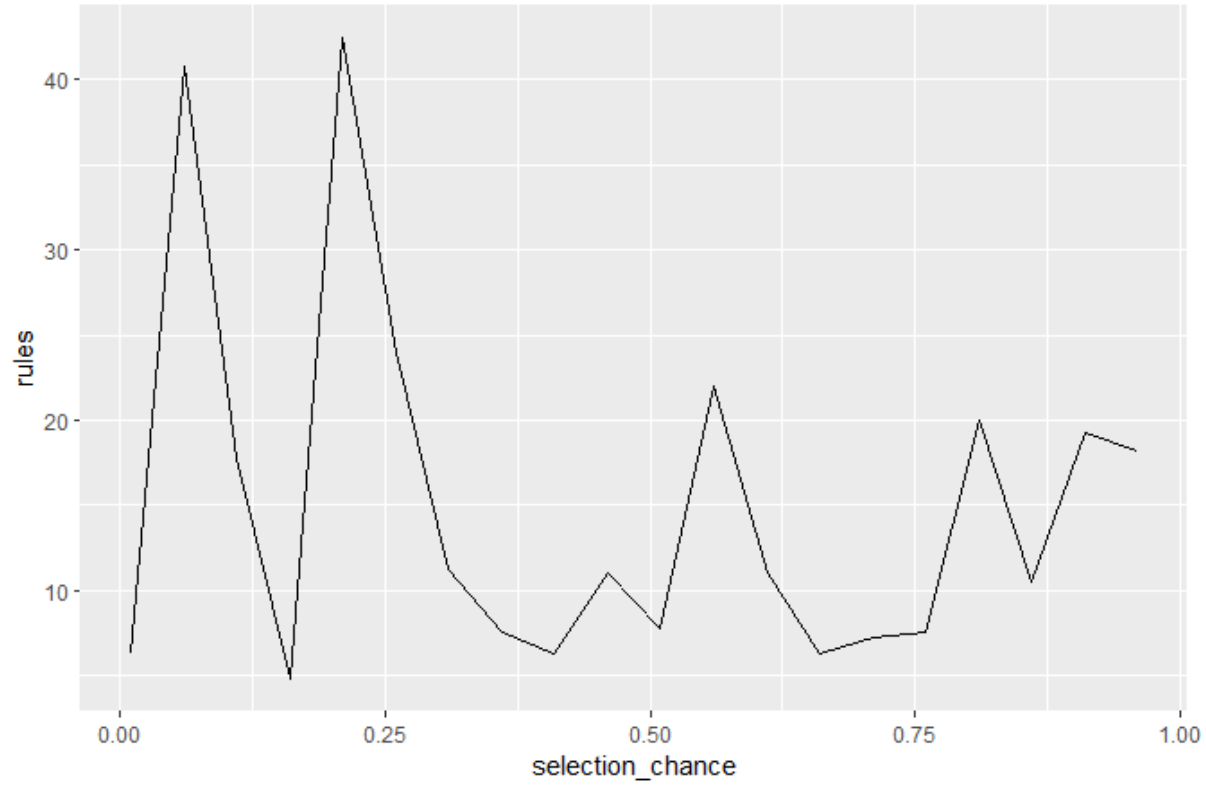

Figura 3.7: Evolución del número de reglas al incrementar la probabilidad de selección

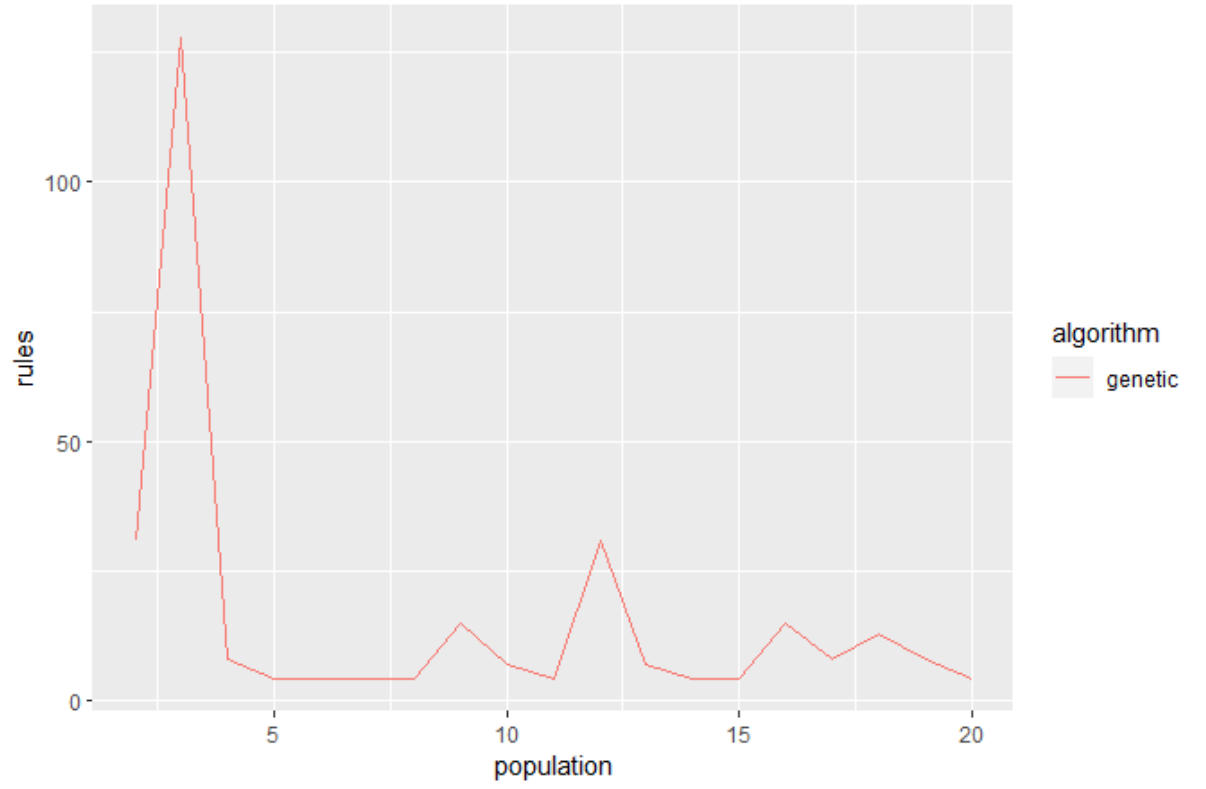

Figura 3.8: Evolución del número de reglas al incrementar la población

Capítulo 4

Conclusiones

4.1. Objetivos

Los objetivos, como fueron definidos en la propuesta del Trabajo de Fin de Máster, los podemos encontrar en el Capítulo 1.3.

Así pues, podemos agrupar los objetivos en tres categorías: El objetivo de investigación, el objetivo de estudio de soluciones y los objetivos de implementación.

4.1.1. Investigación

Se ha investigado sobre las redes Bayesianas y los métodos más actuales [11], y su taxonomía [13]. Se ha investigado en la generación de explicaciones a partir de redes Bayesianas [12] y se han identificado los diferentes métodos de generación de explicaciones en estos modelos [13]. Se ha investigado sobre las definiciones de los sistemas actuales de toma de decisiones [5] y las limitaciones de dichos sistemas [3]. Además, se ha investigado sobre la generación de listas de explicaciones [14] y sobre algoritmos de optimización para estas listas [23]. También ha sido necesario estudiar los orígenes, las ideas principales y el funcionamiento de los distintos algoritmos utilizados, como se ve en el Capítulo 1.2.4. Además de una pequeña investigación en la creación y documentación de paquetes en R [28].

En definitiva, se ha realizado una investigación en profundidad del estado del arte de las redes Bayesianas, los métodos de explicación, la generación de listas de reglas y los algoritmos de optimización.

4.1.2. Soluciones posibles

Se realizó un análisis del problema y de las necesidades a resolver mediante la investigación explicada anteriormente. Y una vez que se identificó el problema como una búsqueda combinatoria de bases óptimas, el siguiente paso fue, en consecuencia, estudiar y elegir soluciones que se adaptaran al problema en cuestión. En particular, de entre todas las implementaciones posibles se eligieron 3 algoritmos de variante complejidad, que permitían encontrar soluciones de forma óptima.

4.1.3. Implementación

Se ha llevado a cabo la implementación de un paquete en R capaz de evaluar redes Bayesianas, generar las bases de conocimiento y listas de explicaciones. Además, en este mismo paquete se han implementado los distintos algoritmos de optimización elegidos, así como métodos para generar un diálogo simple entre el usuario y el modelo.

Todas estas funcionalidades se pueden encontrar definidas en la documentación del Anexo A. Aún así, enumeraremos rápidamente aquellas funciones principales para explicar su funcionamiento:

createKB Esta función toma una red Bayesiana y genera la base de conocimiento (KB) que se deriva de dicho modelo. El objeto de KB es esencial para trabajar con el paquete.

tabu Esta función implementa el algoritmo de la búsqueda Tabú (o Taboo en inglés). Toma como entrada la KB creada, el número de iteraciones y el tamaño máximo de la lista tabú. Devuelve una KB con la base que más reduce el número de reglas.

sima Esta función implementa el algoritmo de recocido simulado (o simulated annealing). Toma como entradas la KB, la temperatura inicial del sistema, la probabilidad aleatoria de movimientos no óptimos y el factor de enfriado. Devuelve la KB optimizada.

genetic Esta función implementa el algoritmo genético el cual toma como entrada la KB, el tamaño de la población, la probabilidad de mutación, la probabilidad de selección y el número de generaciones. Devuelve la KB con la base óptima encontrada.

createKBM2L Dada una base de conocimiento, esta función devuelve la lista que optimiza espacialmente la KB. La idea es obtener la KBM2L de una KB que haya sido optimizada por alguno de los 3 algoritmos anteriores.

query.list Dada una evidencia parcial y la KBM2L devuelve todas aquellas instancias que coinciden con la evidencia dada y la decisión que se toma en cada caso.

fixed Dada una lista de evidencia y la KBM2L este método devuelve la parte fija de la evidencia, es decir, devuelve los atributos que más influyen en la decisión tomada.

4.2. Lineas futuras

En primer lugar cabe destacar la falta de optimización para modelos grande que presenta el paquete desarrollado. Para modelos pequeños o medianos (como “asia”), el modelo es capaz de trabajar de forma rápida con los datos proporcionados. Pero, al incrementar el tamaño de los modelos, ya sea incrementando el propio número de nodos o la cantidad de niveles de cada nodo, los tiempos incrementan demasiado para ser usable. Como ya justificamos en el Capítulo 1.2.4, pequeños incrementos en una o varias dimensiones de los modelos producen un crecimiento exponencial en las estructuras propias con las que se trabaja.

Así pues, una de las líneas futuras que proponemos es expandir la funcionalidad del paquete añadiendo métodos para la optimización de grandes modelos. Esto se puede lograr de diversas forma, desde modificar los propios algoritmos para buscar pequeñas optimizaciones, como añadir nuevas funcionalidades que sean capaces de gestionar estos incrementos en el tamaño.

En concreto, la solución que proponemos se trata de crear nuevos métodos los cuales utilicen los algoritmos implementados como base, pero dividan las bases de conocimiento de gran tamaño es matrices gestionable por nuestro paquete. Mediante la instanciación de la evidencia, se puede reducir el tamaño de cualquier base de conocimiento para que sea tratable, almacenándola y trabajando con ella por partes.

Otro aspecto a desarrollar en más profundidad es el sistema de interacción con el usuario. Nuestro sistema utiliza una funcionalidad muy básica de "conversación" con el usuario. Las preguntas que un usuario puede hacer son reducidas y requieren de conocimiento tanto del funcionamiento de una red Bayesiana como del propio problema.

Una posible mejora sería la implementación de más métodos para interrogar la base de conocimiento, explicaciones adaptables al conocimiento del usuario, explicaciones multi-modales (capaces de combinar texto con información de forma gráfica) o incluso explicaciones en lenguaje natural.

Por último, una linea futura interesante se trata del estudio de sensibilidad de los modelos. En nuestro caso, el estudio de los parámetros del modelo y la sensibilidad de estos ha sido muy reducida. A pesar de esto, estos atributos son muy relevantes a la hora de trabajar con una red Bayesiana y pueden influir en el comportamiento, el estudio y la comprensión del modelo.

El estudio de la sensibilidad de los parámetros de un modelo es especialmente relevante para aquellos problemas que sean críticos (por ejemplo un modelo para diagnosticar pacientes con cáncer). El estudio de la sensibilidad puede servir para verificar la lógica interna del modelo.

Así pues, proponemos una linea futura la cual expanda en la creación de las bases de conocimientos incluyendo los parámetros de los modelos en la propia base de conocimientos. Es decir, dado un modelo, la base de conocimiento no solo representará todas las posibles combinaciones de estados de los nodos, sino que además también cuenta con todas las posibles combinaciones de parámetros del propio modelo. Así pues, al disponer de una KB con esta información añadida, se puede descubrir la sensibilidad de cada uno de los diferentes parámetros, comparando los resultados dados por el modelo con una misma evidencia, pero una secuencia de parámetros distintos.

4.3. Conclusiones personales

4.3.1. Planificación

La planificación del proyecto se definió en el Capítulo 2.1.3. En el diagrama de Gannt se pueden identificar las tareas que se identificaron, su distribución temporal por semanas y los hitos principales (las 2 entregas intermedias y la final).

Esta planificación muestra un desarrollo en cascada, con las tareas organizadas de forma secuencial, sin bucles ni "sprints". Aún así, esto tiene un sentido puramente logístico, ya que al solo ser una persona resulta complicado iterar sobre el trabajo de uno mismo o tratar de paralelizar las tareas. Cabe destacar, el apartado 4.5, el cual sí se lleva a cabo en paralelo al desarrollo de otras actividades. Esto se debe a que el apartado 4.5 corresponde a las pruebas del paquete, resultando más sencillo ir "testeando" al mismo tiempo que se desarrollan el código. De la misma forma, la tarea *5. Documentación de resultados* también se planificó como ir desarrollándola en paralelo.

Como se puede observar en el diagrama de Gannt actualizado (la Figura 4.1) la planificación final no se ha cumplido. En primer lugar, vemos que hasta el mes de marzo la planificación se ha cumplido correctamente. El tiempo para las tareas fue estimado correctamente y, por tanto, la planificación se cumplió.

Por otra parte, a partir del mes de abril empezaron a suceder retrasos. Las tareas de codificación 4.3, 4.4 y 4.5 ocuparon mucho más tiempo del planeado, en algunos casos el tiempo necesario para terminar estas tareas fue más del doble planeado. Afortunadamente, la fecha límite del proyecto también sufrió modificaciones y fue alargada 4 semanas más. Esto ha permitido que, aunque la estimación del trabajo no fuera correcta, no se hayan dado retrasos en las entregas parciales ni finales. Así pues, la planificación ha sufrido una serie de retrasos y modificaciones que han implicado 4 semanas extra de trabajo, pero sin resultar en incumplir los plazos de entrega.

4.3.2. Investigación

Como ya se ha explicado en el Capítulo 4.1, para poder llevar a cabo este trabajo ha sido necesaria una investigación profunda en el estado del arte de la IA, las redes Bayesianas y en la generación de explicaciones.

Desde un punto de vista personal, esta investigación ha permitido adquirir un nuevo punto de vista mucho más informado y actualizado en el sector de los sistemas para la toma de decisiones. Si no se está al corriente del estado del arte resulta fácil presuponer que la industria y el mundo académico priorizan sistemas basados en modelos de caja negra, como pueden ser las redes neuronales o los transformadores. Estos sistemas son muy potentes y dan muy buenos resultados, permitiendo que incluso el público "mainstream" los conozca y los use [29].

Pero basta indagar un poco en el mundo académico para darse cuenta que esto no es así. Existe un gran número de expertos, científicos y divulgadores que llevan años tratando de hacer entender la importancia de los sistemas interpretables en nuestra industria [3]. Los modelos interpretables son una herramienta que presentan muchas ventajas frente a las alternativas más opacas. La fiabilidad y la transparencia que ofrecen este tipo de modelos son extremadamente relevantes en el entorno tecnológico actual.

Así pues, la investigación llevada a cabo ha servido, no solo, para identificar y reconocer los límites de los modelos explicables, sino también para conocer, profundizar y dar a conocer los modelos interpretables y sus fortalezas.

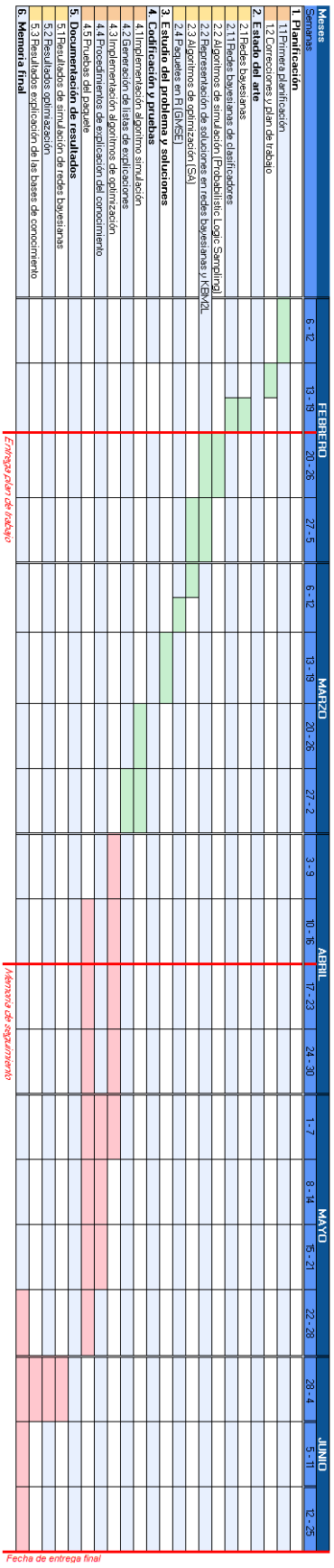

Figura 4.1: Diagrama de Gannt Final

4.4. Análisis de impacto

En el contexto actual, donde la sostenibilidad y el desarrollo responsable se han convertido en prioridades globales, es fundamental analizar cómo los avances en el campo de la IA pueden contribuir a los Objetivos de Desarrollo Sostenible (ODS) establecidos por las Naciones Unidas. Estos ODS abarcan una amplia gama de áreas temáticas, desde la erradicación de la pobreza y el hambre hasta la acción climática y la educación de calidad. Este capítulo se centra en analizar el impacto del TFM en relación con los ODS.

Los Objetivos de Desarrollo Sostenible, o ODS, son un conjunto de objetivos globales que apuntan a erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos. Estos objetivos fueron acordados en 2015 entre los líderes globales con la intención de alcanzarlos en 2030 [30].

El primer ODS que destacamos se trata del *Objetivo 9: Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación*. Este objetivo es especialmente relevante para nosotros, ya que este TFM se enmarca en el contexto de la innovación tecnológica. El desarrollo de herramientas para la toma de decisiones transparentes y fiables resulta un punto clave en el avance de la IA y los sistemas de soporte para expertos.

Además, como ya hemos estado justificando durante el trabajo, el desarrollo de herramientas software transparentes, fiables y carentes de “bias” o prejuicios, representa una sociedad más justa e igualitaria. Los modelos actuales no permiten una visión clara en su funcionamiento interno, escondiendo muchas veces los prejuicios voluntarios o no, de sus creadores. Si para los sistemas críticos se utilizasen modelos transparentes, las herramientas de IA pueden convertirse en un poderoso mecanismo para lograr el *Objetivo 5: Lograr la igualdad entre los géneros y empoderar a todas las mujeres y las niñas* y el *Objetivo 10: Reducir la desigualdad en los países*. Los modelos y las herramientas no-discriminatorias son el primer paso en la sistematización de la igualdad y la justicia.

Bibliografía

[1] R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. <https://www.R-project.org/>.

[2] RStudio Team (2020). RStudio: Integrated Development for R. RStudio, PBC, Boston, MA URL http://www.rstudio.com/

[3] Rudin, C. (2019). *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*. Nat Mach Intell 1, 206–215, 2019. https://doi.org/10.1038/s42256-019-0048-x

[4] Jindal A. (2022). *Misguided Artificial Intelligence: How Racial Bias is Built Into Clinical Models*. Brown Hospital Medicine. doi:10.56305/001c.38021

[5] C. O'Sullivan (2020). *Interpretability in machine learning*, Medium, https://towardsdatascience.com/interpretability-in-machine-learning-ab0cf2e66e1 (accessed Jun. 14, 2023).

[6] Molnar, C. (2023) *Interpretable machine learning: A guide for making Black Box models explainable*. Christoph Molnar: Munich.

[7] Pearl J. (1985). *Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning* (UCLA Technical Report CSD-850017). 7th Conference of the Cognitive Science Society, University of California.

[8] Concha Bielza and Pedro Larrañaga (2014). *Discrete Bayesian network classifiers: A survey*. ACM Comput. Surv. 47, 1, Article 60 (April 2014), 43 pages. DOI: http://dx.doi.org

[9] Yuan, Changhe & Lim, Heejin & Lu, Tsai-Ching (2011). *Most Relevant Explanation in Bayesian Networks*. J. Artif. Intell. Res. (JAIR). 42. 309-352. 10.1613/jair.3301.

[10] Kevin P. Murphy (2012). *Machine Learning A Probabilistic Perspective*. The MIT Press. Cambridge, Massachusetts. London, England

[11] Conor Hennessy, Alberto Bugarín, and Ehud Reiter (2020). *Explaining Bayesian Networks in Natural Language: State of the Art and Challenges*. In 2nd Workshop on Interactive Natural Language Technology for Explainable Artificial Intelligence, pages 28–33, Dublin, Ireland. Association for Computational Linguistics.

[12] Carmen Lacave and Francisco J. Díez. (2002). *A review of explanation methods for Bayesian networks*, The Knowledge Engineering Review, vol. 17, no. 2, pp. 107–127, 2002.

[13] Derks I.P., de Waal A. (2020) *A Taxonomy of Explainable Bayesian Networks*. In: Gerber A. (eds) Artificial Intelligence Research. SACAIR 2021. Communications in Computer and Information Science, vol 1342. Springer, Cham.

[14] Fernández del Pozo, J.A., Bielza, C. and Gómez, M. (2003) *A list-based compact representation for large decision tables management*, European Journal of Operational Research.

[15] Fred Glover (1986). *Future paths for integer programming and links to artificial intelligence, Computers & Operations Research*, Volume 13, Issue 5, 1986, Pages 533-549, ISSN 0305-0548.

[16] Fred Glover (1989). *Tabu Search – Part 1*. ORSA Journal on Computing. Pages 190–206.

[17] Hamming, R. W. (1950). *Error detecting and error correcting codes*. The Bell System Technical Journal. 29 (2): 147–160. doi:10.1002/j.1538-7305.1950.tb00463.x

[18] Kirkpatrick, S.; Gelatt Jr, C. D.; Vecchi, M. P. (1983). *Optimization by Simulated Annealing*. Science. 220 (4598): 671–680.

[19] Pincus, Martin (1970). *A Monte-Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems*. Journal of the Operations Research Society of America. 18 (6): 967–1235.

[20] Khachaturyan, A.: Semenovskaya, S.: Vainshtein B., Armen (1979). *Statistical-Thermodynamic Approach to Determination of Structure Amplitude Phases*. Soviet Physics, Crystallography. 24 (5): 519–524.

[21] Cerny, V. (1985) *Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm*. Journal of Optimization Theory and Applications, 45, 4151.

[22] Fleischer, Mark. (1996). *Simulated annealing: Past, present, and future*. Proceedings of the 27th conference on Winter simulation. 155-161.

[23] Albert Y. Zomaya and Rick Kazman (2010). *Simulated annealing techniques*. Algorithms and theory of computation handbook: general concepts and techniques. Chapman & Hall/CRC, 33.

[24] Rechenberg, I. (1965) *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment Library Translation.

[25] Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[26] K. De Jong (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, University of Michigan, Ann Arbor.

[27] S. Lauritzen, D. Spiegelhalter (1988). *Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems (with discussion)*. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 50(2):157-224.

[28] Wickham, H. (2023) *R packages: Organize, test, document, and share your code*. O'REILLY MEDIA.

[29] Vincent, J. (2022). *Chatgpt proves AI is finally mainstream - and things are only going to get weirder*. The Verge. https://www.theverge.com/2022/12/8/23499728/ai-capability-accessibility-chatgpt-stable-diffusion-commercialization (accessed Jun. 16, 2023)

[30] Portada - Desarrollo Sostenible. United Nations. Available at: https://www.un.org/sustainabledevelopment/es/ (Accessed: 24 June 2023).

Anexo

Apéndice A

Documentación SERB

El código se encuentra disponible en el repositorio de GitHub en la siguiente dirección: `https://github.com/TomeuRamis/SERB`

Package 'SERB'

June 15, 2023

Type Package

Title Síntesis de Explicaciones en Redes Bayesianas

Version 1.0.0

Author Bartomeu Ramis

Maintainer Bartomeu Ramis <bartomeuramis99@gmail.com>

Description A simple package that allows for the creation of Knowledge Bases from Bayesian Networks, their spacial optimisation and quering.

License /

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Imports dplyr, bnlearn

Depends R (>= 2.10)

R topics documented:

count_rules *Count Rules of Knowledge Base*

Description

Count the number of independent "Rules" that define the Knowledge Base.

Usage

```
count_rules(KB, verbose)
```

Arguments

KB	Table structure of a Knowledge Base. The two last columns must be Decision and Prob.
verbose	Boolean value. If TRUE prints out the ruleset in indices notation, if FALSE does not print any information.

Details

A Rule of a KB is defined as the last item of a adjacent set of evidences with the same Decision. The fewer rules a KB has, the more optimal its space optimization is.

Value

Returns an the number of rules that define the KB.

References

Fernández del Pozo, J.A., Bielza, C. and Gómez, M. (2003) *A list-based compact representation for large decision tables management*, European Journal of Operational Research.

Examples

```
#Create the associated KB
KB <- createKB(bn, "tub")
count_rules(KB, FALSE)
```

createKB *Create Knowledge Base*

Description

Create a Knowledge Base from a given Bayesian Network and a Target node from that same network. The table structure created contains two new columns: Decision and Prob.

Usage

```
createKB(BN, node)
```

Arguments

BN	Must be a Discrete Bayesian Network created with the "bnlearn" package.
node	String name of the Target Node we want to study.

Details

This function creates a table with all possible combinations of input values for the target node. From this table we query the model repeatedly, saving the Decision given and the Probability of said Decision. Each Decision and Probability is saved alongside the evidence that generated it, and it is return in the structure of a Table, called Knowledge Base.

Value

A Table with all possible evidence combinations and the value and probability of the Target node for each instance.

References

Fernández del Pozo, J.A., Bielza, C. and Gómez, M. (2003) *A list-based compact representation for large decision tables management*, European Journal of Operational Research.

Examples

```
#Create the KB from a preloaded Bayesian Network (bn)
KB <- createKB(bn, "tub")
```

---

createKBM2L *Creation of Knowledge Base Matrix To List structure*

Description

Creates a KBM2L list from a KB structure. A KBM2L is a list that reduces the space needed to store a KB structure with, almost, no information loss.

Usage

```
createKBM2L(KB)
```

Arguments

KB	Knowledge Base table from which to create the KBM2L.

Value

A list that contains the rules that form the KB entered as input.

Examples

```
#Create the associated KB
KB <- createKB(bn, "tub")
#Optimize KB
optimalKB <- genetic(KB, 20, 0.05, 0.9, 100)

KBM2L <- createKBM2L(optimalKB)
```

fixed *Identify the Fixed of a Rule*

Description

Identifies the Fixed part of a rule. The Fixed part explains the parameters that are most relevant to the decision chose. So this function returns the Most Relevant attributes of a set of evidence.

Usage

```
fixed(kbm2l, evidence)
```

Arguments

kbm2l	KBM2L structure to query.
evidence	Must be a vector containing evidence for all the atributes of a KB, including the Decision and Prob columns.

Value

Returns the list of the s that comprise the Fixed Part of a rule.

Examples

```
#Create the associated KB
KB <- createKB(bn, "tub")
#Optimize KB
optimalKB <- genetic(KB, 20, 0.05, 0.9, 100)
#Create the KBM2L
KBM2L <- createKBM2L(optimalKB)

#Identify the fixed part of the first item of a KB
fix1 <- fixed(KBM2L, optimalKB[1,])

#Identify the fixed part of the second rule (this time using the KBM2L structure)
fix2 <- fixed(KBM2L, KBM2L[2,])
```

genetic *Genetic Algorithm*

Description

Genetic algorithm that searches for the Base that optimizes the number of rules of a KB.

Usage

```
genetic(KBo, np, m, q, t)
```

Arguments

KBo	Table structure of the KB to optimize.
np	(Integer) Number of individuals in the population.
m	(Double) Mutation chance for each new individual. Should be between 0 and 1.
q	(Double) Selection chance for a individual. Should be between 0 and 1.
t	(Integer) Number of generations to iterate through.

Value

Returns the KB that has been found that has the less number of rules, and therefore is the more space optimal.

Examples

```
#Create the associated KB
KB <- createKB(bn, "tub")

optimalKB <- genetic(KB, 20, 0.05, 0.9, 100)
```

---

query.list *Query the KBM2L with partial evidence*

Description

Make a query to the KBM2L structure in order to obtain all possible instances of a partial (or complete) evidence. Given some partial evidence, this function return all possible alternatives with their associated decision and probability. Using complete evidence is a particular case where the there is only one possible decision.

Usage

```
query.list(KBM2L, evidence)
```

Arguments

KBM2L	KBM2L structure to query.
evidence	Vector of partial evidence to search in the KB structure.

Value

Returns a list of the possible instances of the evidence given with their associated Decision and Probability.

Examples

```
#Create the associated KB
KB <- createKB(bn, "tub")
#Optimize KB
optimalKB <- genetic(KB, 20, 0.05, 0.9, 100)
#Create the KBM2L
KBM2L <- createKBM2L(optimalKB)

#Create some partial evidence
evidence <- setNames(c("yes", "yes", "no"), c("Tub", "Asia", "Smoke"))

query.list(KBM2L, evidence)
```

---

readKB *Read Knowledge Base*

Description

Load the Knowledge Base structure from a text file.

Usage

```
readKB(file)
```

Arguments

file	String name of the file that contains the KB.

Value

Returns the Table structure saved in the file.

Examples

```
#File should be a KB previously saved
#KB <- readKB("output.txt")
```

---

sima *Simulated Annealing algorithm*

Description

Simulated Annealing search algorithm that searches for the Base that optimizes the number of rules of a KB.

Usage

```
sima(KB, T0, k, c)
```

Arguments

KB	Table structure of the KB to optimize.
T0	(Integer) Initial temperature of the system.
k	(double) Random chance of choosing a non-optimal partial solution. Should be between 0 and 1.
c	(double) Rate of cooling of the system. Should be between 0 and 1.

Value

Returns the KB that has been found that has the less number of rules, and therefore is the more space optimal.

Examples

```
#Create the associated KB
KB <- createKB(bn, "tub")

optimalKB <- sima(KB, 1000, 0.1, 0.9)
```

---

tabu *Taboo Algorithm*

Description

Taboo search algorithm that searches for the Base that optimizes the number of rules of a KB.

Usage

```
tabu(KB, m, maxtabu)
```

Arguments

KB	Table structure of the KB to optimize.
m	(Integer) Number of iterations to search for.
maxtabu	(Integer) Max length of the Taboo list. The longer the list is made, the more memory it has to avoid previous solutions. This parameter has a maximum length defined by the number of attributes of a KB. If the taboo list is too long, at some point all neighbors of a base will be considered taboo and the program will throw out an error.

Value

Returns the KB that has been found that has the less number of rules, and therefore is the more space optimal.

Examples

```
#Create the associated KB
KB <- createKB(bn, "tub")

optimalKB <- tabu(KB, 100, 10)
```

writeKB *Write Knowledge Base*

Description

Write a Knowledge Base structure to an external file.

Usage

```
writeKB(file, KB)
```

Arguments

KB	Knowledge Base to write to file.
file	String name of the file name where you want to store the KB. If the file does not exist it creates it.

Value

Does not return anything.

Author(s)

Bartomeu Ramis Tarragó

Examples

```
#Create a example KB
KB <- data.frame(var1=c(1, 3, 3, 4, 5),
                 var2=c(7, 7, 8, 3, 2),
                 var3=c(3, 3, 6, 6, 8),
                 var4=c(1, 1, 2, 8, 9))
#Save it to file_output.txt
writeKB("file_output.txt", KB)
```

Index