PORTADA ...

Bla bla bla,

Bla bla bla,

Bla bla bla,

Bla bla bla

Frase de libro,

Agradecimientos

Agradecimientos

dadsadsad

as dasdas d sa dsad

sd as dsa d

adsasda

Texto ...

Índice general

Aportaciones

A continuación se expone una lista de los trabajos que han sido publicados como resultado de la investigación llevada a cabo a lo largo de esta tesis doctoral. Estas publicaciones avalan el trabajo realizado poniendo de manifiesto tanto su interés como su validez científica.

(i) Fernando Benito-Picazo, Manuel Enciso, Carlos Rossi, Antonio Guevara. *Enhancing the conversational process by using a logical closure operator in phenotypes implications*. Mathematical Methods in the Applied Sciences, John Wiley & Sons Ltd. DOI: 10.1002/mma.4338, 2017.

Factor de impacto en J.C.R. 2016: 1,017. Posición 108 de 255 (Q2) en la categoría: 'Mathematics, Applied'.

(ii) Fernando Benito-Picazo, Pablo Cordero, Manuel Enciso, Ángel Mora. *Reducing the search space by closure and simplification paradigms. A parallel key finding method*. The Journal of Supercomputing, Springer. DOI: 10.1007/s11227-016-1622-1, 2016.

Factor de impacto en J.C.R. 2016: 1,326. Posición 52 de 104 (Q2) en la categoría: 'Computer Science, Theory & Methods'.

(iii) Fernando Benito-Picazo, Pablo Cordero, Manuel Enciso, Ángel Mora. *Closed sets enumeration: a logical approach*. Proceedings of the Seventeenth International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2017. Cádiz, Spain, July 4-8, pp. 287-292, ISBN: 978-84-617-8694-7.

(iv) Fernando Benito-Picazo, Manuel Enciso, Carlos Rossi, Antonio Guevara. *Conversational recommendation to avoid the cold-start problem.* Proceedings of the Sixteenth International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2016. Cádiz, Spain, July 4-8, pp. 184-190, ISBN: 978-84-608-6082-2.

(v) Fernando Benito-Picazo, Pablo Cordero, Manuel Enciso, Ángel Mora. *Keys for the fusion of heterogeneous information.* Proceedings of the Fifteenth International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2015. Cádiz, Spain, July 6-10, pp. 201-211, ISBN: 978-84-617-2230-3.

Capítulo 1

Introducción

La gestión de la información es uno de los pilares esenciales de la Ingeniería Informática. No es de extrañar, por tanto, que conforme un amplio campo de investigación y conocimiento donde diversas disciplinas como las Matemáticas, la Lógica y la Ingeniería actúen conjuntamente para alcanzar mejores sinergias.

Dentro de esta filosofía y en aras de mejorar el funcionamiento de dos tipos de sistemas de información con son las bases de datos y los sistemas de recomendación, en este trabajo de investigación nos vamos a centrar en el Análisis Formal de Conceptos (FCA, por sus siglas en inglés: *Formal Concept Analysis*); concretamente, en la utilización de dos de sus conceptos fundamentales: los retículos de conceptos y los sistemas de implicaciones. La gestión inteligente de estos elementos mediante técnicas lógicas y computacionales confieren una alternativa para superar obstáculos que podemos encontrar a la hora de trabajar con los sistemas de información mencionados. Comenzamos pues introduciendo brevemente FCA.

Podemos considerar FCA como una teoría matemática y una metodología para derivar una jerarquía de conceptos a partir de una colección de objetos y las relaciones que verifican. De esta forma, el propósito es poder representar y organizar la información de manera más cercana al pensamiento humano sin perder rigor científico. En este sentido se enmarca la cita de

Rudolf Wille: "*El objetivo y el significado del FCA como teoría matemática sobre conceptos y sus jerarquías es apoyar la comunicación racional entre seres humanos mediante el desarrollo matemático de estructuras conceptuales apropiadas que se puedan manipular con la lógica.*"

El término FCA fue acuñado por Wille en 1984 culminando años más tarde con la publicación más citada al respecto en colaboración con Bernhard Ganter [43]. Desde entonces FCA se ha aplicado con éxito en diferentes disciplinas de la Ciencia, como por ejemplo: minería de datos, biología celular [38], genética [64], economía, ingeniería del software [114], medicina [88], derecho [82], gestión de la información [98], etc.

La motivación principal de FCA aboga por representar conjuntos de objetos y atributos por medio de tablas de datos. Estas tablas se denominan contextos formales y representan las relaciones binarias entre esos objetos y atributos. Principalmente, existen dos formas básicas para representar el conocimiento: los retículos de conceptos y los conjuntos de implicaciones.

Desde hace años, existen en la literatura estudios [68] donde se han investigado y comparado diferentes algoritmos para obtener el retículo de conceptos a partir del conjunto de datos (en adelante *dataset* por su nomenclatura habitual en el campo). Sin embargo, debido a que el tamaño del retículo de conceptos es, en el peor de los casos, $2^{min(|G|,|M|)}$, siendo G el conjunto de objetos y M el conjunto de atributos, el coste computacional de los métodos encargados de su construcción constituye una limitación de la aplicación de FCA.

Por otro lado tenemos el conjunto de implicaciones. Las implicaciones pueden considerarse *grosso modo* como reglas del tipo *si-entonces*, o en inglés: *if-then rules*, cuya noción principal es un concepto muy intuitivo: cuando se verifica una premisa, entonces se cumple una conclusión. Esta idea básica se refleja en numerosos campos de conocimiento bajo diferentes nombres. Así, en base de datos relacionales se denominan dependencias funcionales [23], en FCA son implicaciones [43], en programación lógica son reglas lógicas de programación [90]. No obstante, al igual que en el caso del retículo de conceptos, también existen ciertas desventajas a la hora de trabajar con implicaciones, de hecho, la propia extracción del conjunto

completo de implicaciones de un dataset es una tarea que presenta una complejidad exponencial [25].

En cualquier caso, ambas formas (retículos e implicaciones) son capaces de representar la misma información contenida en un dataset y es posible pasar una representación a la otra. Ahora bien, trabajar con conjuntos de implicaciones nos permite aplicar técnicas automáticas basadas en la lógica. Con esta última mención, llegamos al momento donde podemos avanzar el propósito principal de esta tesis doctoral, que principalmente consiste en aplicar mecanismos lógicos sobre conjuntos de implicaciones que nos permitan desarrollar métodos para realizar un tratamiento eficiente de la información.

En virtud de ello, en este trabajo de investigación nos vamos a centrar en la utilización del conjunto de implicaciones que se verifique en un dataset como semilla para los métodos de tratamiento eficiente de la información basados en la lógica que se han desarrollado. La aproximación a través de la lógica es posible gracias a sistemas axiomáticos válidos y completos como los Axiomas de Armstrong [3] y la Lógica de Simplificación [86] (SL por sus siglas en inglés: *Simplification Logic*), entre otros.

Estos métodos aplicados sobre conjuntos de implicaciones nos proporcionan la base para trabajar en esta tesis doctoral fundamentalmente sobre los siguientes tres campos de conocimiento: claves minimales, generadores minimales y sistemas de recomendación conversacionales.

Antes de entrar con mayor detalle en los anteriores tres campos, es necesario hacer una importante declaración previa. Tal y como se ha mencionado anteriormente, vamos a trabajar con el conjunto de implicaciones que se verifican en un dataset. Sin embargo, hay que dejar claro que no es competencia de este trabajo el estudiar técnicas para la extracción de estas implicaciones (lo cual es más una tarea de minería de datos), sino que la intención es partir del punto en el que ya contamos con el conjunto de implicaciones para trabajar con él. A este respecto, podemos mencionar los trabajos más citados en la literatura en relación a la extracción del conjunto de implicaciones a partir de datasets [57, 133, 134]. Son trabajos de suma importancia desde el punto de vista teórico pero también desde el punto de

visto práctico, pues incluyen las implementaciones de las aplicaciones que realizan la extracción de las implicaciones.

Sin perjuicio de lo anterior, dado que hemos llegado a crear datasets propios sobre los que poder realizar experimentos como veremos más adelante, si bien no entramos en las técnicas de extracción de implicaciones que utilizan esas aplicaciones, sí hemos tenido que convertirnos en usuarios de estas aplicaciones para poder obtener el conjunto atributos e implicaciones que se verifican. Dicho esto, retomamos el texto pasando a introducir los tres campos de aplicación donde hemos utilizado los conjuntos de implicaciones.

Claves Minimales

En primer lugar, se presenta el tema de las claves minimales. El concepto de clave es fundamental en cualquier modelo de datos, incluyendo el modelo de datos relacional de Codd [23]. Una clave de un esquema relacional está compuesta por un subconjunto de atributos que representan el *dominio* de una determinada función cuya *imagen* es la totalidad del conjunto de atributos. Estas funciones se pueden representar por medio de Dependencias Funcionales (FD, por sus siglas en inglés: *Functional Dependencies*) que especifican una relación entre dos subconjuntos de atributos, e.g. A y B, asegurando que para cualesquiera dos tuplas de una tabla de datos, si verifican A, entonces también verifican B. Nótese el cambio de denominación de implicación a dependencia funcional por estar en el entorno de los sistemas de base de datos relacionales.

La identificación de las claves minimales de una determinada relación es una tarea crucial para muchas áreas de tratamiento de la información: modelos de datos [112], optimización de consultas [65], indexado [80], etc. El problema consiste en el descubrimiento de todos los subconjuntos de atributos que componen una clave minimal a partir de un conjunto de FD que se cumplen en un esquema de una tabla del modelo relacional.

Las claves no sólo son parte fundamental a considerar durante la fase de diseño de sistemas de base de datos relacionales, sino que también se

consideran una poderosa herramienta para resolver multitud de problemas referentes a diversos aspectos del tratamiento de la información. Como muestras de la relevancia de este problema, encontramos numerosas citas en la literatura, entre las que podemos destacar las siguientes. En [113], los autores afirman que: "*identification of keys is a crucially important task in many areas of modern data management, including data modeling, query optimization (provide a query optimizer with new access paths that can lead to substantial speedups in query processing), indexing (allow the database administrator to improve the efficiency of data access via physical design techniques such as data partitioning or the creation of indexes and materialized views), anomaly detection, and data integration*". Más recientemente, en referencia a áreas emergentes como el *linked-data*, en [95], los autores delimitan el problema manifiestando: *"establishing semantic links between data items can be really useful, since it allows crawlers, browsers and applications to combine information from different sources."*

En la actualidad, existen diversos algoritmos capaces de resolver el problema de averiguar las claves minimales utilizando diferentes técnicas clásicas como veremos en el Capítulo 3, sin embargo, ya adelantamos que para averiguar el conjunto de claves minimales, en este trabajo nos centraremos en aquellos algoritmos guiados por la lógica, y más concretamente, por aquellos basados en el paradigma de Tableaux [87, 102].

El problema de estos métodos de búsqueda de claves surge a la hora de tratar con grandes cantidades de información ya que el procedimiento de construcción del Tableaux produce una explosión tal del árbol del espacio de búsqueda que nos lleva a sobrepasar las capacidades de la máquina, incluso para problemas pequeños como se ha demostrado en [26].

Sin embargo, una propiedad muy interesante de los métodos basados en Tableaux es su generación de subproblemas independientes los unos de los otros a partir del problema original. De esta forma, el uso del paradigma de Tableaux nos permite abordar estos problemas aplicando técnicas de supercomputación o de computación de alto rendimiento (HPC, por sus siglas en inglés: *High Performance Computing*) mediante las cuales seremos capaces de tratar con grandes cantidades de información y obtener resulta-

dos en tiempo razonable. Entraremos más en detalle de estas técnicas, sus implementaciones y sus resultados en el Capítulo 3.

Para esta labor de computación de alto rendimiento, hay que indicar que a lo largo de este trabajo de investigación se han adquirido y aplicado las competencias necesarias para poder trabajar en entornos de supercomputación; en concreto, durante los últimos años, se ha trabajado fervientemente con el Centro de Supercomputación y Bioinnovación de la Universidad de Málaga[1]. La posibilidad de tratar con este Centro nos ha proporcionado dos beneficios fundamentales: por un lado, se ha alcanzado una elevada pericia para trabajar en entornos de HPC y para realizar implementaciones que aprovechen una alta cantidad de recursos, y por otro lado, se han podido obtener resultados empíricos sobre experimentos utilizando estrategias de paralelismo, que han desembocado en contribuciones científicas [10, 13] y que habría sido imposible conseguir en la actualidad sin contar con tales recursos computacionales.

Generadores Minimales

Como se ha mencionado anteriormente, una forma de representar en FCA el conocimiento es el retículo de conceptos. Esta representación otorga una visión global de la información con un formalismo muy fuerte, abriendo el puerta para utilizar la teoría de retículos como una metateoría para gestionar la información [15].

Los conjuntos cerrados son la base para la generación del retículo de conceptos ya que éste puede ser construido a partir de los conjuntos cerrados, considerando la relación de subconjuntos como la relación de orden. En este punto nace el concepto de generadores minimales como aquellas representaciones de los conjuntos cerrados que no contengan información supérflua, es decir, representaciones canónicas de cada conjunto cerrado [43].

Los conjuntos cerrados junto con los generadores minimales son esenciales para obtener una representación completa del conocimiento en FCA,

[1] http://www.scbi.uma.es/

pero no sólo son interesantes desde un punto de visto teórico. La importancia de los generadores minimales puede apreciarse claramente a través de citas tales como: *"Minimal generators have some nice properties: they have relatively small size and they form an order ideal"*, existente en importantes estudios como el de Poelmans et al. [96] o [99]. Además, los generadores minimales se han usado como punto clave para generar bases, las cuales constituyen una representación compacta del conocimiento que facilita un mejor rendimiento de los métodos de razonamiento basados en reglas. Missaoui et al. [83,84] presentan el uso de generadores minimales para calcular bases que impliquen atributos positivos y negativos cuyas premisas son generadores minimales.

La contrapartida es que la obtención de todos los conjuntos cerrados y sus respectivos generadores minimales es un problema con complejidad exponencial. Sin embargo, dado que nuestro objetivo es explotar las posibilidades de operar con conjuntos de implicaciones, en este trabajo vamos a combinar ambos aspectos enumerando todos los conjuntos cerrados a partir de un conjunto dado de implicaciones. De hecho, iremos un paso más allá, calculando no sólo todos los conjuntos cerrados, sino que para cada uno de ellos produciremos sus respectivos generadores minimales. El método propuesto es una continuación del trabajo llevado a cabo en [29] sobre un método basado en la SL_{FD}. Ese método opera sobre un conjunto de implicaciones aplicando un conjunto de reglas de inferencia construyendo un espacio de búsqueda en forma de árbol.

Respecto a la producción de los generadores minimales, el problema de fondo que vamos a encontrar es similar al que sucedía con las claves minimales, y es que, al tratar con grandes cantidades de información, los métodos realizados para producir los generadores minimales, conllevan unas necesidades de cómputo que sobrepasan los límites de la máquina. Por tanto, una vez más, tenemos que trasladar las implementaciones de los métodos a versiones paralelas que nos permitan funcionar bajo arquitecturas de HPC. La consecución de este punto ha constituido otra fase del desarrollo de este trabajo de investigación que ha culminado con contribuciones científicas [12].

Sistemas de Recomendación Conversacionales

La última aplicación que se ha llevado a cabo en esta tesis doctoral haciendo uso de los conjuntos de implicaciones y los conjuntos cerrados se produce en el campo de los sistemas de recomendación (SR).

De forma muy básica, podríamos considerar a los SR como herramientas que agrupan un amplio abanico de técnicas y aplicaciones de los sistemas de información con la intención de potenciar y favorecer la experiencia de usuario. Estos sistemas están presentes en muchas áreas diferentes de la sociedad actual (comercio electrónico, turismo, redes sociales, películas, música, noticias, etc.) en las que es bastante frecuente contar con gran cantidad de información. Por tanto, es comprensible el tremendo crecimiento y la importancia de los SR en nuestra sociedad actual. De forma evidente, la tecnología y la ciencia han colaborado en el desarrollo y la expansión de los SR hasta el punto de convertirse en un área de investigación bien consolidada en los últimos años.

La mayoría de los SR basan sus recomendaciones en predecir cuán adecuado es un ítem para satisfacer la necesidad de un usuario y en filtrar la lista de alternativas posibles. Para lograrlo, se aplican diferentes técnicas de recomendación que dan nombre a los diferentes tipos de SR que existen, entre los que destacamos los siguientes: basados en contenido, basados en conocimiento, filtrado colaborativo, basado en el contexto, conversacionales (ver Sección 5.1). Se puede consultar una clasificación más detallada en el libro de Adomavicius y Tuzhilin [2] que, junto con la contribución de Bobadilla et al. [17], constituyen las referencias más citadas en el campo.

Sin embargo, si bien es cierto que los SR están alcanzando una enorme importancia en muchos de los aspectos relacionados con el tratamiento de la información para mejorar la experiencia de usuario, existen numerosas dificultades que han de afrontarse a la hora de diseñar e implementar un SR. En la lista de problemas relacionados con los SR [108] podemos encontrar: el arranque en frío (aparece cuando un nuevo elemento se incluye en el sistema careciendo de la información necesaria para participar en las recomendaciones), privacidad (puede surgir un problema si el sistema

necesita información sensible con respecto al usuario), oveja-negra (existen elementos que no manifiestan similitud suficiente con ningunos otros, estos elementos tan singulares pueden verse olvidados en la mayoría de las recomendaciones), escasez (pueden existir elementos del SR que sean difíciles de recomendar si la información que se tiene sobre ellos es insuficiente), ataques maliciosos (son acciones que se pueden realizar sobre un SR para romper su fiabilidad, rendimiento, confianza), postergación (elementos que no son muy populares quedan relegados por aquellos otros que gozan de mayor solicitud), etc.

Cada SR puede adolecer de uno o varios de estos problemas de forma simultánea, y normalmente cada problema suele estar más vinculado a unos tipos de SR que con otros. Sin embargo, un problema común de los SR aparece cuando es necesario trabajar sobre conjuntos de datos con un alto número de características (variables o atributos). Esta situación, que es uno de los principales problemas que se ha abordado en este trabajo, se conoce como el fenómeno de la *maldición de la dimensionalidad*. En estos casos, tratar de aplicar técnicas de minería de datos (clasificación, regresión, agrupamiento, análisis de reglas de asociación, etc.) se convierte en una tarea muy compleja porque los cálculos computacionales necesarios conllevan unos niveles de complejidad que requieren una cantidad excesiva de tiempo y recursos.

Abordar la generación de recomendaciones haciendo uso de FCA es una aproximación existente en la literatura desde hace años. En [34], los autores utilizan FCA para agrupar elementos y usuarios en conceptos para posteriormente, realizar recomendaciones colaborativas según la afinidad con los elementos vecinos. Más tarde, en [107], los autores introducen un modelo para el filtrado colaborativo basado en FCA para generar correlaciones entre datos a través de un diseño del retículo. Zhang et al. [138] propusieron un sistema basado en similitud agrupando la información contextual en grafos mediante el cual llevar a cabo recomendaciones sobre las interacciones sociales entre usuarios. En [71, 72], se utilizan relaciones difusas e implicaciones ponderadas para especificar el contexto y SL_{FD} para desarrollar un proceso lineal de filtrado que permite a los SR podar el conjunto

original de elementos y así mejorar su eficiencia. Recientemente, en [143] se propone y utiliza un novedoso SR personalizado basado en el retículo de conceptos para descubrir información valiosa de acuerdo con los requisitos e intereses de los usuarios de forma rápida y eficiente. Todos estos trabajos subrayan claramente cómo FCA puede aplicarse con éxito en el campo de los SR.

Desde el punto de vista de este trabajo, centramos nuestros esfuerzos en la aplicación de técnicas de tratamiento de implicaciones y conjuntos cerrados sobre la vertiente de SR denominada: SR conversacionales [22,136]. Pero no nos ocuparemos únicamente de la estrategia de recomendación sino también del proceso de cómo obtener una recomendación. En este sentido, se introduce el concepto de Recuperación de Información o IR por sus siglas en inglés, *Information Retrieval*, que básicamente se encarga de realizar búsquedas para determinar cuán bien responde cada objeto a una consulta. Numerosos trabajos en la literatura relacionan el uso de FCA en modelos basados en IR [24, 59].

En estos sistemas, se aplica un proceso iterativo en el cual el usuario va seleccionando características que quiere que los ítems verifiquen. A través de este intercambio y usando diferentes técnicas, el sistema progresa eligiendo (o incluso prediciendo) subconjuntos de elementos que concuerdan con las preferencias del usuario, hasta alcanzar un resultado final con un número de opciones adecuado. El problema principal en estos sistemas es que, si el conjunto de datos presenta una alta dimensionalidad, el número de pasos hasta alcanzar una recomendación razonable puede ser muy alto.

La solución que hemos realizado consiste en gestionar el problema de la alta dimensionalidad mediante un proceso de selección de características guiado por el usuario (experto humano) dentro de un sistema conversacional [136]. Para ello, una vez más haremos uso de una gestión inteligente de las implicaciones y de los conjuntos cerrados que nos va a permitir reducir sustancialmente el número de pasos necesarios en el diálogo entre el usuario y la aplicación para conseguir una recomendación adecuada en tiempo y forma. Para ello contaremos con el apoyo de la SL_{FD} introducida por os autores en [27] como lógica diseñada para desarrollar métodos de deducción.

En particular, se utilizará el algoritmo del cierre de atributos SL_{FD} [85] como núcleo de un marco de selección de atributos que será el que nos permita reducir el número de etapas del diálogo.

Además, se han realizado numerosas pruebas de aplicación sobre la propuesta que se ha desarrollado. En concreto, se han realizado pruebas utilizando información real sobre enfermedades y fenotipos como podemos apreciar en una de las contribuciones que avalan este trabajo de investigación [14]. Además, la propuesta que hemos desarrollado, al igual que la gran mayoría de los SR, permite actuar sobre diferentes *datasets* de forma que podemos utilizar el mismo procedimiento para mejorar las recomendaciones conversacionales en diversos entornos como veremos más adelante. Esta versatilidad es una característica notoria, ya que permite libertad de maniobra en el caso de que se quieran introducir ciertos cambios, o simplemente que los datos sean diferentes. En ese sentido, la propuesta de este trabajo casa con los conceptos de adaptabilidad y longevidad de los SR ya que el funcionamiento es independiente de la información de base con la que trabaje, sólo necesitamos conocer el conjunto de atributos e implicaciones subyacente a los datos.

Finalmente, antes de llegar a las últimas líneas del capítulo que dedicaremos a establecer el contenido del documento y las contribuciones producidas, cabe resaltar ya en este punto la naturaleza dual de la tesis, en el sentido de que mantendremos una línea de investigación en fundamentos teóricos complementada con la aplicación de dichos resultados en los dos campos de conocimiento mencionados anteriormente, bases de datos y sistemas de recomendación. Haremos especial hincapié en la parte aplicada del estudio con la intención de facilitar la transferencia de conocimiento a entornos diferentes del ámbito académico como el mercado empresarial. Pasamos entonces ahora a desglosar la estructura del documento y la producción conseguida.

Estructura de la Tesis

En este primer capítulo introductorio, hemos fijado los puntos fundamentales de la tesis, como son: el marco de trabajo sobre el que vamos a actuar, las técnicas que utilizaremos y los principales objetivos que se pretenden alcanzar. Concretamente, hemos estipulado la utilización de FCA y los conjuntos de implicaciones como base del estudio sobre la que aplicar técnicas basadas en la lógica para mejorar el tratamiento de la información.

A continuación, entraremos en el Capítulo 2, en el que hemos recopilado un conjunto de conceptos previos necesarios relacionados con: FCA, la lógica de simplificación, los sistemas de implicaciones y los operadores de cierre. Tras estos dos primeros capítulos, ya contaremos con el conocimiento previo necesario para abordar las dos partes principales de la tesis.

La primera parte, dedicada a las claves y los generadores minimales, consta a su vez de dos capítulos principales, uno dedicado al estudio y las contribuciones relacionadas con las claves minimales 3, y otro análogo para los generadores minimales 4. En cuanto a las claves minimales, se incluye sección con diferentes casos de aplicación donde su conocimiento es muy beneficioso. Además, se introduce el problema de la búsqueda de claves y los algoritmos que se han investigado al respecto. Cierran el capítulo dos secciones principales, en la primera se detalla la implementación realizada de los métodos de claves, tanto en su versión secuencial como paralela, y en la segunda, se detallan los experimentos y resultados alcanzados en ambos casos.

En el capítulo dedicado a los generadores minimales 4 se procede de forma análoga al de claves. Así, se presenta el problema de calcular los generadores minimales y los métodos que se han investigado y desarrollado a lo largo del trabajo realizado. A continuación, se detallan los experimentos llevados a cabo para probar el rendimiento de estos métodos. Finalmente, se presentan implementaciones paralelas de los métodos y sus respectivos resultados.

La segunda parte de la tesis estará dedicada al estudio realizado sobre el campo de los sistemas de recomendación y las contribuciones alcanzadas. En

este sentido, se incluye una primera introducción al campo de conocimiento y al estado del arte en el Capítulo 5. Seguidamente, en el Capítulo 6 se detalla el proceso llevado a cabo para desarrollar un SR conversacional, incluyendo explicaciones concretas sobre su funcionamiento y las medidas de evaluación utilizadas para calcular su rendimiento. Tras ello, llegaremos al Capítulo 7 donde se exponen los experimentos llevados a cabo haciendo uso del SR conversacional junto con los resultados obtenidos. Podemos considerar este capítulo de importancia superior pues recoge el resultado de aplicar los conceptos teóricos en aras de obtener una aplicación real de ingeniería.

Para finalizar, cerraremos la tesis con un último capítulo 8 dedicado a recopilar las principales conclusiones obtenidas y a proponer caminos por los que seguir ahondando en la investigación en esta materia. Además, se incluye una relación de las referencias consultadas y los respectivos índices de figuras, términos y tablas.

Capítulo 2

Preliminares

A lo largo de este capítulo vamos a introducir las principales ideas, definiciones y propiedades de FCA como marco de investigación principal sobre el que se sustenta el trabajo de investigación desarrollado. Procederemos de forma que consigamos que el texto sea autocontenido en la medida de lo posible. Más concretamente, se introducen los principales conceptos sobre conjuntos de implicaciones y conjuntos cerrados que serán necesarios a lo largo del documento para poder plasmar en detalle las contribuciones realizadas. La referencia principal de este campo de conocimiento viene de la mano de Wille y Ganter en [43].

2.1 Análisis de conceptos formales

De forma general, podemos considerar FCA como un método de análisis de datos que facilita la extracción de conocimiento y la posibilidad de poder razonar al respecto. Es un marco conceptual con el que: analizar, estructurar, visualizar y sobre todo, revelar el conocimiento subyacente a los datos utilizando, generalmente, técnicas de minería de datos. Existen varias de estas técnicas para actuar sobre conjuntos de objetos, conjuntos de atributos y las relaciones que se establecen entre ellos [118, 119].

La característica principal de FCA es que, basado en sólidos funda-

mentos matemáticos, cubre una amplia gama de áreas de aplicación. Su motivación original fue la representación de retículos completos de objetos y sus propiedades por medio de contextos formales, que son tablas de datos que representan relaciones binarias entre objetos y atributos. Por otro lado, FCA puede verse como una alternativa razonable a la Teoría de Retículos en términos epistemológicos [129].

El punto de partida de FCA es un **contexto formal**, lo cual nos conduce necesariamente a las dos siguientes definiciones.

Definición 2.1.1 (Atributo)**.** *Un atributo A es un identificador para un elemento en un dominio D.*

Definición 2.1.2 (Contexto formal)**.** *Un contexto formal es una tripleta $\mathbf{K} := (G, M, I)$ que consiste en dos conjuntos no vacíos, G y M, y una relación binaria I entre ellos. Los elementos de G se llaman objetos del contexto, y los elementos de M se llaman atributos del contexto. Para $g \in G$ y $m \in M$, escribimos $< g, m > \in I$ o gIm si el objeto g posee el atributo m.*

De forma más específica, un contexto formal se puede considerar como un grafo bipartito que refleje las relaciones entre los conjuntos G y M, o también, como una una tabla donde los objetos se sitúan en las filas de la tabla y los atributos en las columnas, de forma que un valor lógico en cada celda nos indica si un objeto $g \in G$ posee un atributo $m \in M$.

Ejemplo 2.1.3. *Consideremos un ejemplo de contexto formal en el cual los objetos (G) son hoteles, los atributos (M) son diferentes servicios que ofrece el establecimiento y en cada celda de la tabla encontraremos una equis siempre y cuando el correspondiente hotel ofrezca el correspondiente servicio. Por tanto, tendremos el contexto formal $\mathbf{K} := (G, M, I)$ en el cual:*

G = {Fuerte Estepona Suites, Hotel Buenavista, Hotel Paraiso, Apts Marriot Playa, Hotel Piedra Paloma, Hostal Hospederia V Cent}

M = {AC, Bar, Gym, Internet, Masajes, Parking}

La Tabla 2.1 muestra la relación binaria I del contexto formal K utilizando información real de varios hoteles de la Costa del Sol.

Cuadro 2.1: Ejemplo de contexto formal utilizando datos reales del sector hotelero

Nombre del Hotel	AC	Bar	Gym	Internet	Masajes	Parking
Fuerte Estepona Suites	✓	✓	✓	✓	✓	✓
Hotel Buenavista		✓				✓
Hotel Paraiso	✓	✓				✓
Apts Marriot Playa				✓		
Hotel Piedra Paloma		✓				✓
Hostal Hospederia V Cent	✓	✓		✓		✓
...						

Por tanto, vemos como el hotel Fuerte Estepona Suites ofrece todos los servicios disponibles en el contexto, mientras que los Apts Marriot Playa únicamente ofrecen el servicio de Internet.

A partir del contexto formal, se definen dos operadores llamados operadores de derivación.

Definición 2.1.4 (Operadores de derivación)**.** *Dado un contexto formal* $\mathbf{K} := (G, M, I)$*, a continuación se definen dos funciones, denominadas operadores de derivación:*

$$( )' \colon 2^G \to 2^M \qquad ( )' \colon 2^M \to 2^G$$

$$A' = \{m \in M \mid g\ I\ m \quad \forall g \in A\} \qquad B' = \{g \in G \mid g\ I\ m \quad \forall m \in B\}$$

Ambas funciones se denotan con el mismo símbolo porque no hay lugar a confusión. El primero asigna a cada conjunto de objetos, el conjunto de atributos comunes a todos los objetos, y el segundo asigna, a cada conjunto de atributos, el conjunto de objetos que tienen todos estos atributos.

El par de operadores de derivación constituye una conexión anti-monótona de Galois [92] con las consecuencias correspondientes, como que ambas composiciones son operadores de cierre, una de ellas en $(2^G, \subseteq)$ y la otra en $(2^M, \subseteq)$.

Definición 2.1.5 (Mapeo anti-monótono)**.** *Sean* G *y* M *dos conjuntos no vacíos. Un mapeo de la forma* $f : 2^G \to 2^M$ *se dice anti-monótono si* $A_1 \subseteq A_2$ *implica que* $f(A_2) \subseteq f(A_1) \ \ \forall A_1, A_2 \subseteq G$*.*

Definición 2.1.6 (Conexión de Galois)**.** *Sean G y M dos conjuntos no vacíos. Un par (f,g) de dos mapeos $f : 2^G \to 2^M$ y $g : 2^G \to 2^M$ es una conexión de Galois entre G y M si, para cada $A \subseteq G$ y $B \subseteq M$ se cumple que: $A \subseteq g(B)$ si y sólo si $B \subseteq f(A)$.*

El operador de cierre $''$ (i.e. aplicar el operador de derivación $'$ dos veces) verifica algunas propiedades interesantes que serán fundamentales para poder desarrollar la teoría formal.

Definición 2.1.7 (Operador de cierre)**.** *Dado un conjunto no vacío M, un operador de cierre sobre M es una función $()'' : 2^G \to 2^M$ que satisface las siguientes propiedades:*

- *Idempotente (ítems): $X'''' = X''$ $\quad \forall X \in 2^M$*
- *Monótona (atributos): $X \subseteq Y \to X'' \subseteq Y''$ $\quad \forall X, Y \in 2^M$*
- *Extensa: $X \subseteq X''$ $\quad \forall X \in 2^M$*

En general no se verifica que $X'' = X$, no obstante, cuando un conjunto de objetos, $X \subseteq G$ verifica $X'' = X$, se llama conjunto cerrado. De manera análoga a los conjuntos de objetos, podemos hablar de conjuntos de atributos cerrados. El par $(M,'')$ se denomina sistema cierre y un conjunto $A \subseteq M$ se denomina conjunto cerrado para $''$ si es un punto fijo para $''$, es decir, $''(A) = A$. La idempotencia de los operadores de cierre nos lleva al hecho de que, para todo sistema de cierre $(M,'')$ y todo $A \subseteq M$, el conjunto $''(A)$ es cerrado para $''$.

Además, los conjuntos cerrados definen lo que se denominan *conceptos formales*. De forma general, un concepto formal, que constituye un punto clave en FCA, permite describir formalmente un hecho del modelo y caracterizar un conjunto de objetos por medio de los atributos que comparten y viceversa.

De forma más intuitiva, un par (X, Y) es un concepto si se verifica que:

- Cada objeto en X tiene todos los atributos de Y.

- Para cada objeto en G que no está en X, existe un atributo en Y que el objeto no tiene.
- Para cada atributo en M que no está en Y, hay un objeto en X que no tiene ese atributo.

En otras palabras, (X, Y) es un concepto formal si X contiene todos los objetos cuyos atributos están en Y y, análogamente, Y contiene todos los atributos verificados por los objetos en X.

De esta forma, conseguimos introducir en la definición las dos partes esenciales: en primer lugar, el conjunto de objetos con propiedades comunes, y en segundo lugar, el conjunto de atributos que caracterizan a dichos objetos. Únicamente aquellos pares de conjuntos (X, Y) que tienen un cierre completo, establecen un concepto por sí mismos. En esta situación, los conjuntos X e Y son cerrados y se llaman, respectivamente, la *extensión* y la *intensión* del concepto. Para un conjunto de objetos X, el conjunto de sus atributos comunes X', describe la similitud de los objetos de X, mientras que el conjunto X'' es la agrupación de aquellos objetos que tienen como atributos comunes a X', en particular, todos los objetos de X, es decir, $X \subseteq X''$.

Más formalmente:

Definición 2.1.8 (Concepto formal)**.** *Sea* $\mathbf{K} := (G, M, I)$ *un contexto formal y* $X \subseteq G$, $Y \subseteq M$. *El par (X,Y) se denomina concepto formal si* $X' = Y$ *y* $Y' = X$. *El conjunto de objetos X se denomina extensión del concepto (X,Y) mientras que el conjunto de atributos Y será la intensión del concepto.*

El conjunto de todos los conceptos formales de un contexto formal K constituye el denominado *retículo de conceptos* con la relación de inclusión que se muestra a continuación.

Si (X_1, Y_1) y (X_2, Y_2) son conceptos, se define un orden parcial, $\leq$, de forma que $(X_1, Y_1) \leq (X_2, Y_2)$ si $X_1 \subseteq X_2$, o equivalentemente, si $Y_2 \subseteq Y_1$.

Los pares de conceptos en este orden parcial tienen una única máxima cota inferior, que es el concepto generado por $X_1 \cap X_2$. De forma análoga,

poseen una única mínima cota superior, el concepto generado por los atributos $Y_1 \cap Y_2$. Por medio de estas operaciones de cálculo del máximo y del mínimo de dos conceptos, se satisfacen los axiomas que definen un retículo. La Figura 2.1 muestra un ejemplo básico de contexto formal y retículo de conceptos asociado.

Cuadro 2.2: Ejemplo de contexto formal básico

	adulto	joven	femenino	masculino
niño		✓		✓
niña		✓	✓	
hombre	✓			✓
mujer	✓		✓	

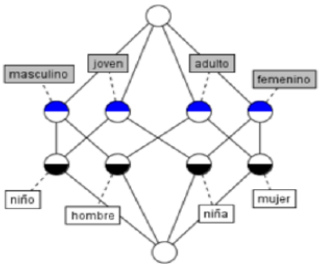

Figura 2.1: Retículo de conceptos

A partir del retículo de conceptos podemos obtener el contexto formal que representa. Los nodos del diagrama representan los objetos y atributos del contexto formal. Cada atributo del contexto corresponde a un elemento del retículo que tiene el conjunto de atributos minimal que contiene ese atributo, y un conjunto de objetos formado por todos los objetos con ese atributo. Por tanto, en la Figura 2.1 estaríamos hablando de: masculino, joven, adulto y femenino. Análogamente, cada objeto del contexto corresponde a un elemento del retículo que está formado por el conjunto de objetos minimal que contiene ese objeto, y un conjunto de atributos que consta de todos los atributos de dicho objeto. Esta vez, en la Figura 2.1 nos referimos a: niño, hombre, niña y mujer.

Una alternativa gráfica de definir los conceptos formales es la siguiente. Desde la representación del contexto formal, un concepto formal se puede reconocer por medio de una submatriz de tal manera que todas las celdas de la submatriz son verdaderas (no es necesario que la submatriz esté formada por celdas contiguas). De esta forma, en la Tabla 2.3, vemos como podemos extraer un concepto formal como el par ({*Plymouth Satellite,Chevrolet Malibu*},{*Consumo, Cilindrada, Movilidad, Aceleración*}) a partir del contexto formal.

Cuadro 2.3: Conceptos formales a partir de la representación del contexto formal.

Modelo de coche	Consumo	Cilindrada	Movilidad	Potencia	Peso	Aceleración
Chevrolet Vega				✓		✓
Toyota Corolla				✓		
Ford Pinto			✓		✓	
Volkswagen Beetle	✓		✓	✓	✓	✓
Plymouth Satellite	✓	✓	✓			✓
Chevrolet Malibu	✓	✓	✓			✓

Otra alternativa aparece en la representación como grafo bipartito, donde un concepto formal aparece como un subgrafo bipartito completo, es decir, aquel que tiene todas las aristas posibles. Las conexiones que se establecen en el retículo nos indican que el objeto X tiene un atributo Y si y sólo si existe un camino estrictamente creciente o estrictamente decreciente que permite ir de X a Y en el retículo.

Por otro lado, otro concepto equivalente a las conexiones de Galois y los operadores de cierre en el marco de FCA es el denominado sistema de implicaciones [43]. Como ya se adelantó en el Capítulo 1, las implicaciones constituyen el eje fundamental de trabajo en esta Tesis doctoral pues serán el mecanismo que nos permita abordar el problema del tratamiento eficiente de la información por medio de la lógica. Reflejan una forma relativamente sencilla de trasladar las relaciones lógicas que encontramos entre los atributos que poseen los objetos de nuestros datasets ya que combinan una forma muy simple y natural de escribir reglas del tipo *si-entonces* con una gestión muy eficiente y automatizada de la información.

Un ejemplo básico del uso de implicaciones para reflejar información contenida en los datos podemos verlo utilizando la información de la Tabla 2.1. En ese ejemplo vemos que en ese dataset hay una relación que siempre se cumple y es la siguiente. Si el hotel provee servicio de aire acondicionado (AC) entonces tiene servicio de Parking. He ahí un claro ejemplo de regla *si-entonces*, o en nuestro caso, de implicación, que se verifica en el contexto.

Este concepto de implicación es el que guía este trabajo otorgándonos un potencial para automatizar técnicas de razonamiento, con el beneficio que eso conlleva. Además, las implicaciones son elemento fundamental para

el desarrollo de formas más eficientes que el retículo de conceptos de representación del conocimiento.

Por tanto, a partir de las implicaciones y de las propiedades básicas que verifican, podemos definir un cálculo lógico que nos permita realizar sistemas de deducción completos sobre el contexto actual. En cierto sentido, podríamos decir que hemos pasado de tener un conocimiento basado en ejemplos a disponer de un conocimiento abstracto que introduce sistemas de razonamiento más elaborados, partiendo únicamente de las observaciones concretas que hemos realizado, es decir, hemos aprendido reglas generales a partir de ejemplos.

Para finalizar, podemos considerar a las implicaciones, las conexiones de Galois y los operadores de cierre, como distintas alternativas de tratar la información que se puede extraer de un contexto formal. No obstante, las conexiones de Galois y los retículos de conceptos permiten una representación gráfica, mientras que los operadores de cierre y los sistemas implicacionales nos facilitan el razonamiento por medio de la lógica.

En definitiva, todas estas nociones y propiedades son los fundamentos sobre los cuales se consolida FCA. Recordamos que la motivación original de FCA es la reestructuración de esta teoría utilizando la teoría matemática de conceptos y jerarquías con la intención de acercarse al raciocinio humano y facilitar las conexiones con las Lógicas Filosóficas del pensamiento [127,128].

2.2 Lógica de Simplificación

Introducimos la lógica SL_{FD} [27] describiendo sus cuatro pilares fundamentales: su lenguaje, su semántica, su sistema axiomático y su método de razonamiento automático.

Lenguaje

Definición 2.2.1. *Dado un conjunto M finito de símbolos (denominados atributos) no vacío, el lenguaje sobre M se define como:*

$$\mathcal{L}_M := \{A \to B \mid A, B \subseteq M\}$$

Las fórmulas $A \to B$ se denominan *implicaciones* y los conjuntos A y B reciben el nombre de *premisa* y *conclusión* de la implicación respectivamente. Los conjuntos $\sum \subseteq \mathcal{L}_M$ se denominan *sistemas de implicaciones* sobre M.

Para simplificar la notación:

- Usaremos letras minúsculas para denotar los elementos en M, mientras que las mayúsculas denotan sus subconjuntos.
- Omitimos las llaves en premisas y conclusiones, es decir, *abcde* denota el conjunto $\{a, b, c, d, e\}$.
- Escribimos sus elementos por yuxtaposición, es decir, para $X \cup Y$ escribiremos XY.
- Para la diferencia, cambiamos $X - Y$ por $X \setminus Y$.

Definición 2.2.2 (Implicación)**.** *Una implicación sobre un conjuntos de atributos M es una expresión de la forma $X \to Y$, donde X e Y son conjuntos de atributos, i.e. $X \subseteq M$ y $Y \subseteq M$.*

Definición 2.2.3 (Implicación trivial)**.** *Una implicación $X \to Y$ es trivial si $Y \subseteq X$.*

Definición 2.2.4 (Implicación unitaria)**.** *Sea X un conjunto de atributos y a un atributo del conjunto, $a \in X$, una implicación se dice unitaria si su conclusión es un conjunto unitario, i.e. $X \to a$.*

Ejemplo 2.2.5. *Sea $M = \{a, b, c, d\}$ un conjunto de atributos. Se cumple que $\{a \to b, a \to c\} \equiv \{a \to bc\}$ porque para cada contexto formal $\mathbf{K} := (G, M, I)$, se verifica que $\{b, c\}' = \{b\}' \cup \{c\}'$. Por tanto, $\{a\}' \subseteq \{b\}'$ y $\{a\}' \subseteq \{c\}'$ si y sólo si $\{a\}' \subseteq \{b, c\}'$. Este ejemplo nos permite asegurar que cualquier sistema de implicaciones es equivalente a un sistema de implicaciones unitarias.*

Proposición 2.2.6. *Sea M un conjunto de atributos y $\Sigma \subseteq \mathcal{L}_M$. Se verifica que:*

$$\Sigma \equiv \bigcup_{A \to B \in \Sigma} \{A \to b \mid b \in B\}$$

Semántica

Tras la definición del lenguaje pasamos ahora a introducir la semántica utilizada, para lo cual utilizamos el concepto de *operador de cierre* que reescribimos a continuación.

Un operador de cierre en M es una función $c\colon 2^M \to 2^M$ que es *extensivo* ($X \subseteq c(X)\ \forall X \subseteq M$), *monótono* ($X_1 \subseteq X_2$ implica que $c(X_1) \subseteq c(X_2)\ \forall X_1, X_2 \subseteq M$), e *idempotente* ($c(c(X)) = c(X) \quad \forall X \subseteq M$). Los puntos fijos para c (i.e. $X \subseteq M$ tales que $c(X) = X$) se denominan *conjuntos cerrados*.

Un operador de cierre c en M recibe el nombre de *modelo* para una implicación $A \to B \in \mathcal{L}_M$ si $B \subseteq c(A)$; se denota $c \models A \to B$. Además, dado un sistema de implicaciones $\Sigma \subseteq \mathcal{L}_M$ y un operador de cierre c en M, $c \models \Sigma$ denota que $c \models A \to B$ para todo $A \to B \in \Sigma$.

Decimos que $A \to B$ se *deriva semánticamente* de Σ, denotado $\Sigma \models A \to B$, si cualquier modelo para Σ también es un modelo para $A \to B$, i.e. $c \models \Sigma$ implica que $c \models A \to B$ para todo operador de cierre c en M.

Sistema Axiomático

Los conocidos Axiomas de Armstrong es el primer sistema descrito para tratar sistemas de implicaciones utilizando la lógica. Tiene su origen en [3] donde se utilizó para estudiar las propiedades de las dependencias funcionales en el modelo relacional de Codd [23]. Este sistema ha tenido una clara influencia en el diseño de varias lógicas sobre implicaciones, todos ellas construidas alrededor del paradigma de la transitividad. Con el paso de los años, otros trabajos han propuesto sistemas axiomáticos equivalentes [5, 58, 93].

Estos axiomas son válidos al generar sólo implicaciones dentro del cierre del conjunto de implicaciones (denotado como F^+) cuando se actúa sobre el conjunto F. También son completos ya que la aplicación repetitiva de las reglas generarán todas las implicaciones en el conjunto cerrado F^+.

Este sistema axiomático está formado por un axioma, denominado *Reflexivo*, y dos reglas de inferencia, *Aumentativa* y *Transitiva* que se definen

a continuación

Sean A, B, C subconjuntos de atributos de M. Entonces, tenemos:

$$\texttt{[Ref]}\ \frac{}{AB \to A} \qquad \texttt{[Aug]}\ \frac{A \to B \cup C}{AC \to BC} \qquad \texttt{[Tran]}\ \frac{A \to B,\ B \to C}{A \to C}$$

Si bien es cierto que el sistema de Armstrong es un trabajo que acumula innumerables citas en diferentes trabajos a lo largo de los años, su utilidad práctica se ve mermada debido a la dificultad que conlleva a la hora de plasmar las demostraciones. Es por ello que su uso está enfocado al estudio teórico de las implicaciones en vez de al desarrollo de nuevos algoritmos. No obstante, como se ha mencionado, es un buen punto de partida para el desarrollo de nuevas lógicas para el tratamiento de las implicaciones.

En este punto es donde aparece la SL_{FD} [86], que constituye un marco mucho más adecuado en términos de la automatización del razonamiento utilizando implicaciones. De hecho, SL_{FD} será el núcleo fundamental de los métodos y las aplicaciones que se han llevado a cabo a lo largo de esta tesis doctoral.

SL_{FD} constituye una lógica válida y completa [27] en la que el principal elemento del lenguaje son las implicaciones. Se define formalmente como:

Definición 2.2.7. *Sea M un conjunto finito, la formulación de SL_{FD} son expresiones, denominadas implicaciones, de la forma $X \to Y$, donde X e Y son subconjuntos de M.*

Las implicaciones se interpretan de forma conjuntiva, es decir, corresponden a las fórmulas $a_1 \wedge \ldots \wedge a_n \to b_1 \wedge \ldots \wedge b_m$ donde las proposiciones $a_1, \ldots a_n, b_1, \ldots, b_m$ son elementos del conjunto M. La interpretación es la siguiente:

Definición 2.2.8. *Sea O y M dos conjuntos finitos, que representan objetos y atributos respectivamente, y I una relación en $O \times M$. Una implicación de SL_{FD} , $X \to Y$, donde X y Y son subconjuntos de M, es válida en I si y sólo si*

$$\{o \in O \mid (o, x_i) \in I \ \ \forall x_i \in X\} \subseteq \{o \in O \mid (o, y_j) \in I \ \ \forall j_i \in Y\}$$

Además de su forma natural de expresar el conocimiento como regla, las implicaciones proporcionan un sistema de razonamiento lógico y de inferencia. Su tratamiento simbólico se propuso como hemos mencionado anteriormente en [3], sin embargo, debido al rol central que desempeña la transitividad en ese sistema axiomático, el desarrollo de métodos ejecutables para resolver problemas de implicaciones se basa en métodos indirectos. SL_{FD} evita el uso de la transitividad y se guía por la idea de simplificar el conjunto de implicaciones mediante la eliminación atributos redundantes de manera eficiente [85]. Por consiguiente, la introducción de SL_{FD} abrió la puerta al desarrollo de métodos de razonamiento automatizados directamente basados en su novedoso sistema axiomático [28,30] que resumimos a continuación.

SL_{FD} se define como el par (L_{FD}, S_{FD}) donde S_{FD} tiene el siguiente esquema axiomático:

$$\texttt{[Ref]} \quad \frac{}{A \cup B \to A}$$

junto con las siguientes reglas de inferencia, denominadas *fragmentación*, *composición* y *simplificación* respectivamente.

$$\texttt{[Frag]} \ \frac{A \to B \cup C}{A \to B} \qquad \texttt{[Comp]} \ \frac{A \to B,\ C \to D}{A \cup C \to B \cup D} \qquad \texttt{[Simp]} \ \frac{A \to B,\ C \to D}{A \cup (C \setminus B) \to D}$$

Remarcamos que el lenguaje SL_{FD} considera como fórmulas válidas aquellas en donde cualquiera de sus dos partes puede ser el conjunto vacío, denotado $A \to \varnothing$ y $\varnothing \to A$. Sus significados fueron discutidos en [27]. En ese trabajo, también presentamos el siguiente resultado donde la derivación de una implicación $A \to B$ se reduce a la derivación de la fórmula $\varnothing \to B$ teniendo $\varnothing \to A$. Este resultado se usará más adelante en el diseño de nuestro novedoso método de cierre.

Proposición 2.2.9. *Para cualquier Γ y $\forall X, Y \subseteq M$, $\Gamma \vdash X \to Y$ si y sólo si $\Gamma \cup \{\top \to X\} \vdash \top \to Y$*

Definición 2.2.10. *Se dice que una implicación $A \to B$ se deriva sintácticamente de un sistema de implicaciones Σ, y se denota por $\Sigma \vdash A \to B$, si*

existe una secuencia de implicaciones $\sigma_1, \ldots, \sigma_n \in \mathcal{L}_M$ *tal que* $\sigma_n = (A \to B)$ *y, para todo* $1 \leq i \leq n$*, la implicación* σ_i *satisface una de las siguientes condiciones:*

- σ_i *es un axioma, es decir, verifica el esquema* `[Ref]`*.*
- $\sigma_i \in \Sigma$*.*
- σ_i *se obtiene a partir de implicaciones pertenecientes a* $\{\sigma_j \mid 1 \leq j < i\}$ *aplicando las reglas de inferencia* `[Frag]`*,* `[Comp]` *o* `[Simp]`*.*

La secuencia $\sigma_1, \ldots, \sigma_n$ constituye una *demostración* para $\Sigma \vdash A \to B$.

La derivación sintáctica proporciona una gestión automatizada de las implicaciones. En particular, se puede usar para resolver el denominado problema de las implicaciones: dado un conjunto de implicaciones Γ y una implicación $A \to B$, queremos responder si $A \to B$ se deduce de Γ. Este problema se puede abordar utilizando el operador de cierre.

El sistema axiomático es válido y completo, es decir, se cumple que la derivación sintáctica y semántica coinciden. Esto significa que cada regla que se puede deducir con este sistema puede derivarse semánticamente (el sistema axiomático es válido) y viceversa (el sistema axiomático es completo).

Teorema 2.2.11. *Sea* M *un conjunto finito no vacío de atributos,* $\Sigma \subseteq \mathcal{L}_M$ *y* $A \to B \in \mathcal{L}_M$*. Entonces,* $\Sigma \models A \to B$ *si y sólo si* $\Sigma \vdash A \to B$*.*

La derivación sintáctica nos conduce a un nuevo operador de cierre denominado *cierre sintáctico.*

Definición 2.2.12. *Dado un sistema de implicaciones* $\Sigma \subseteq \mathcal{L}_M$*, un conjunto* $X \subseteq M$ *se dice cerrado respecto de* Σ *si* $A \subseteq X$ *implica* $B \subseteq X$ *para toda* $A \to B \in \Sigma$*. Debido a que* M *es cerrado con respecto a* Σ *y cualquier intersección de conjuntos cerrados es cerrada, podemos definir el siguiente operador de cierre:*

$$(\,)^+_\Sigma \colon 2^M \to 2^M \qquad X^+_\Sigma = \bigcap \{Y \subseteq M \mid Y \text{ es cerrado con respecto a } \Sigma \text{ y } X \subseteq Y\}$$

El siguiente teorema es esencial para calcular cierres y a su vez también para introducir un método de razonamiento automático.

Teorema 2.2.13 (Teorema de deducción)**.** *Sea* $A \to B \in \mathcal{L}_M$ *y* $\Sigma \subseteq \mathcal{L}_M$*. Entonces,*

$$\Sigma \vdash A \to B \quad sii \quad B \subseteq A^+_\Sigma \quad sii \quad \{\varnothing \to A\} \cup \Sigma \vdash \{\varnothing \to B\}$$

Corolario 2.2.14. *Sea* $\Sigma \subseteq \mathcal{L}_M$*.* $\forall X \subseteq M$*, se tiene que*

$$X^+_\Sigma = \text{máx}\{Y \subseteq M \mid \Sigma \vdash X \to Y\}.$$

La principal ventaja de SL_{FD} radica en que las reglas de inferencia pueden considerarse reglas de equivalencia. Como consecuencia, SL_{FD} se ha podido utilizar como núcleo principal para el desarrollo de métodos automáticos para diversas aplicaciones (e.g. obtener claves minimales, cálculos del cierre) como veremos más adelante. Un estudio más detallado al respecto, incluyendo teoremas y demostraciones, puede verse en [85].

Método de Razonamiento Automático

El problema de las implicaciones se ha abordado tradicionalmente utilizando un método básico que recibe $A \subseteq M$ como entrada y utiliza de forma exhaustiva la relación de subconjuntos recorriendo iterativamente Γ y agregando nuevos elementos al cierre. Este método, propuesto en la década de 1970 [77] y puede considerarse la base principal donde se han sustentado tantos otros. Podemos verlo detallado en el Algoritmo 2.1.

Más tarde, varios autores han desarrollado otros métodos mediante el uso de diferentes técnicas, resolviendo eficientemente este problema en tiempo lineal. En [94], los autores muestran que la complejidad del problema de cierre es $\mathtt{O}(|\mathcal{A}|\,|\Gamma|)$. En [85], presentamos un método de cierre de atributos estrechamente relacionado con el sistema axiomático SL_{FD} . También demostramos que nuestro método tiene un mejor rendimiento que aquellos basados en el cierre clásico.

Por nuestra parte, el método de razonamiento automático en SL_{FD} se basa en el Teorema de Deducción y un conjunto de equivalencias. Siguiendo

Algorithm 2.1: Cierre clásico

```
Data: Γ, A
Result: A⁺_Γ
begin
    A⁺_Γ := A
    repeat
        A' := A⁺_Γ
        foreach X → Y ∈ Γ do
            if X ⊆ A⁺_Γ and Y ⊈ A⁺_Γ then
                A⁺_Γ := A⁺_Γ ∪ {Y}
    until A⁺_Γ = A';
    return A⁺_Γ
```

la forma habitual, dos sistemas de implicaciones $\Sigma_1, \Sigma_2 \subseteq \mathcal{L}_M$ se dicen equivalentes si sus modelos coinciden, es decir, se cumple que para todo operador de cierre en M, $c \models \Sigma_1$ sii $c \models \Sigma_2$.

La siguiente proposición proporciona las tres equivalencias, denominadas también: *Fragmentación, Composición y Simplificación*, que se se aplican y justifican el nombre de la lógica SL_{FD} porque, si las leemos de izquierda a derecha, eliminan la información redundante en el sistema de implicaciones (cf. [85]).

Proposición 2.2.15. *Sean $A, B, C, D \subseteq M$. Se verifican las siguientes equivalencias:*

(i) $\{A \to B\} \equiv \{A \to B \setminus A\}$

(ii) $\{A \to B, A \to C\} \equiv \{A \to B \cup C\}$

(iii) $A \cap B = \varnothing$ *y* $A \subseteq C$ *implica* $\{A \to B, C \to D\} \equiv \{A \to B, C \setminus B \to D \setminus B\}$

Basándonos en el Teorema de Deducción y las equivalencias anteriores, en [85], los autores presentan un novedoso algoritmo para calcular cierres

utilizando SL_{FD} , denominado `Cls` que actúa según el siguiente procedimiento.

Dado un conjunto $A \subseteq M$ y un conjunto de implicaciones Σ, `Cls` calcula el par (A^+_Σ, Γ), siendo A^+_Σ el cierre sintáctico de A con respecto a Σ y Γ el conjunto residual de implicaciones respecto a $\varnothing \to A^+_\Sigma$. Esto permite determinar si una implicación $A \to B$ puede deducirse de Σ.

Los pasos del algoritmo desglosados en lenguaje natural y de forma más detallada son:

- En primer lugar, se incluye la fórmula $\varnothing \to A$ en Σ y se usa como semilla por el método de razonamiento mediante las equivalencias mencionadas en la proposición anterior.

- A continuación, el algoritmo entra en un proceso iterativo en el cual se irán aplicando las siguientes equivalencias mientras no se alcance la condición de parada.

  - **Eq. I**: Si $B \subseteq A$ entonces $\{\emptyset \to A, B \to C\} \equiv \{\emptyset \to A \cup C\}$.
  - **Eq. II**: Si $C \subseteq A$ entonces $\{\emptyset \to A, B \to C\} \equiv \{\emptyset \to A\}$.
  - **Eq. III**: En otro caso $\{\emptyset \to A, B \to C\} \equiv \{\emptyset \to A, B \setminus A \to C \setminus A\}$.

- En el momento en el que no sea posible aplicar ninguna de las equivalencias anteriores, el algoritmo termina dando como resultado el conjunto cierre A^+_Σ, es decir, el cierre sintáctico de A con respecto a Σ, y además, el conjunto residual de implicaciones Γ.

Formalmente, el algoritmo `Cls` puede verse en la Función Cls.

Aunque es cierto que existen en la literatura numerosas propuestas de algoritmos para calcular el cierre (la mayoría de ellas como modificaciones del cierre clásico de Maier [77]), la principal novedad y ventaja que aporta `Cls` es que, de manera simultánea, también reducimos el número de implicaciones, guardando el conocimiento complementario que describe la información que no pertenece al cierre. Este hecho nos coloca sin lugar a dudas en una posición privilegiada, ya que nos evita el elevado coste de un

Function Cls(A,Σ)

input : A, conjunto de atributos sobre los que se quiere calcular el cierre;
Σ, un sistema de implicaciones;
output: A^+_Σ, cierre sintáctico de A con respecto a Σ;
Γ, el sistema de implicaciones residual;

```
repeat
    Γ := Σ
    Σ := ∅ foreach B → C ∈ Γ do
        if B ⊆ A then
            A := A ∪ B
        else if C ⊈ A then
            Σ := Σ ∪ {B ∖ A → C ∖ A}
until Γ = Σ
return (A, Γ)
```

proceso de minería de datos para extraer el nuevo conjunto de implicaciones para el conjunto de datos reducido después de cada aplicación del método, algo imprescindible cuando se utilizan las implementaciones clásicas. En consecuencia, el proceso supera los costos de minería de datos preservando una complejidad lineal a lo largo del proceso.

Finalizamos este apartado mostrando un ejemplo de aplicación del algoritmo del cierre propuesto.

Ejemplo 2.2.16. *Sean* $\Sigma = \{a \to c, bc \to d, c \to ae, d \to e\}$ *y* $A = \{c, e\}$. *El algoritmo* `Cls` *devuelve* $A^+ = \{a, c, e\}$. *La siguiente tabla muestra la traza de ejecución paso a paso del algoritmo.*

Guide	Σ			
$\emptyset \to ce$	$a \to c$	$bc \to d$	$c \to ae$	$d \to e$
$\emptyset \to ce$	$a \to \not{c}$	$b\not{c} \to d$	$\not{c} \to a\not{e}$	$d \to \not{e}$
$\emptyset \to ace$		$b \to d$		

Por tanto, una vez finalizado la ejecución del algoritmo, obtenemos: $\texttt{Cls}(\{c, e\}, \{a \to c, bc \to d, c \to ae, d \to e\}) = (\{a, c, e\}, \{b \to d\})$, *donde el cierre es* $\{c, e\}^+_\Sigma = \{a, c, e\}$ *y el conjunto residual de implicaciones queda*

como: $\Gamma = \{b \to d\}$.

Parte I

Claves y Generadores Minimales

Capítulo 3

Claves Minimales

Identificar las claves de un esquema relacional es una tarea imprescindible dentro de numerosas áreas diferentes de la gestión de la información. Ejemplos de ello nos remontan incluso décadas atrás, donde podemos ver que las claves constituyen un elemento fundamental en acciones como pueden ser: la optimización de consultas [65], el indexado [65] o el modelado de datos [112], pero el concepto y sus aplicaciones son igualmente extensibles a sistemas actuales.

Generalmente, es tarea de ingeniería establecer y elegir las claves como parte de la normalización del esquema. El reto es encontrar esos atributos del esquema que nos permitan identificar de forma única cada tupla de la relación.

Técnicamente, una clave de un esquema relacional está compuesta por un subconjunto de atributos que actúan como el dominio de una función cuya imagen es el propio conjunto de atributos. Estas funciones se describen como dependencias funcionales (en adelante FD, por sus siglas en inglés, *Functional Dependencies*), las cuales especifican una restricción entre los dos conjuntos de atributos, que denotamos como $A \rightarrow B$, y que nos asegura que para cualquier tupla de la relación, si la tupla verifica el antecedente A, entonces también verifica el consecuente B. Tal y como hemos visto en los capítulos introductorios, este definición coincide exactamente con el concepto de *if-then rules* que hasta ahora hemos introducido según la

denominación de implicación por hallarnos en el marco de FCA, sin embargo, esta vez, la diferencia de nombre se debe simplemente al entorno en el que ahora vamos a trabajar, esto es, las bases de datos relacionales, donde se habla de dependencias funcionales para este tipo de reglas. Se expone formalmente en las siguientes definiciones.

Definición 3.0.17 (Atributo)**.** *Un atributo A es un identificador para un elemento en un dominio D.*

Definición 3.0.18 (Dependencia funcional)**.** *Sea Ω un conjunto de atributos. Una dependencia funcional (FD) sobre Ω es una expresión de la forma $X \to Y$, donde $X, Y \subseteq \Omega$. Se satisface en una tabla R siempre y cuando para cada dos tuplas de R, si verifican X, entonces también verifican Y.*

Definición 3.0.19 (Dependencia funcional trivial)**.** *Sea Ω un conjunto de atributos. Una dependencia funcional (FD) $X \to Y$ sobre Ω, se denomina trivial si $Y \subseteq X$.*

Definición 3.0.20 (Dependencia funcional unitaria)**.** *Sea Ω un conjunto de atributos. Una dependencia funcional (FD) $X \to Y$ sobre Ω, se denomina unitaria si Y es un conjunto unitario.*

Las claves nos permiten identificar de forma unívoca cada una de las tuplas que existan en una relación y podemos definirlas por medio de FD como sigue:

Definición 3.0.21 (Clave)**.** *Dada un tabla R sobre un conjunto de atributos Ω, decimos que K es un clave de R si se verifica la dependencia funcional $K \to \Omega$ en R.*

Una característica muy importante de la claves es su minimalidad. Una clave se considerará minimal cuando todos y cada uno de los atributos que la forman son imprescindibles para mantener su naturaleza de clave, es decir, no contiene ningún atributo superfluo. Formalmente:

Definición 3.0.22 (Clave minimal)**.** *Dada un tabla R sobre un conjunto de atributos Ω, decimos que K es una clave minimal de R si se verifica la dependencia funcional $K \to \Omega$ en R y $\nexists a \in K$ tal que $K \setminus \{a\} \to \Omega$.*

Nota 3.0.23. *Es de vital importancia mencionar que aunque para averiguar el conjunto de claves vamos a trabajar sobre FDs, no es objetivo de esta Tesis extraerlas a partir de un esquema relacional. Para ello, existen desde hace tiempo numerosas técnicas en la literatura que se ocupan de realizar esta tarea [57, 133, 134]. Por tanto, haremos uso de estas técnicas a la hora de enfrentarnos con un esquema relacional para obtener el conjunto de FD que se verifican sobre esos datos de forma que tengamos la semilla sobre la que aplicar las técnicas de búsqueda de claves que son motivo de estudio en esta tesis.*

Para ilustrar de forma básica el concepto de clave, vamos a utilizar el siguiente Ejemplo 3.0.24.

Ejemplo 3.0.24. *Supongamos que disponemos de la Tabla 3.1. Es una pequeña tabla donde se refleja información que relaciona títulos de películas, actores, países, directores, nacionalidad y años de estreno.*

De esta información, utilizando los métodos comentados anteriormente, podemos extraer el siguiente conjunto de FDs:

$\Gamma = \{Titulo, Año \rightarrow Pais;\ Titulo, Año \rightarrow Director;\ Director \rightarrow Nacionalidad\}$.

Cuadro 3.1: Tabla de películas

Título	Año	País	Director	Nacionalidad	Actor
Pulp Fiction	1994	USA	Quentin Tarantino	USA	John Travolta
Pulp Fiction	1994	USA	Quentin Tarantino	USA	Uma Thurman
Pulp Fiction	1994	USA	Quentin Tarantino	USA	Samuel Jackson
King Kong	2005	NZ	Peter Jackson	NZ	Naomi Watts
King Kong	2005	NZ	Peter Jackson	NZ	Jack Black
King Kong	1976	USA	De Laurentiis	IT	Jessica Lange
King Kong	1976	USA	De Laurentiis	IT	Jeff Bridges
Django Unchained	2012	USA	Quentin Tarantino	USA	Jamie Foxx
Django Unchained	2012	USA	Quentin Tarantino	USA	Samuel Jackson

Esta tabla tiene una sola clave minimal: $\{Titulo, Año, Actor\}$ *que corresponde con el conjunto de atributos necesario para identificar cualquier tupla de la relación.*

Aquellos lectores que no estén familiarizados con las nociones formales de FD, claves y tablas relacionales pueden consultar uno de los trabajos más citados al respecto en [37].

Es conveniente aclarar que el trabajo llevado a cabo no es una cuestión de minería de datos [40]. De forma muy general podríamos decir que la minería de datos puede considerarse un proceso computacional para descubrir patrones en grandes volúmenes de datos con el objetivo de extraer información de un conjunto de datos y transformarla en estructuras para diversos usos [130]. Sin embargo, nuestra labor consiste en desarrollar, a partir de la información ya extraída de los datos, los mecanismos y algoritmos necesarios para encontrar las claves de los conjuntos de datos, y en general, descubrir el conocimiento implícito, que nos permitan realizar un tratamiento más inteligente y eficiente de la información.

En este capítulo, analizaremos el problema de la búsqueda de claves 3.2 y el estado del arte 3.3, pero antes vamos a mostrar algunas situaciones donde la existencia de este problema es relevante 3.1. Cerrarán este capítulo dos secciones más; la primera es realmente importante ya que presenta la implementación de los métodos y la filosofía de computación paralela que se ha utilizado 3.4 y en la última se recogen en diversas tablas los resultados obtenidos por cada uno de los métodos 3.5.

3.1 Aplicaciones

Antes de seguir avanzando, es momento de apoyar el valor intrínseco de los datos mostrando una situaciones de ejemplo donde conocer las claves del sistema es fundamental.

Ejemplo 3.1.1. *Sea el caso en que tenemos almacenados los datos personales de los empleados de una determinada empresa (e.g. nombre, edad, DNI, teléfono, etc.). Se podría pensar que una forma de identificar a uno de los empleados podría ser a través de su nombre y apellidos. Sin embargo, esto no sería correcto puesto que hay personas diferentes cuyos nombres y apellidos pueden coincidir. Tal podría ser el caso de nombres más habituales*

tanto dentro del territorio nacional como Antonio Fernández o fuera, como Peter Williams. Otra alternativa podría ser elegir el número de teléfono, pero también podría ser un fallo en tanto en cuanto una misma familia o compañeros de trabajo pueden compartir el mismo número de línea fija de teléfono.

En este ejemplo, el elemento que nos permite identificar de forma única a un empleado es el DNI, y por tanto, ésa debe ser la clave de nuestro esquema. Y además, puesto que solamente contiene la información necesaria para ser clave, estamos ante una clave minimal. Sin embargo, no es la única clave que existe. Pensemos que si tomamos combinaciones de valores como *DNI* y *Apellidos*, también estaríamos obteniendo una clave válida del esquema, pero en este caso, puesto que tenemos información que no es imprescindible (e.g. *Apellidos*), no estaríamos ante una clave minimal.

Ejemplo 3.1.2 (Optimización de consultas)**.** *Dependiendo del sistema de información con el que estemos tratando, la complejidad que pueden alcanzar algunas consultas puede ser tal que su tiempo de respuesta no sea admisible. Un ejemplo de una consulta de cierta complejidad dentro de un hipotético modelo sanitario a nivel nacional, podría consistir en solicitar al sistema la lista de los varones mayores de 40, que hayan tenido al menos una intervención quirúrgica, estén casados, posean vivienda propia, tengan al menos dos hijos, ...*

Este tipo de consultas pueden necesitar acceder a tal cantidad de recursos que a la hora de obtener una respuesta del sistema, el tiempo de espera puede ser intratable. Por tanto, tener el sistema bien diseñado de manera que las consultas se puedan hacer de forma eficiente será un aspecto fundamental del sistema; una de las formas principales de conseguir esto es mediante un buen diseño que nos permita decidir a través de qué elementos dirigir la búsqueda, es decir, conocer las claves del modelo de datos [36].

Un aspecto muy importante en las tecnologías de la información es tener mecanismos que nos permitan detectar errores que pueden producirse en la gestión y almacenamiento de la información. Estos errores pueden

deberse a multitud de causas y no sólo se producen a la hora de diseñar un nuevo sistema, sino que pueden aparecer a lo largo del tiempo de vida de un sistema ya sea por el uso prolongado, cambios en las tecnologías, la intervención de diferentes personas, etc. En estos casos, es imprescindible tener mecanismos para encauzar la búsqueda del error y para ello, es fundamental conocer las claves del sistema [4]. Para mostrar una situación en la que conocer las claves nos puede ayudar en la detección de errores supongamos la siguiente situación.

Ejemplo 3.1.3 (Detección de errores)**.** *Suele ser común que se produzcan errores en bases de datos debido a las peculiaridades de algunos nombres personales. Los más comunes suelen ser la repetición de registros. Por ejemplo, pensemos una editorial que tenga una base de datos enorme de artículos científicos donde se almacena simplemente el nombre de los autores y el de las publicaciones. Ahora, pongamos por caso que en una publicación el nombre de uno de los autores es Aurora Manjón Ramos, mientras que en otra perteneciente a la misma autora, el nombre registrado es Aurora Manjón-Ramos. Si queremos obtener las aportaciones de este autor en este sistema, habrá que tener en cuenta la sutil diferencia en la forma de los apellidos pues de lo contrario la lista de artículos no será consistente.*

Esto ocurre porque el sistema está abierto al fallo humano al no tener mecanismos para bloquear la posible duplicidad de registros. Actualmente y en relación a este caso particular, el sistema de control DOI elaborado por la Corporación Nacional para Iniciativas de Investigación (CNRI) es un claro ejemplo de una forma de clave con la que identificar los trabajos. En definitiva, poseer una clave que nos determine cómo se introduce nueva información en el sistema será esencial para evitar este tipo de situaciones de error.

3.2 El problema de la búsqueda de claves

El problema de la búsqueda de claves consiste en encontrar todos los subconjuntos de atributos que componen una clave mínimal a partir de un

conjunto de FD que se verifican en un esquema de una tabla de de datos relacional. Es un campo de estudio con décadas de antigüedad en el que podemos remontarnos a un primer estudio preliminar en [39], donde las claves se estudiaron dentro del ámbito de la matriz de implicaciones u otros tantos trabajos como [44,104] que se centran en averiguar estas claves minimales.

Las claves constituyen un elemento crucial en cualquier modelo de datos, incluyendo el modelo de datos relacional de Codd [23]. Sin embargo, la dificultad al enfrentarnos con el problema de la búsqueda de claves surge debido a que, dado un conjunto de atributos A, la cardinalidad del conjunto 2^A hace que haya que abordar el problema aplicando técnicas que guíen la búsqueda de los conjuntos candidatos a ser claves minimales.

El cálculo de todas las llaves minimales representa un problema complejo. En [76, 135] se incluyen resultados interesantes acerca de la complejidad del problema; los autores demuestran que el número de claves minimales para un sistema relacional puede ser exponencial respecto al número de atributos, o factorial respecto al número de dependencias. Además, establecieron que el número de claves está limitado por el factorial del número de dependencias, por tanto, no existe un algoritmo que resuelva el problema en tiempo polinómico. Hay otros resultados que apoyan la complejidad del problema; es un problema NP-completo decidir si existe una clave de tamaño a lo sumo k dado un conjunto de FD [76].

Las principales referencias sobre este problema apuntan a los trabajos de Lucchesi y Osborn en [76] que muestran un algoritmo para calcular todas las claves candidatas. Por otro lado, Saiedian y Spencer [103] presentaron un algoritmo usando grafos con atributos para encontrar todas las claves posibles de un esquema de base de datos relacional. No obstante, demostraron que sólo podía aplicarse cuando el grafo de FDs no estuviera fuertemente conectado. Otro ejemplo lo encontramos en el trabajo de Zhang [140] en el cual se utilizan mapas de Karnaugh [63] para calcular todas las claves. También existen trabajos más recientes sobre el cálculo de las claves minimales como son [113,131] y otra contribución actual que aborda el problema en un estilo lógico [28]. Asimismo, en [73, 123, 124] los autores propusieron

el uso de FCA [43] para abordar problemas relacionados con la búsqueda y la gestión de las implicaciones, que pueden considerarse complementarios a nuestro trabajo.

Es significativo como el problema de la búsqueda de claves aparece en diversos campos de conocimiento. Por ejemplo, en [11] se hace mención a la importancia de conocer las claves en áreas emergentes como el *linked-data*. Por otro lado, en [28], mostramos cómo el problema de las claves mínimales en las bases de datos tiene su análogo en FCA, donde el papel de las FDs se manifiesta, como ya hemos comentado, como implicaciones de atributos. En ese artículo, el problema de las claves mínimales se presentó desde un punto de vista lógico y para ello se empleó un sistema axiomático para gestionar las FDs y las implicaciones; este sistema es el que en apartados anteriores hemos presentado como SL_{FD} [27].

En nuestro objetivo de esta parte de la Tesis nos vamos a concentrar en los algoritmos de búsqueda de claves basados en la lógica, y más específicamente, en aquellos que utilizan el paradigma de Tableaux [87] para determinar las claves de un esquema relacional utilizando un sistema de inferencia.

De forma muy general, podemos decir que los métodos tipo Tableaux representan el espacio de búsqueda como un árbol, donde sus hojas contienen las soluciones (claves). El proceso de construcción del árbol comienza con una raíz inicial y desde allí, las reglas de inferencia generan nuevas ramas etiquetadas con nodos que representan instancias más simples del nodo padre. La mayor ventaja de este proceso es su versatilidad, ya que el desarrollo de nuevos sistemas de inferencia nos permiten diseñar un nuevo método. Las comparaciones entre estos métodos se pueden realizar fácilmente ya que su eficiencia va de la mano con el tamaño del árbol generado.

Esto nos lleva a un punto de partida fundamental, los estudios de R. Wastl (Universidad de Wurzburg, Alemania) [125, 126] donde se introduce por primera vez un sistema de inferencia de tipo Hilbert para averiguar todas las claves de un esquema relacional. A modo de ejemplo básico, en la Figura 3.1 podemos ver un ejemplo de árbol de búsqueda según el paradigma de Tableaux desarrollado según las reglas de inferencia del sistema de

inferencia $\mathbb{K}$ de Wastl.

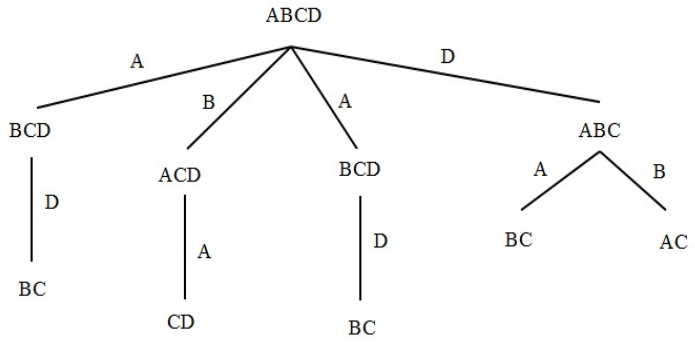

Figura 3.1: Ejemplo de Tableaux utilizando el sistema de inferencia $\mathbb{K}$ de Wastl.

Siguiendo esta línea, en [26] los autores abordan el problema de la búsqueda de claves utilizando un sistema de inferencia basado en la lógica de simplificación para dependencias funcionales [27]. En [85] los autores muestran la equivalencia entre SL_{FD} y los axiomas de Armstrong [3] junto con un algoritmo para calcular el cierre de un conjunto de atributos. Los autores también exponen una comparación con otros algoritmos del cierre que aparecen en la literatura para demostrar la orientación práctica del sistema lógico.

Más tarde, en [28], los autores introdujeron el método SST, basado en la introducción del test de minimalidad que evita la apertura de ramas adicionales del árbol, por lo que el espacio de búsqueda se vuelve más reducido, logrando un gran rendimiento en comparación con sus predecesores.

Recordemos que para abordar el problema de la búsqueda de claves en grandes sistemas, nuestro objetivo era utilizar las técnicas lógicas sobre una implementación paralela de los métodos que, mediante el uso de recursos de supercomputación, nos permitan alcanzar resultados en un tiempo razonable. En esta línea, son varios los trabajos que han utilizado la paralelización para afrontar problemas relacionados con implicaciones o FCA.

Un algoritmo paralelo para el tratamiento de implicaciones enmarcado en el campo de los hipergrafos lo podemos encontrar en [117]. A su vez, Krajca et al. [67] presentan un algoritmo paralelo para el cálculo de conceptos formales. Por nuestra parte, una primera aproximación a la paralelización del método de Wastl [125, 126] y el algoritmo de claves [26] fue presentado en [10], donde se muestra cómo el paralelismo puede integrarse de forma natural en los métodos basados en tableaux. Fue este el punto de partida para el desarrollo de los nuevos métodos más eficientes que vemos a continuación.

3.3 Algoritmos para el cálculo de claves

Nuestra contribución principal en relación a esta parte de la tesis se produce de la siguiente forma. Basándonos en el sistema axiómatico de la lógica SL_{FD} 2.2, proponemos un nuevo método llamado *Closure Keys (CK)* que incorpora un mecanismo eficiente de poda que utiliza el método de cierre basado en SL_{FD} para mejorar el método SST. El nuevo método se basa en la relación fuerte entre la noción de clave y el operador de cierre, no sólo a nivel de definición, sino también en cuanto a su construcción.

El operador de cierre definido en [85] y que hemos introducido previamente en el capítulo de Preliminares 2 como Cls, nos permite reducir el espacio de búsqueda realizando reducciones en el camino hacia las hojas, donde finalmente se obtienen las claves. Además, se ha desarrollado una implementación paralela tanto del método SST como del método CK junto con varios experimentos, los cuales necesitan llevarse a cabo en entornos de supercomputación, para confirmar las mejoras que se han alcanzado.

Método SST

En [28] se presentó un nuevo algoritmo, denominado SST, para calcular todas las claves minimales usando una estrategia de estilo tableaux, abriendo la puerta a incorporar el paralelismo en su implementación. SST se basa en la noción de cierre de conjunto, una noción básica en la teoría de base

de datos que permite caracterizar el conjunto máximo de atributos que se puede alcanzar, desde un determinado conjunto de atributos A con respecto a un conjunto de FD, utilizando el sistema axiomático. Por lo tanto, si el cierre de A se denota como A^+_Γ, el sistema de inferencia para FD nos permite inferir la FD $A \to A^+_\Gamma$. El enfoque con estilo lógico para el problema de las claves minimales consiste en la enumeración de todos los conjuntos de atributos A tales que se verifique la FD: $A \to \Omega$.

El método SST presenta dos mejoras fundamentales con respecto a sus predecesores. En primer lugar, consigue una reducción en el número de ramas y la segunda mejora consiste en el desarrollo de potentes reglas de inferencia que reducen la profundidad de las ramas, lo cual acelera significativamente la búsqueda de las claves. Estas dos nuevas reglas de inferencia, denominadas `[sSimp]` y `[lSimp]` y que se muestran a continuación son dos extensiones concretas de la regla de simplificación de la SL_{FD}.

$$[sSimp] \quad \frac{A \to B \quad C \to D}{A(C \setminus B) \to D \setminus AB} \qquad [lSimp] \quad \frac{A \to B \quad C \to D}{A(C \setminus B) \to D}$$

Estas reglas guían la construcción del espacio de búsqueda para descubrir todas las claves minimales [28]. El método avanza paso a paso construyendo un árbol desde el problema original hasta llegar a las hojas donde encontramos las soluciones, es decir, las claves. Las reglas se aplican a cada par (conjunto de atributos, conjunto de implicaciones) correspondiente a cada nodo del árbol para producir nuevos nodos en un nivel inferior, que serán el origen de nuevas ramas del árbol de búsqueda.

Al aplicar `[lSimp]` sobre el subconjunto de atributos en cada nodo y la implicación en el borde correspondiente, obtenemos la nueva raíz en la rama. Por ejemplo, en la Figura 3.2, a partir de Ω y $A_1 \to B_1$, obtenemos el nuevo subconjunto Ω_1.

Por otra parte, la aplicación de `[sSimp]` sobre la implicación establecida en cada nodo se hace tomando cada implicación como pivote y aplicando la regla al resto de las implicaciones para generar nuevas ramas. Podemos verlo en la Figura 3.2, donde la primera rama se genera tomando $A_1 \to B_1$

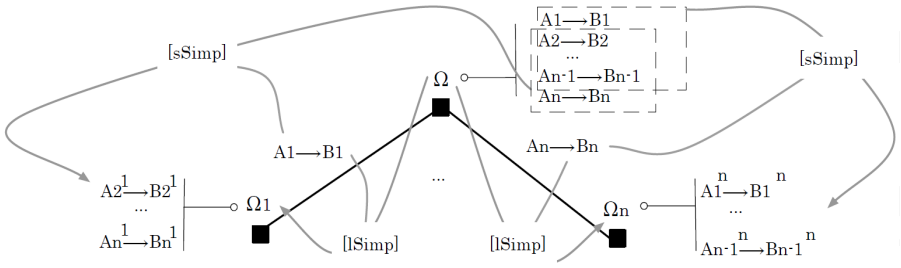

Figura 3.2: Aplicación de las reglas `[sSimp]` y `[lSimp]` en el algoritmo *Closure Keys*.

como pivote y aplicándolo al resto de implicaciones $A_2 \to B_2, \ldots, A_n \to B_n$. Cuando el conjunto de implicaciones en un nodo es el conjunto vacío, hemos llegado a una hoja del árbol y por tanto, a una clave, que agregamos a la solución global. A continuación mostramos un ejemplo completo de aplicación.

Ejemplo 3.3.1. *Sea un conjunto de atributos* $U = \{a, b, c, d, e, g\}$ *y un conjunto de implicaciones* $\Gamma = \{ab \to c, bc \to d, be \to c, cg \to b, c \to a, d \to eg, ce \to g\}$. *El conjunto* K *de todas las claves minimales que se extraen es* $K = \{cd, bc, ce, cg, bd, ab, be\}$ *y el Tableaux que se genera pondemos verlo en la Figura 3.3.*

Obsérvese que el primer nivel del árbol tiene sólo cuatro hijos, que corresponden a las cuatro fórmulas minimales en Γ. *En el caso del método* SL_{FD} *estaríamos hablando de 7 hijos en este primer nivel. La eficaz estrategia proporcionada por el método SST y sus nuevas reglas de inferencias redunda en el éxito en la construcción de todo el árbol. De esta forma, el árbol final tiene 21 nodos mientras que en el caso del método* SL_{FD} *hablamos de 244.*

SST muestra un gran rendimiento en comparación con sus predecesores como hemos visto hasta ahora y como puede comprobarse en el amplio estudio realizado sobre el método en [9]. El beneficio principal en la reducción del espacio de búsqueda de debe a la introducción del test de inclusión para evitar la apertura de ramas extra. Gracias a ello, SST no abre algunas

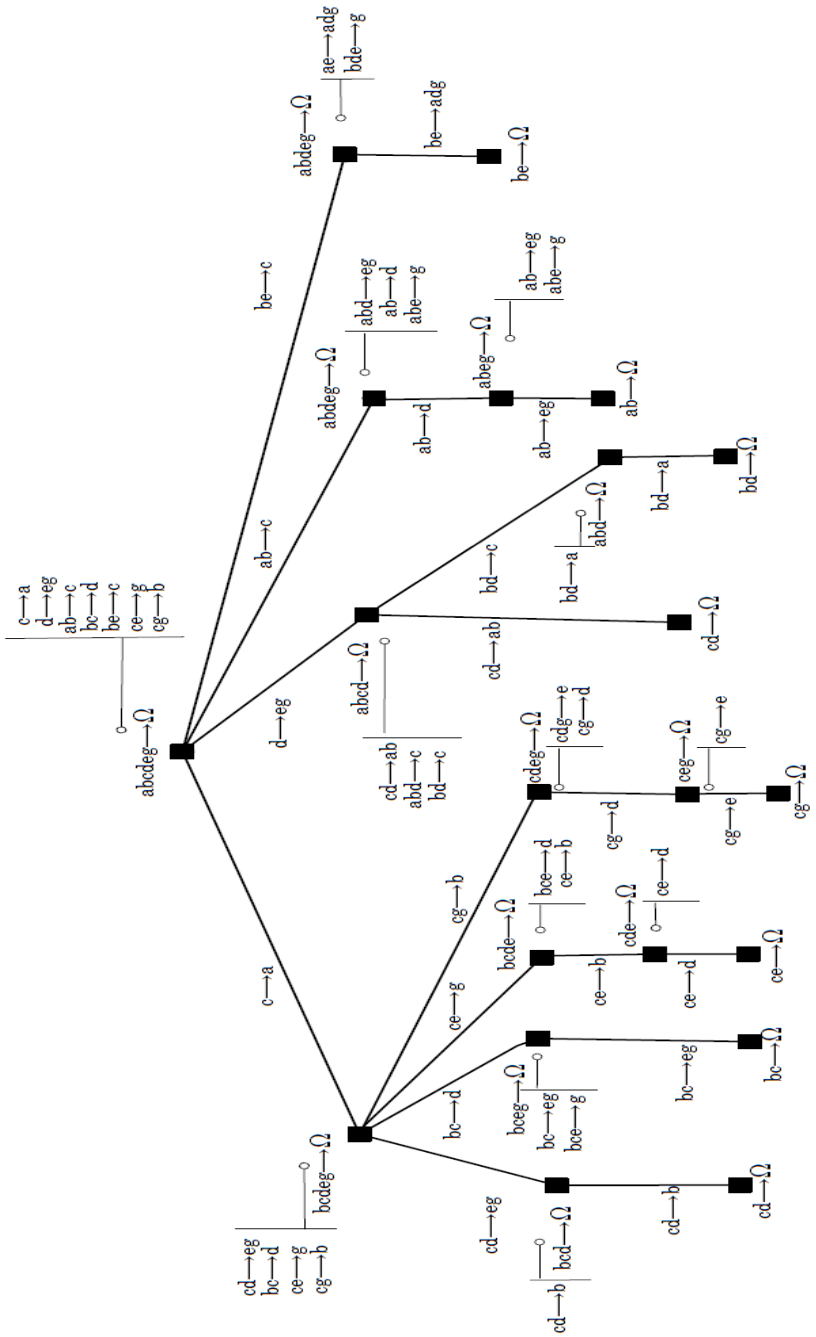

Figura 3.3: Tableaux completo para los datos del ejemplo 3.3.1.

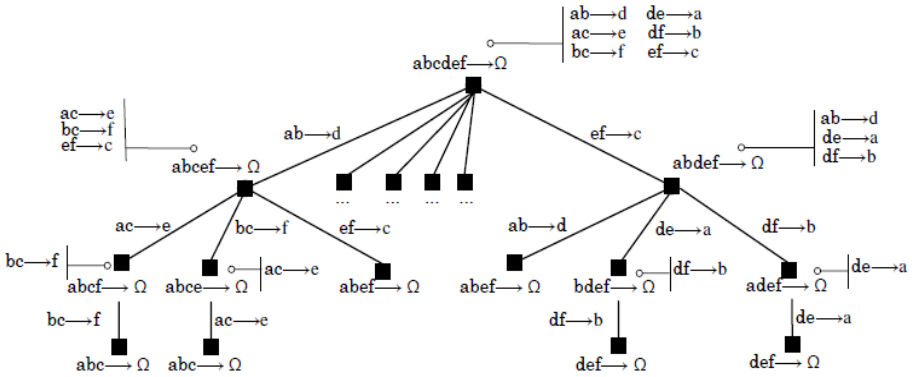

Figura 3.4: Método SL_{FD}. Tableaux de 3 niveles. 37 nodos. 18 hojas.

ramas que sabemos que van a producir las mismas claves que se calculan en otra rama. Poder identificar que tales ramas resultarán en hojas con información duplicada no es una tema trivial. Para abordarlo, hemos definido la noción de implicación minimal con respecto a un conjunto de implicaciones.

Definición 3.3.2. *Sea* Γ *un conjunto de implicaciones. La implicación* $A \to B \in \Gamma$ *es minimal si* $\forall C \to D \in \Gamma$*, se cumple que* $C \not\subset A$*.*

Podemos apreciar un ejemplo básico de cómo SST supera a su predecesor en la SL_{FD}, comparando los resultados obtenidos entre las Figuras 3.4 y 3.6, para SL_{FD} y SST respectivamente.

Método CK

El objetivo principal para seguir progresando en esta línea es la reducción del espacio de búsqueda que se puede cuantificar a través del número de nodos en el árbol. Con esta intención, hemos estudiado cómo reducir aún más el tamaño del árbol al acortar su profundidad. El resultado es este nuevo método *CK* cuyo núcleo es el algoritmo del cierre lógico publicado en [85] y que ya hemos visto como Cls puede considerarse un novedoso enfoque a los métodos clásicos de cierre, ya que proporciona una nueva característica: además del conjunto de atributos que constituye la salida del

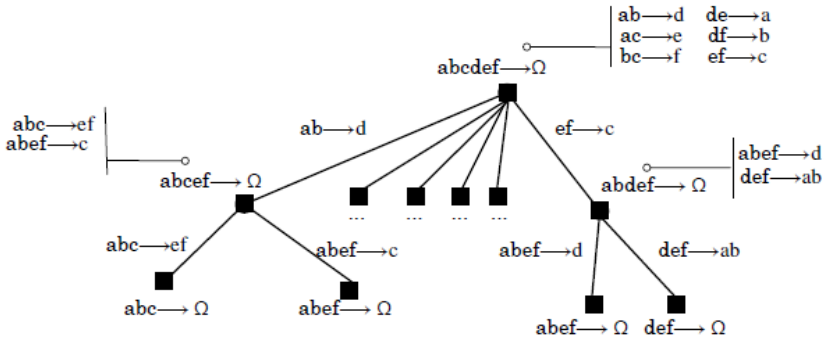

Figura 3.5: Método SST. Tableaux de 2 niveles. 19 nodos. 12 hojas.

operador de cierre, el método proporciona un subconjunto de implicaciones del conjunto Γ original. Este nuevo subconjunto de implicaciones reúne el conocimiento que puede considerarse como el complemento del cierre y se puede usar de una manera inteligente para encontrar claves minimales por medio del operador de cierre.

La ventaja principal del método es que recibe un conjunto de implicaciones Γ y un subconjunto de atributos $X \subseteq \Omega$. Calcula el conjunto cierre X^+ respecto a Γ, y además, un nuevo conjunto Γ' que contiene el conjunto de implicaciones que guarda la semántica que queda fuera del cierre X^+. Si $\Gamma' = \varnothing$, entonces $X^+ = \Omega$ (véase [85] para más detalles).

Por tanto, proponemos aplicar el Cls a cada implicación minimal del conjunto Γ en cada nodo para abrir nuevas ramas con el resultado producido por el cierre Cls. La definición de nuestro nuevo método de claves minimales se presenta en el Algoritmo 3.1.

La llamada principal del Algoritmo 3.1 debe ser Closure_Keys$(\Omega, \Gamma, \varnothing, \varnothing)$. A continuación, para obtener todas las claves mínimales, recogemos los elementos minimales del cuarto parámetro (que actúa como un acumulador en modo de entrada/salida) de esta llamada a procedimiento con respecto a $(2^{\Omega}, \subseteq)$. El siguiente teorema asegura que el Algoritmo 3.1 proporciona un método válido y completo.

Algorithm 3.1: Closure Keys (CK)

Data:
Ω un conjunto de atributos.
Γ un conjunto de implicaciones.
$\mathcal{C}$ una variable acumuladora con un subconjunto de atributos.
$\mathcal{K}$ una variable acumuladora con conjuntos de atributos.
begin
  $\mathcal{A} := \Omega$
  if $\Gamma = \varnothing$ **then**
    Add$(\mathcal{K}, \{\mathcal{A} \cup \mathcal{C}\})$;
  else
    foreach $X \to Y \in$ *[IM]*(Γ) **do**
      $< X', \Gamma' >$ **:= Cls**(X, Γ)
      $\mathcal{C} := \mathcal{C} \cup X$
      Closure_Keys($X', \Gamma', \mathcal{C}, \mathcal{K}$**)**
  /* Donde [IM](Γ) **denota las implicaciones en** Γ **que son minimales. */**

Teorema 3.3.3. *Sea Ω un conjunto de atributos y Γ un sistema de implicaciones sobre M. Entonces, el Algoritmo 3.1 devuelve todas las claves minimales.*

***D*emostración 1.** *En primer lugar, la terminación del algoritmo está garantizada porque, en cada llamada recursiva de Closure_Keys, la cardinalidad del sistema de implicaciones se reduce estrictamente. Además, el número de llamadas recursivas está limitado por la cardinalidad de Γ. El núcleo del método es el cierre Cls [85] que proporciona un resultado compuesto, es decir, $Cls(X, \Gamma) =< X^+, \Gamma' >$.*

Este método genera todas las claves minimales porque, en cada paso produce el cierre X^+ y si resulta que ese cierre no es el conjunto Ω completo, procedemos seleccionando un nuevo antecedente del conjunto de implicaciones Γ'. En virtud de esto, una vez que alcanzamos una hoja, la unión de los antecedentes en el camino conforma una clave y por lo tanto, el método es válido.

Con respecto a la completitud, *cualquier clave del sistema se va a encontrar ya que el espacio de búsqueda inducido por nuestro método realiza una búsqueda exhaustiva.*

□

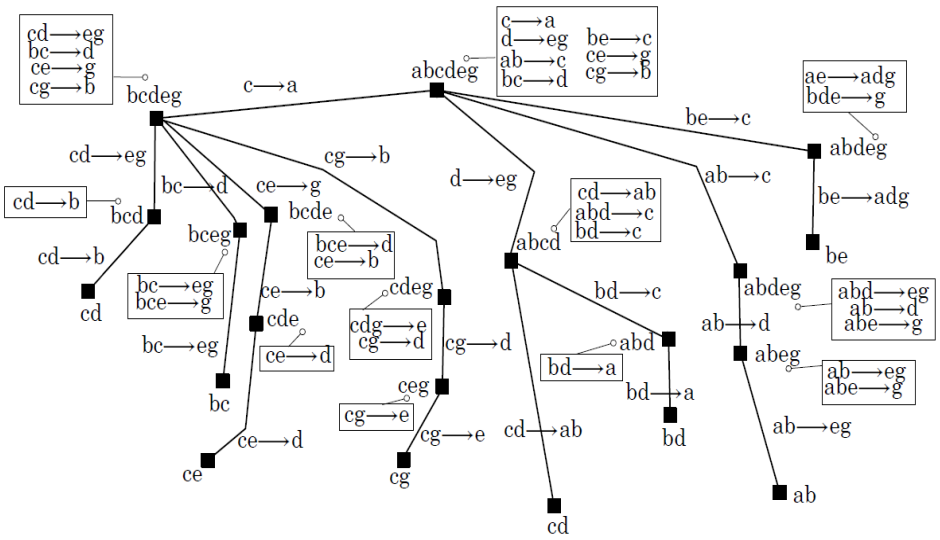

Figura 3.6: Ejemplo completo de aplicación utilizando el método SST.

Al igual que en el apartado anterior donde mostramos un ejemplo de comparación entre el árbol generado por el método SL_{FD} y el método SST (Figura 3.4 y 3.6, respectivamente), un ejemplo ilustrativo que muestra la reducción en el árbol proporcionado por CK con respecto al método SST se muestra en las figuras 3.7 y 3.6 respectivamente.

3.4 Implementación

Como conclusión de los ejemplos presentados en la sección anterior, vemos que incluso para ejemplos básicos, el árbol de búsqueda va alcanzando dimensiones considerables. De hecho, podemos consultar estudios previos donde el método SL_{FD} llega a construir árboles de millones de nodos [10]. Por tanto, para poder salvar este escollo y utilizar los métodos sobre grandes cantidades de datos, vamos a utilizar estrategias de paralelismo sobre arquitecturas hardware con altos recursos.

Nuestra intención es aprovechar el diseño del Tableaux en nuestros métodos para dividir el problema original en instancias atómicas que pueden

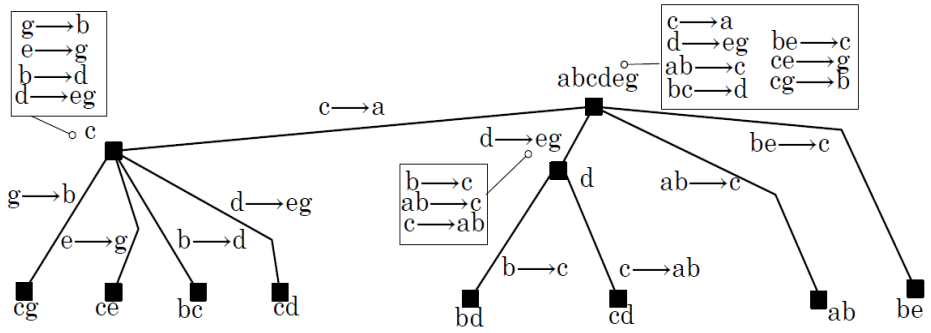

Figura 3.7: Ejemplo completo de aplicación utilizando el método CK donde se aprecia la mejora obtenida por CK sobre SST en cuando a la reducción del árbol de búsqueda, pasando de 21 nodos en SST a 11 con CK.

resolverse por separado en un tiempo razonable y con los recursos disponibles. El resultado con todas las claves minimales lo obtenemos a partir de unificar los resultados individuales obtenidos para cada una de esas instancias atómicas.

De esta forma, las implementaciones paralelas de los algoritmos funcionarán en dos etapas:

(i) **Etapa de división (secuencial).** Ejecuta el método de búsqueda de claves, en vez de construir el árbol entero, se detendrá en un cierto nivel determinado según un valor de corte (ver a continuación) generando un conjunto de sub-problemas, uno por nodo del árbol en ese nivel. El resultado de esta etapa es un conjunto de problemas de búsqueda de claves más simples.

(ii) **Etapa de resolución (paralela).** En esta segunda etapa, ejecutamos el algoritmo de búsqueda de claves sobre cada uno de los subproblemas generados en la etapa anterior de forma paralelavy, al final, combinamos todas las soluciones para obtener todas las claves minimales.

La Figura 3.8 muestra un esquema conceptual de la estrategia paralela.

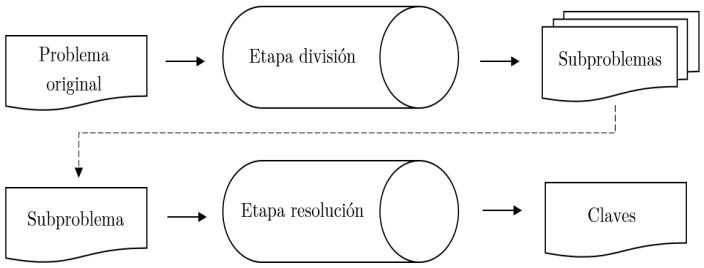

Figura 3.8: Esquema del funcionamiento en dos etapas del código paralelo.

Podemos utilizar este tipo de estrategia *MapReduce* [33] sobre la implementación paralela porque, debido a la naturaleza del árbol, cada rama será totalmente independiente de otras, por lo que cada nodo puede tratarse de manera independiente. Ese es la gran ventaja de usar el paralelismo para de estas técnicas; podemos enviar una gran cantidad de problemas a diferentes núcleos de computación de manera que pueden ser resueltos simultáneamente. En virtud de esto, el tamaño de los problemas en la entrada puede see mucho mayor de lo que la literatura nos muestra hasta ahora sin exceder las limitaciones de la máquina; hemos alejado el límite de una manera sustancial por el momento.

Como hemos mencionado, mediante la aplicación del código parcial, dividimos el problema de entrada en varios sub-problemas, sin embargo, es necesario aclarar cuál es el valor que decide ese punto de parada (BOV, en inglés, *break-off value*).

Valor de parada

Decidir el BOV es una labor crucial de la investigación actual debido a las siguientes consideraciones. Por un lado, si decidimos parar en un nivel cercano a la raíz del árbol seleccionando un BOV bajo, ciertamente estamos reduciendo el tiempo de ejecución de la etapa de división y sólo se crearán unos pocos subproblemas. Dado que el árbol no habría podido expandirse todavía, entonces no contaremos con suficiente material para ser gestionado en paralelo usando diferentes núcleos. Por otro lado, si deten-

emos la división de la etapa parcial llegados a un nivel más profundo del árbol y lejos de la raíz, la división seguramente creará una gran cantidad de subproblemas. Esta situación nos va a permitir resolver esos subproblemas usando múltiples núcleos pero, el tiempo de ejecución de la primera etapa seguramente sea mayor. Por lo tanto, hemos tenido que seleccionar BOV de forma empírica, ya que es realmente difícil averiguar el valor más adecuado simplemente analizando la información de entrada.

En nuestro caso, como valor de parada que determina el nivel en el que la rama se considera atómica para ser tratada por la etapa paralela, usamos el cardinal del conjunto de implicaciones del nodo actual, ya que hemos observado que, cuanto mayor es, más larga *suele ser* la rama. Sin embargo, en la etapa actual de la investigación, sólo podemos considerar eso como una tendencia, nunca una certeza.

Una vez que tenemos todos estos subproblemas, y debido a que todos ellos conforman un estado individual del algoritmo, pueden ser resueltos por el código paralelo. Sin embargo, si estamos tratando con problemas con una cantidad considerable de atributos e implicaciones, el número de subproblemas generados podría ser enorme (hemos realizado experimentos con más de 50,000 subproblemas [9]). Por lo tanto, su gestión es una tarea que solo está al alcance de una gran cantidad de recursos, como los ofrecidos por el Centro de Supercomputación y Bioinnovación de la Universidad de Málaga [1].

3.5 Experimentos y resultados

Llegamos a la sección donde vamos a mostrar una serie de experimentos y resultados para determinar cómo los algoritmos estudiados e implementados en la sección anterior 3.3 tratan con problemas más complejos de forma que se aprovechen los beneficios del paralelismo.

Hemos elaborado de forma aleatoria un conjunto de problemas variando la cantidad de atributos e implicaciones. En primer lugar, comenzamos con

[1] http://www.scbi.uma.es

problemas que contienen 100 implicaciones, cada una de ellas construida a partir de 100 atributos diferentes posibles. Y en segundo lugar, avanzamos un paso más, considerando problemas con 150 implicaciones y 150 atributos. Hay que tener en cuenta que estos números van más allá de las capacidades de la máquina, como se ha demostrado en estudios previos de este trabajo [26], e incluso mejoran sustancialmente los resultados dados en [10], donde ya se aplicaron técnicas paralelas.

Obviamente, ambos métodos obtienen las mismas claves de forma que se valida experimentalmente el método. Por lo tanto, hemos omitido este parámetro en tablas de resultados para mejorar la legibilidad ya que en gran parte de las ocasiones el conjunto de claves puede llegar a ser considerablemente grande. Ahora bien, para comparar los números alcanzados mediante el uso de SST y de CK, nos centramos en dos parámetros fundamentales, tiempos de ejecución y la cantidad de nodos del árbol. La elección de estos dos parámetros se debe al siguiente razonamiento.

Cuando nos planteamos la idea de la comparación de resultados, lo primero que consideramos fueron los tiempos que necesitaba cada uno de los métodos para obtener los resultados: era lo más evidente. No obstante, este parámetro está íntimamente ligado a la arquitectura que estemos utilizando para ejecutar el algoritmo, lo cual hace que el resultado dependa en gran medida de los recursos que se están utilizando y no tanto de la calidad o eficiencia del propio algoritmo. En este sentido, se oscurecía la utilidad teórica de los resultados obtenidos. Esto nos llevó a añadir como segundo parámetro de la comparación, el número de nodos del árbol como medida de la dimensión alcanzada por el problema. Además, si alguien hiciera otro método, utilizara otro código o empleara recursos hardware diferentes que desembocaran en una mejora del tiempo, siempre podríamos atenernos al tamaño del árbol pudiendo defender si realmente es una mejora en el método o en la ejecución debido a la arquitectura.

El número de nodos siempre será el mismo para un experimento individual sin importar la cantidad de veces que realicemos el test, sin embargo, los tiempos de ejecución no coincidirán exactamente con esta circunstancia; podrían ser ligeramente diferentes debido a su naturaleza intrínseca. Las

diferencias con respecto al número de nodos mostrarán cómo los nuevos métodos han mejorado el algoritmo reduciendo drásticamente la profundidad del árbol, y en consecuencia, los tiempos de ejecución. Además, hemos incluido una última columna para mostrar el ratio entre el número de nodos y el tiempo de ejecución. Finalmente, cada tiempo de ejecución que se muestra es el fruto de un estudio *a posteriori* de los resultados obtenidos de cada ejecución, de forma que conservamos los más fiables [142].

Como hemos mencionado anteriormente, la arquitectura hardware que se ha utilizado para desarrollar cada prueba que se muestra en la tesis se puede visitar en [2]. En particular, hemos desarrollado cada experimento usando 32 nodos cluster SL230, utilizando 32 núcleos y 64Gb de memoria RAM. Las comunicaciones se realizan a través de una red Infiniband FDR. Durante los experimentos, estos núcleos están reservados sólo para nuestro uso, de manera que podamos evaluar resultados fiables con respecto a los tiempos de ejecución.

Puede parecer que conseguir mejores resultados con la estrategia paralela descrita es simplemente un problema referente a la cantidad de recursos que tengamos disponibles, sin embargo, esto no es del todo cierto. Si bien es cierto que en la mayoría de los casos hemos obtenidos mejores resultados gracias a utilizar más recursos, algunos experimentos para los que se aumentó el número de núcleos disponibles no alcanzaron las expectativas esperadas en términos de tiempos de ejecución, como se muestra brevemente en la Tabla 4.4. Incrementar el número de núcleos para favorecer el paralelismo dentro de este tipo de problemas es una cuestión en la que aún queda mucho por investigar.

Entrando ya en los experimentos realizados, el primero resuelve una batería de problemas con 100 atributos y 100 implicaciones y los resultados podemos verlos en la Tabla 3.3 que está formada por dos partes. Una cabecera en la que se indica la configuración del experimento, y un cuerpo principal donde se recogen los valores alcanzados para cada problema por parte de cada método. Este cuerpo principal consta de 6 columnas que

[2] http://www.scbi.uma.es

Cuadro 3.2: Intentos de mejorar los tiempos de ejecución en base a aumentar el número de núcleos disponibles para la implementación paralela.

	Implicaciones 150	Attrib 150	Parada 140		
Problema & Método	División$_t$(s)	Total$_t$(s)	Nodos	Cores	Ratio
150150-4-SST	581	885	55.211	32	62
150150-4-CK	48	65	25.477	32	391
150150-4-SST	576	880	55.211	64	62
150150-4-CK	46	64	25.477	64	398

desglosamos a continuación:

(i) Nombre del problema y el método utilizado para resolverlo.

(ii) Número de subproblemas generados en la etapa primera del algoritmo.

(iii) Tiempo transcurrido para la etapa primera.

(iv) Tiempo total de ejecución (etapa parcial + etapa paralela).

(v) Número de nodos del árbol.

(vi) Ratio entre nodos y tiempo de ejecución.

Incluso para una cantidad tan grande de atributos y dependencias, sólo han sido necesarios alrededor de 10 minutos para que el algoritmo más lento finalice (problema 100100-3). Además, el tamaño máximo alcanzado del árbol (problema 100100-7) ronda los 300k nodos. Sería totalmente descabellado pensar en resolver estos problemas usando versiones secuenciales de los algoritmos, ya que los tiempos de ejecución se irían de las manos. No obstante, estos son resultados son altamente alentadores ya que hubiera sido impensable tratar de reproducir estos experimentos con los métodos previos [10]. Por tanto, es evidente cómo el paralelismo es una opción muy acertada para abordar este tipo de problemas.

Cuadro 3.3: Métodos paralelos aplicados a problemas grandes (I)

	Attrib 100	Implicaciones 100	Parada 90	Cores 32	
Problema & Método	Subp	División$_t$(s)	Total$_t$(s)	Nodos	Ratio
100100-1-SST	14	0	1	33	33
100100-1-CK	0	0	0	15	15
100100-2-SST	1.354	36	105	25.621	244
100100-2-CK	212	4	15	12.715	847
100100-3-SST	8.602	183	644	192.574	299
100100-3-CK	1.286	37	99	94.255	952
100100-4-SST	400	7	26	1.704	65
100100-4-CK	15	1	2	751	375
100100-5-SST	39	0	2	119	59
100100-5-CK	0	0	1	42	42
100100-6-SST	1.808	37	123	7.856	63
100100-6-CK	115	4	9	3.698	410
100100-7-SST	6.167	182	489	275.429	563
100100-7-CK	1.378	24	90	118.884	1.320
100100-8-SST	5.104	146	415	182.167	438
100100-8-CK	1.014	19	68	81.632	1.200
100100-9-SST	314	11	25	868	34
100100-9-CK	0	1	1	341	341
100100-10-SST	1.130	27	84	12.541	149
100100-10-CK	136	4	10	6.128	612

Se obtienen resultados especialmente notables como son los problemas 100100-{1,5,9}. En estos casos, hay que notar que CK no crea ningún subproblema. Por ello, la mejora que se produce es doble:

(i) En algunos casos, con el mismo valor de parada, el nuevo método ni siquiera necesita avanzar en la implementación paralela, la parcial es suficiente para resolver estos problemas.

(ii) Para aprovechar esta circunstancia, llegado el caso en el que tengamos que trabajar con problemas aún más complejos, podemos establecer un valor de parada más cercano a la raíz del árbol de forma que el tiempo parcial se reducirá significativamente.

Para el segundo experimento, seguimos la misma línea que el anterior y desarrollamos una nueva batería de 10 problemas, aumentando esta vez el número de implicaciones y atributos disponibles hasta 150. Cuanto mayor es la complejidad de estos problemas, mayor es la mejora lograda por el nuevo método CK. Los tiempos de ejecución y la cantidad de nodos han sido drásticamente reducido como se muestra en la Tabla 3.4. Esta vez, decidimos agregar un experimento adicional (problema 150150-EXTRA) debido a los notables resultados que alcanzó.

Igual que en el caso anterior, hay varios experimentos que merecen ser discutidos por separado. Por ejemplo, si fijamos nuestra atención en los problemas 150150-{3,7}, el número de subproblemas generados es mucho menor para el método CK que para SST. La diferencia, que no es ni mucho menos trivial, también propicia la gran reducción de los tiempos de ejecución de la etapa parcial. Los tiempos totales de ejecución van en la misma dirección. En cuanto al tamaño del árbol, los beneficios al introducir el cierre son bastante llamativos; en la mayoría de los casos, la cantidad de nodos se reduce en torno al 50 %. Podemos apreciar las diferencias y las mejoras obtenidas más cómodamente de forma gráfica en las Figuras 3.9 y 3.10.

Cuadro 3.4: Métodos paralelos aplicados a problemas grandes (II)

	Attrib	Implicaciones	Parada	Cores	
	150	150	140	32	
Problema & Método	Subp	División$_t$(s)	Total$_t$(s)	Nodos	Ratio
150150-1-SST	165	6	14	911	65
150150-1-CK	11	2	3	374	124
150150-2-SST	2.949	229	394	116.517	295
150150-2-CK	347	25	44	54.375	1.235
150150-3-SST	12.968	1.049	1.716	157.947	92
150150-3-CK	822	125	165	68.531	415
150150-4-SST	5.352	581	885	55.211	62
150150-4-CK	344	48	65	25.477	391
150150-5-SST	5.361	211	484	32.377	66
150150-5-CK	168	27	36	12.522	347
150150-6-SST	771	72	155	17.298	111
150150-6-CK	79	7	11	8.110	737
150150-7-SST	9.473	638	1.252	576.912	460
150150-7-CK	1.754	97	187	262.621	1.404
150150-8-SST	5.466	424	857	510.627	595
150150-8-CK	966	57	104	257.267	2.473
150150-9-SST	235	25	45	3.632	80
150150-9-CK	24	3	4	1.726	431
150150-10-SST	3.403	348	555	102.537	184
150150-10-CK	277	31	46	45.962	999
150150-EXTRA-SST	31.401	2.950	30.983	21.404.732	690
150150-EXTRA-CK	8.049	354	1.320	10.614.386	8.041

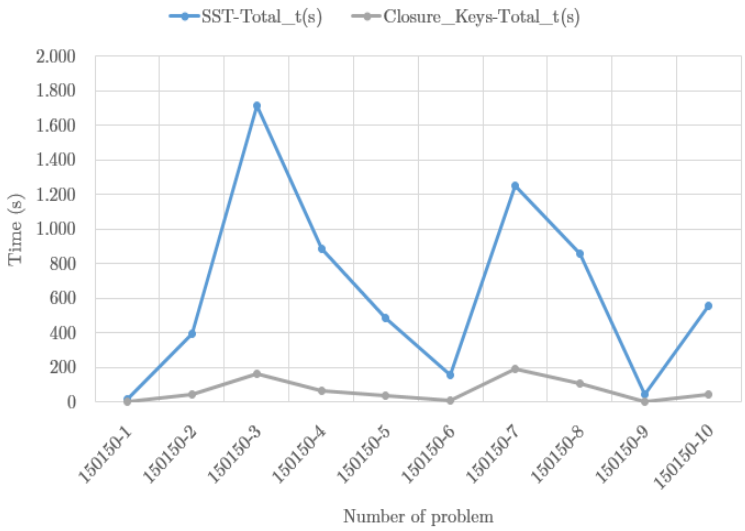

Figura 3.9: Tiempos de ejecución de los métodos paralelos aplicados a problemas grandes (II).

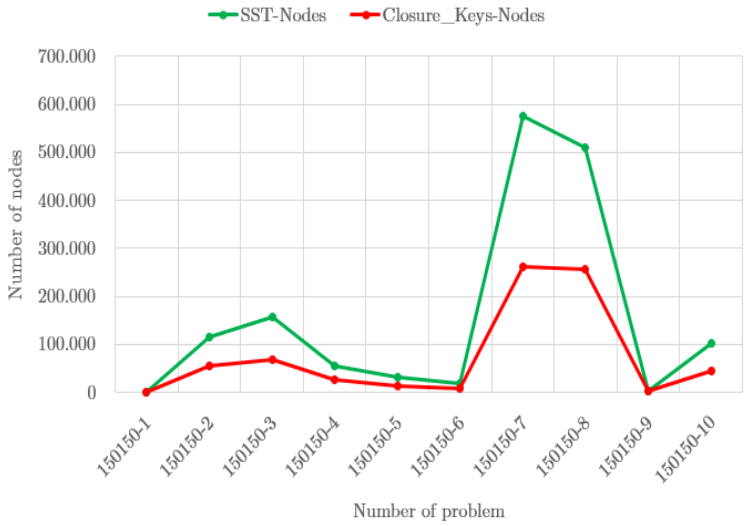

Figura 3.10: Tiempos de ejecución de los métodos paralelos aplicados a problemas grandes (II).

Capítulo 4

Generadores Minimales

El sistema de inferencia de SL_{FD} [27, 29] y la salida del algoritmo Cls se pueden usar para enumerar todos los conjuntos cerrados y todos los generadores minimales a partir de un conjunto de implicaciones y de atributos. Por tanto, el contenido de este capítulo va a estar muy ligado al anterior y muchos de los conceptos van a aparecer nuevamente.

En el capítulo de introducción 1 hablábamos de dos formas de representación del conocimiento: los sistemas de implicaciones y los retículos de conceptos. Asimismo, vimos que existen varios problemas y soluciones en cuanto a extraer ambas representaciones a partir de un *dataset* de entrada. Sin embargo, en este momento vamos a tratar con un problema complementario que permite conectar ambas representaciones del conocimiento: la enumeración de todas los conjuntos cerrados de un conjunto dado de implicaciones. Además, proponemos un método para producir no sólo todos los conjuntos cerrados, sino también, para cada uno de ellos, su representante canónico, denominados generadores minimales.

La importancia de los generadores mínimos está bien justificada por trabajos notables como el profundo estudio de Poelmans et al. [96] o el de Qu et al. [99]. Por otra parte, han sido utilizados como punto fundamental para construir bases, las cuales constituyen una representación compacta del conocimiento que permite un mejor rendimiento de los métodos de

razonamiento basados en reglas [83, 84]. Todos los autores mencionados han considerado el *dataset* como la entrada del problema, es decir, los generadores minimales y los conjuntos cerrados se infieren a partir de la información original. En [52,91] se proponen algunos métodos para resolver este problema.

En nuestro caso, centrándonos una vez más en el tratamiento inteligente de los conjuntos de implicaciones, vamos a utilizarlas como los elementos para describir la información y además, como base del diseño de un método para enumerar todos los conjuntos cerrados y sus generadores minimales a partir de esta información, y no del *dataset* original (como en los trabajos mencionados), lo cual, hasta donde sabemos, no existe trabajo previo al respecto. El nuevo método propuesto utiliza la SL_{FD} y es una evolución de [29]. Este método trabaja sobre el conjunto de implicaciones aplicando unas reglas de inferencia y construyendo un espacio de búsqueda de árbol, lo cual como ya adelantamos al principio del capítulo, va a ser muy parecido a los árboles del caso de las claves minimales.

La generación del sistema de implicaciones asociado a un operador de cierre es un problema difícil y el inverso también, teniendo una complejidad exponencial. Por lo tanto, nos vamos a enfocar en la definición de un método para resolver este problema y en el diseño de una implementación más eficiente, particularmente acercándonos al problema utilizando computación paralela.

Por tanto, en este capítulo, tras una relacionar brevemente las implicaciones con los generadores minimales 4.1, pasaremos a una sección fundamental donde se exponen cada uno de los métodos analizados e implementados 4.2. Para cada uno de ellos se presenta su definición en forma de algoritmo y se acompaña de un ejemplo ilustrativo de su funcionamiento. Una vez presentados los métodos, en la sección 4.3 se presenta de forma más detallada los pruebas realizadas y los resultados obtenidos por cada método. De forma similar a lo que se presentó para el capítulo de claves, cerrarán el capítulo los experimentos y los resultados obtenidos, esta vez, sobre la arquitectura de computación paralela 4.4.

4.1 Implicaciones y generadores minimales

Se dice que un sistema de implicaciones es completo para un operador de cierre si captura todo el conocimiento relacionado con ese operador (contexto), es decir, el sistema de implicaciones expresa en el lenguaje de la lógica todo el conocimiento relativo al cierre. Formalmente:

Definición 4.1.1. *Sea* $c\colon 2^M \to 2^M$ *un operador de cierre sobre* M *y* $\Sigma \subseteq \mathcal{L}_M$*. El sistema de implicaciones* Σ *se dice que es completo para* c *si se verifica la siguiente equivalencia:*

$$\forall A \to B \in \mathcal{L}_M, \quad c \models A \to B \quad \textit{si y sólo si} \quad \Sigma \models A \to B$$

De forma inmediata, cuando un sistema de implicaciones Σ es completo para un operador de cierre c, su operador de cierre sintáctico coincide con c, es decir, $X^+_\Sigma = c(X) \ \ \forall X \subseteq M$.

Como hemos mostrado, cualquier operador de cierre c en M puede asociarse con un sistema de implicaciones. Esta conexión establece una forma de gestionar el trabajo del operador de cierre c por medio de su derivación sintáctica y, como consecuencia, se puede elaborar un método para realizar esta gestión. Pero, ¿qué pasa con la conexión inversa? Es decir, dado un conjunto de implicaciones, ¿es posible generar el operador de cierre c asociado a él? Tal pregunta es el núcleo de esta parte de la tesis y su solución implica enumerar todos los conjuntos cerrados.

Si tenemos que $X, Y \subseteq M$ satisfacen que $X = Y^+_\Sigma$, es habitual decir que Y es un generador del conjunto cerrado X. Observe que cualquier subconjunto de X que contiene Y es también un generador de X. Dado que trabajamos con conjuntos finitos de atributos, el conjunto de los generadores de un conjunto cerrado se pueden caracterizar por sus generadores minimales.

Definición 4.1.2 (Generador minimal)**.** *Sea* $c\colon 2^M \to 2^M$ *un operador de cierre, sea* $C \subseteq M$ *un conjunto cerrado, i.e.* $c(C) = C$*, y sea* $A \subseteq M$*. El conjunto* A *se denomina generador minimal (en adelante, mingen) para* C *con respecto a* c *si* $c(A) = C$ *y,* $\forall X \subseteq A$*, si* $c(X) = C$*, entonces* $X = A$*.*

4.2 Métodos para calcular los generadores minimales

En [29], se utilizó la SL_{FD} como herramienta para encontrar todos los generadores minimales a partir de un conjunto de implicaciones. El método emplea la función Cls para guiar la búsqueda de nuevos candidatos de generador minimal. Concretamente, dado un conjunto de atributos M y un sistema de implicaciones Σ, el método realiza un mapeo $mg_\Sigma \colon 2^M \to 2^{2^M}$ que satisface la siguiente condición.

$$\forall X, Y \subseteq M$$

$X \in mg_\Sigma(C)$ si y sólo si C es cerrado para $(\ )^+_\Sigma$ y X es un generador minimal para C.

Ejemplo 4.2.1. *Sea* $\Sigma = \{a \to c, bc \to d, c \to ae, d \to e\}$*, el mapeo* mg_Σ *se describe como:*

X	$\varnothing$	b	e	be	de	ace	bde	$acde$	$abcde$
$mg_\Sigma(X)$	$\varnothing$	b	e	be	d	a	bd	ad	ab
						c		cd	bc

En otro caso, X *no es cerrado y* $mg_\Sigma(X) = \varnothing$*. Nótese que* $\varnothing$ *es cerrado y* $mg_\Sigma(\varnothing) = \{\varnothing\}$*, i.e.* $\varnothing$ *es un generador minimal del conjunto cerrado* $\varnothing$*.*

Los algoritmos que presentamos en esta sección deben usar la siguiente operación para este tipo de mapeos. Dados dos mapeos: $mg_1, mg_2 \colon 2^M \to 2^{2^M}$, el mapeo $mg_1 \sqcup mg_2 \colon 2^M \to 2^{2^M}$ se define como:

$$(mg_1 \sqcup mg_2)(X) = \texttt{Minimals}(mg_1(X) \cup mg_2(X)) \ \ \forall X \subseteq M$$

Por tanto, $Y \in (mg_1 \sqcup mg_2)(X)$ sii Y es un conjunto minimal de $mg_1(X) \cup mg_2(X)$ en $(2^M, \subseteq)$, i.e. $Y \in mg_1(X) \cup mg_2(X)$ y no existe otro conjunto de atributos $Z \in mg_1(X) \cup mg_2(X)$ tal que $Z \subsetneq Y$.

Finalmente, con lo mostrado en esta sección, ya tenemos todas las herramientas necesarias para definir el método *Minimal Generator*.

4.2.1 Método MinGen

El primer método de cálculo de los generadores minimales que hemos estudiado e implementado se introdujo originalmente en [29]. En nuestro caso, para la tesis lo describimos según la función MinGen y lo acompañamos de un ejemplo de aplicación en 4.2.2

Function MinGen(M, Label, Guide, Σ)

---

input : Conjunto de atributos M.
Un conjunto auxiliar para construir un generador minimal, Label.
Un conjunto auxiliar para construir un conjunto cerrado, Guide.
Un sistema de implicaciones Σ en M.
output: El mapeo mg_Σ.
begin
  foreach $X \subseteq M$ **do**
    $mg_\Sigma(X) := \varnothing$
  (Guide, Σ) := `Cls`(Guide, Σ)
  $M := M \setminus$Guide
  Premises := $\{A \subseteq M \mid A \to B \in \Sigma \text{ for some } B \subseteq M\}$
  ClosedSets:= $\{X \subseteq M \mid A \not\subseteq X \text{ for all } A \in \text{Premises}\}$
  foreach $X \in$ ClosedSets **do**
    $mg_\Sigma(\text{Guide} \cup X) := \{\text{Label} \cup X\}$
  foreach $A \in$ Premises **do**
    $mg_\Sigma := mg_\Sigma \sqcup \text{MinGen}(M, \text{Label} \cup A, \text{Guide} \cup A, \Sigma)$
  return mg_Σ

La entrada de MinGen es un subconjunto de atributos de M y un conjunto de implicaciones Σ. El resultado es el conjunto de conjuntos cerrados junto con todos los generadores minimlaes que los generan, es decir, $\{\langle C, mg_\Sigma(C)\rangle : C \text{ es un conjunto cerrado}\}$ donde $mg_\Sigma(C)$ es el conjunto de generadores minimales D que satisfacen $D^+_\Sigma = C$.

Ejemplo 4.2.2. *Sea el sistema de implicaciones* $\Sigma = \{a \to c, bc \to d, c \to ae, d \to e\}$. *La función MinGen*$(abcde, \varnothing, \varnothing, \Sigma)$ *devuelve el siguiente resultado:*

X	$\varnothing$	b	e	be	ace	$acde$	$abcde$	de	bde
$mg_{\Sigma}(X)$	$\varnothing$	b	e	be	a	ad	ab	d	bd
					c	cd	bc		

El árbol de búsqueda que se va generando lo podemos ver en la Figura 4.3 y la traza de ejecución la mostramos a continuación.

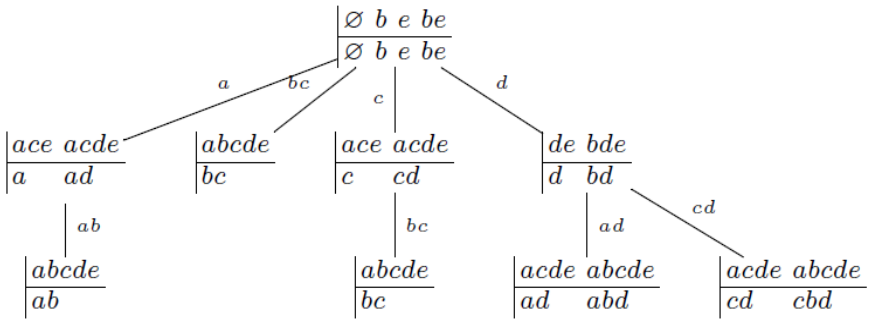

Figura 4.1: Árbol de búsqueda que genera el algoritmo MinGen para el Ejemplo 4.2.2.

$MinGen(abcde, \varnothing, \varnothing, \{a \to c, bc \to d, c \to ae, d \to e\})$:

$$Cls(\varnothing, \Sigma) = (\varnothing, \{a \to c, bc \to d, c \to ae, d \to e\}) \; y$$

X	$\varnothing$	b	e	be
$mg_{\Sigma}(X)$	$\varnothing$	b	e	be

1 $MinGen(abcde, a, a, \{a \to c, bc \to d, c \to ae, d \to e\})$:

$$Cls(a, \{a \to c, bc \to d, c \to ae, d \to e\}) = (ace, \{b \to d\}) \; y$$

X	ace	$acde$
$mg_{\Sigma_1}(X)$	a	ad

1.1 $MinGen(bd, ab, abce, \{b \to d\})$:

$$Cls(abce, \{b \to d\}) = (abcde, \varnothing) \; y$$

X	$abcde$
$mg_{\Sigma_{1,1}}(X)$	ab

Volvemos al nivel 1: $mg_{\Sigma_1} := mg_{\Sigma_1} \sqcup mg_{\Sigma_{1,1}}$ *y entonces*

X	ace	$acde$	$abcde$
$mg_{\Sigma_1}(X)$	a	ad	ab

Volvemos al nivel raiz: $mg_{\Sigma} := mg_{\Sigma} \sqcup mg_{\Sigma_1}$ *y*

X	$\varnothing$	b	e	be	ace	$acde$	$abcde$
$mg_{\Sigma}(X)$	$\varnothing$	b	e	be	a	ad	ab

2 $MinGen(abcde, bc, bc, \{a \to c, bc \to d, c \to ae, d \to e\})$*:*

$Cls(bc, \{a \to c, bc \to d, c \to ae, d \to e\}) = (abcde, \varnothing)$

X	$abcde$
$mg_{\Sigma_2}(X)$	bc

Volvemos al nivel raiz: $mg_{\Sigma} := mg_{\Sigma} \sqcup mg_{\Sigma_2}$ *y*

X	$\varnothing$	b	e	be	ace	$acde$	$abcde$
$mg_{\Sigma}(X)$	$\varnothing$	b	e	be	a	ad	ab
							bc

3 $MinGen(abcde, c, c, \{a \to c, bc \to d, c \to ae, d \to e\})$*:*

$Cls(c, \{a \to c, bc \to d, c \to ae, d \to e\}) = (ace, \{b \to d\})$ *anyd*

X	ace	$acde$
$mg_{\Sigma_3}(X)$	c	cd

3.1 $MinGen(bd, bc, abce, \{b \to d\})$*:*

$Cls(abce, \{b \to d\}) = (abcde, \varnothing)$ *y*

X	$abcde$
$mg_{\Sigma_{3,1}}(X)$	bc

Volvemos al nivel 3: $mg_{\Sigma_3} := mg_{\Sigma_3} \sqcup mg_{\Sigma_{3,1}}$ *y entonces*

X	ace	$acde$	$abcde$
$mg_{\Sigma_3}(X)$	c	cd	bc

Volvemos al nivel raiz: $mg_\Sigma := mg_\Sigma \sqcup mg_{\Sigma_3}$ *y*

X	$\varnothing$	b	e	be	ace	$acde$	$abcde$
$mg_\Sigma(X)$	$\varnothing$	b	e	be	a	ad	ab
					c	cd	bc

4 $MinGen(abcde, d, d, \{a \to c, bc \to d, c \to ae, d \to e\})$*:*

$Cls(d, \{a \to c, bc \to d, c \to ae, d \to e\}) = (de, \{a \to c, c \to a\})$ *y*

X	de	bde
$mg_{\Sigma_4}(X)$	d	bd

4.1 $MinGen(abc, ad, ade, \{a \to c, c \to a\})$*:*

$Cls(ade, \{a \to c, c \to a\}) = (acde, \varnothing)$ *y*

X	$acde$	$abcde$
$mg_{\Sigma_{4,1}}(X)$	ad	abd

Volvemos al nivel 4: $mg_{\Sigma_4} := mg_{\Sigma_4} \sqcup mg_{\Sigma_{4,1}}$ *y entonces*

X	de	bde	$acde$	$abcde$
$mg_{\Sigma_4}(X)$	d	bd	ad	abd

4.2 $MinGen(abc, cd, cde, \{a \to c, c \to a\})$*:*

$Cls(cde, \{a \to c, c \to a\}) = (acde, \varnothing)$ *y*

X	$acde$	$abcde$
$mg_{\Sigma_{4,2}}(X)$	cd	cbd

Volvemos al nivel 4: $mg_{\Sigma_4} := mg_{\Sigma_4} \sqcup mg_{\Sigma_{4,2}}$ *y entonces*

X	de	bde	$acde$	$abcde$
$mg_{\Sigma_4}(X)$	d	bd	ad	abd
			cd	cbd

Volvemos al nivel raiz: $mg_\Sigma := mg_\Sigma \sqcup mg_{\Sigma_4}$ *y efectivamente, obtenemos el resultado que habíamos adelantado*

X	$\varnothing$	b	e	be	ace	$acde$	$abcde$	de	bde
$mg_\Sigma(X)$	$\varnothing$	b	e	be	a	ad	ab	d	bd
					c	cd	bc		

Ejemplo 4.2.3. *Sea el conjunto de atributos* $M = \{1, 2, 3, 4, 5\}$ *y sea el sistema de implicaciones* $\Sigma = \{5 \to 4, 2\ 3 \to 4, 2\ 4 \to 3, 3\ 4 \to 2, 1\ 4 \to 2\ 3\ 5, 2\ 5 \to 1\ 3\ 4, 3\ 5 \to 1\ 2\ 4, 1\ 5 \to 2\ 4, 1\ 2\ 3 \to 4\ 5\}$. *La función* $MinGen(M, \varnothing, \varnothing, \Sigma)$ *genera el árbol de búsqueda que muestra la Figura 4.2 en el que los generadores minimales obtenidos están sombreados en color gris.*

El punto clave del algoritmo MinGen recae en el uso del cierre Cls ya que utiliza las ventajas de la información adicional que proporciona, es decir, Cls se usa no sólo para calcular generadores de conjuntos cerrados sino también conjuntos de implicaciones más pequeños que nos guían en la búsqueda de nuevos subconjuntos que resulten en generadores minimales.

4.2.2 Método MinGenPr

En esta sección presentamos una variante más avanzada del método MinGen anterior. Para ello, hemos incorporado un mecanismo de poda para evitar la generación de generadores minimales y cierres redundantes. El propósito de esta poda es verificar la información de cada nodo en el espacio de búsqueda, evitando la apertura de una rama completa. Esta reducción sólo puede llevarse a cabo si podemos garantizar que toda la información relativa a los generaciones minimales se genera en otras ramas. En la función MinGenPr esta estrategia de poda se implementa en la línea número #1. Por lo tanto, para garantizar que la información generada en una rama sea supérflua, diseñamos una poda basada en el test de inclusión de conjuntos que involucra a todos los nodos del mismo nivel. En la Figura 2, se ilustra cómo funciona esta estrategia de poda aplicada al siguiente ejemplo:

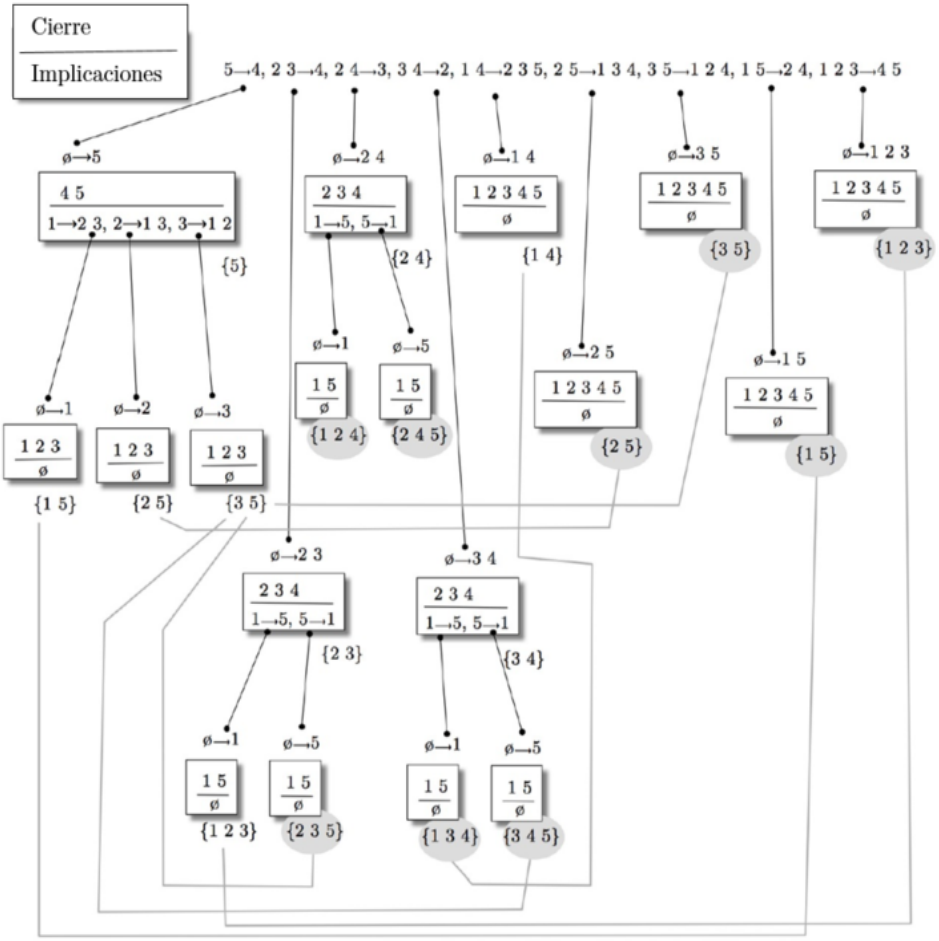

Figura 4.2: Árbol de búsqueda que genera el algoritmo MinGen para el Ejemplo 4.2.3.

Function MinGenPr(M, Label, Guide, Σ)

input : Conjunto de atributos M.
Un conjunto auxiliar para construir un generador minimal, Label.
Un conjunto auxiliar para construir un conjunto cerrado, Guide.
Un sistema de implicaciones Σ en M.
output: El mapeo mg_Σ.
begin
  foreach $X \subseteq M$ **do**
    $mg_\Sigma(X) := \varnothing$
  (Guide, Σ) := `Cls`(Guide, Σ)
  $M := M \setminus$Guide
  Premises := `Minimals`$\{A \subseteq M \mid A \to B \in \Sigma$ for some $B \subseteq M\}$
  ClosedSets:= $\{X \subseteq M \mid A \not\subseteq X$ for all $A \in$ Premises$\}$
  foreach $X \in$ ClosedSets **do**
    $mg_\Sigma($Guide $\cup X) := \{$Label $\cup X\}$
  foreach $A \in$ Premises **do**
    $mg_\Sigma := mg_\Sigma \sqcup$ MinGenPr$(M,$ Label $\cup A,$ Guide $\cup A, \Sigma)$
  return mg_Σ

Ejemplo 4.2.4. *Supongamos el mismo árbol de búsqueda que obteníamos en el Ejemplo 4.2.2. La función MinGenPr* $(abcde, \varnothing, \varnothing, \{a \to c, bc \to d, c \to ae, d \to e\})$ *aplica una estrategia de poda evitando abrir la rama cuya etiqueta es un superconjunto de otra rama en el mismo nivel. En concreto, la rama etiquetada* bc *no se abre porque no es minimal en el conjunto* $\{a, bc, c, d\}$*. La Figura 4.3 muestra el árbol de búsqueda delineando en gris la rama podada. En el Ejemplo 4.2.2 esta rama podada corresponde al ítem 2 de la traza mostrada.*

4.2.3 Método GenMinGen

Finalmente, en este caso proponemos una generalización de la estrategia de poda anterior al considerar el test de inclusión del subconjunto no sólo con la información de los nodos del mismo nivel, sino también con todos los

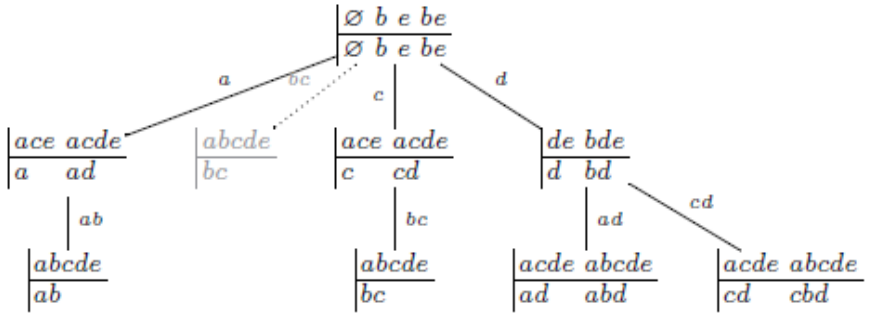

Figura 4.3: Árbol de búsqueda que genera el algoritmo MinGenPr para el Ejemplo 4.2.2.

generadores minimales calculados antes de la apertura de cada rama. Un paso más allá en esta estrategia de poda es aprovechar los generadores minimales ya calculados para aumentar el número de ramas que no es necesario abrir. Como muestra la función GenMinGen, consideramos esta poda general en la línea #1, y después, en la línea #2 se construye la lista de premisas minimales consideradas en cada etapa.

En la Figura 4.4 representamos el espacio de búsqueda correspondiente a la aplicación de la función GenMinGen sobre el Ejemplo 4.2.2. Hay que percatarse de que las ramas etiquetadas con *ad* y *cd* no se abren porque en el nivel anterior las etiquetas *a* y *c* ya se abrieron. En la salida de la función GenMinGen los generadores minimales *ad* y *cd* aparecen en ramas anteriores mientras que *abd* y *cbd* no se computan porque no son realmente generadores minimales. Para verificarlo, podemos fijarnos en los ítems 4.1 y 4.2 de la traza de ejecución del Ejemplo 4.2.2 que corresponden a ramas superfluas.

4.3 Experimentos y resultados

En las secciones anteriores hemos presentado el método original de cálculo de generadores minimales MinGen junto con las dos mejoras de poda, Min-

Function GenMinGen(M, Label, Guide, Σ)

input : Conjunto de atributos M.
Un conjunto auxiliar para construir un generador minimal, Label.
Un conjunto auxiliar para construir un conjunto cerrado, Guide.
Un sistema de implicaciones Σ en M.
output: El mapeo mg_Σ.
begin
  foreach $X \subseteq M$ **do**
    $mg_\Sigma(X) := \varnothing$
  (Guide, Σ) := `Cls`(Guide, Σ)
  $M := M \setminus$Guide
  Premises := `Minimals`$\{A \subseteq M \mid A \to B \in \Sigma$ for some $B \subseteq M\}$
  ClosedSets:= $\{X \subseteq M \mid A \not\subseteq X$ for all $A \in$ Premises$\}$
  foreach $X \in$ ClosedSets **do**
    mg_Σ(Guide $\cup X$) := {Label $\cup X$}
  foreach $A \in$ Premises **do**
[#1] **if** *there no exists* $Y \in$ MinGenList *such that* $Y \subseteq A$ **then**
        mg_Σ :=
        $mg_\Sigma \sqcup$ MinGenPr(M, Label $\cup A$, Guide $\cup A$, Σ, MinGenList)
[#2] **add** A **to** MinGenList
  return mg_Σ

GenPr y GenMinGen, y se ha mostrado un ejemplo completo de ejecución. Ahora, presentamos una comparación global del rendimiento logrado por cada uno de ellos.

Para ello, hemos desarrollado las implementaciones correspondientes para emplear los métodos sobre una batería de conjuntos de implicaciones generadas aleatoriamente. En aras de facilitar la legibilidad y el seguimiento de la comparativa realizada, acompañaremos los resultados obtenidos con varias tablas y gráficas.

Para evaluar esta comparación, vamos a utilizar dos métricas diferentes, el tiempo de ejecución del algoritmo y el número de nodos en el árbol de

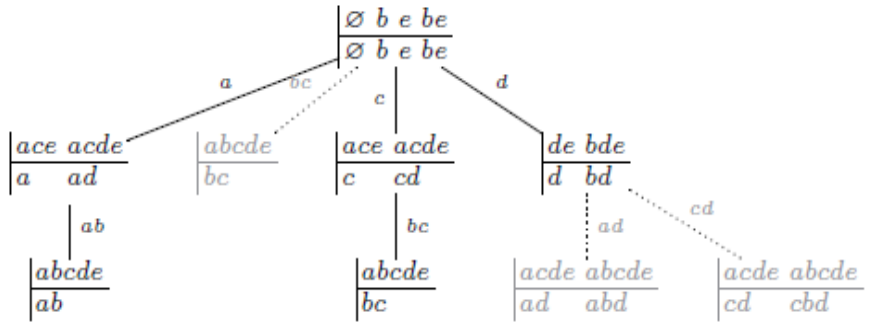

Figura 4.4: Árbol de búsqueda que genera el algoritmo GenMinGen para el Ejemplo 4.2.2.

búsqueda generado por el método. La razón de esta selección es idéntica a la expuesta para el caso de las claves minimales 3.5 y que resumimos brevemente. El tiempo de ejecución surge como la medida clásica para probar el rendimiento, pero siempre está estrechamente relacionado con los recursos con los que estamos trabajando. El número de nodos del árbol representa una medida de la magnitud del problema y es independiente de la arquitectura hardware utilizada. Asimismo, debido a la naturaleza intrínseca del tiempo de ejecución, cada experimento ha sido repetido varias veces para poder obtener valores medios fiables.

Para este experimento, en el que vamos a utilizar las implementaciones secuenciales de los algoritmos, todavía no es necesario hacer uso de recursos de supercomputación dado que los resultados se alcanzan en tiempos razonables. Entonces, la arquitectura hardware utilizada en este caso es: Intel(R) Core(TM) i7-6700HQ CPU 2.60Ghz, 8 Gb memoria RAM, ejecutándose sobre Windows 10.

Hemos generado una batería de pruebas con diferentes entradas para evaluar la implementación secuencial y poder mostrar las mejoras de la poda de los métodos desarrollados. En concreto, el experimento se va a realizar sobre los siguientes dos tipos de conjuntos de datos:

- Datos aleatorios. Hemos generado cinco ficheros de prueba, cada uno

de ellos con 50 implicaciones construidas usando 50 atributos diferentes posibles. Las implicaciones se generan aleatoriamente.

- Datos reales. Hemos generado un fichero con información real tomada a partir de los datos almacenados en *datasets* que podemos encontrar en la web como es el caso de los MovieLens *datasets*[1] que se han usado en otras ocasiones en campos como la educación, investigación o industria [53]. Entraremos en más detalles en relación a estos *datasets* cuando lleguemos a la sección 7.1, pero para completar la explicación de los ficheros de entrada de este experimento, podemos adelantar que vamos a tratar con un fichero cuyos atributos son los diferentes géneros cinematográficos almacenados en el *dataset* de MovieLens (e.g. Acción, Aventura, Thriller, Comedia, ...) con un total de 19 atributos, y cuyas implicaciones serán relaciones entre ellos que nos proporcionan un conjunto Σ final con un total de 245 implicaciones, lo cual supera sustancialmente el test aleatorio.

Dicho eso, podemos ver los resultados de los experimentos en la Figura 4.5 y en la Tabla 4.1, que está organizada en cuatro columnas cuya razón desglosamos a continuación:

(i) Identificador del archivo de entrada y el método utilizado para resolver.

(ii) Tiempo de ejecución (en segundos).

(iii) Número de nodos del árbol generado; nos da una indicación del tamaño del problema.

(iv) Número de generadores minimales obtenidos.

A la luz de los resultados obtenidos, se aprecia fácilmente como las estrategias de poda reducen significativamente tanto el número de nodos como el tiempo de ejecución. Mención especial requiere el experimento

[1] https://grouplens.org/datasets/movielens/

Cuadro 4.1: Resultados obtenidos por los métodos de generadores minimales en su implementación secuencial.

Problema y Método	Total$_t$(s)	Nodos	MinGens
sequential-1-MinGen	489	35.062	1.437
sequential-1-MinGenPr	108	7.734	1.437
sequential-1-GenMinGen	97	6.714	1.437
sequential-2-MinGen	53	32.408	271
sequential-2-MinGenPr	8	4.670	271
sequential-2-GenMinGen	7	3.922	271
sequential-3-MinGen	41	7.518	688
sequential-3-MinGenPr	9	1.642	688
sequential-3-GenMinGen	9	1.611	688
sequential-4-MinGen	693	52.067	1.444
sequential-4-MinGenPr	179	12.802	1.444
sequential-4-GenMinGen	148	11.014	1.444
sequential-5-MinGen	10.647	496.521	2.941
sequential-5-MinGenPr	1.897	72.470	2.941
sequential-5-GenMinGen	1.071	42.957	2.941
MovieLens10M-MinGen	980	254.170	2.681
MovieLens10M-MinGenPr	210	19.187	2.681
MovieLens10M-GenMinGen	198	18.926	2.681

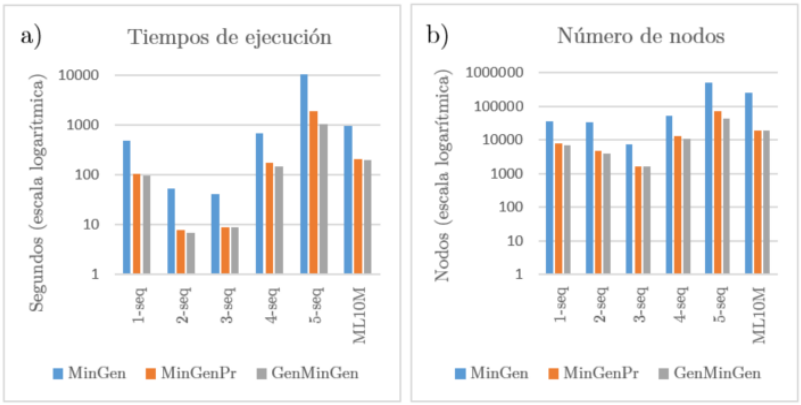

Figura 4.5: Resultados de tiempos de ejecución (a) y número de nodos (b) para el experimento con las implementaciones secuenciales de los métodos. Se ha aplicado escala logarítmica a los ejes con la intención de favorecer la visibilidad de los resultados.

'secuencial-5' ya que muestra cómo ambas métricas se han reducido drásticamente. El tiempo de ejecución se ha reducido de más de 10.000 segundos a menos de 2.000 segundos. Además, también es considerable la reducción con respecto al número de nodos que conlleva una menor necesidad de recursos en términos de memoria y almacenamiento. Finalmente, como era de prever, MinGen es superado por MinGenPr, y a su vez GenMinGen mejora este último.

Los resultados obtenidos son prometedores, pero una vez más recordemos que estamos tratando con una cantidad de información de entrada que, en otras ocasiones puede ser mucho más abundante. Por tanto, al igual que en el caso de las claves minimales, vamos a dar un paso más allá abordando el problema del cálculo de los generadores minimales con una estrategia paralela que nos permita aumentar el tamaño de los *datasets* a la entrada.

4.4 Computación paralela y generadores minimales

Hasta ahora, ya hemos mostrado las mejoras alcanzadas por las estrategias de poda dentro de los métodos de generadores minimales. Sin embargo, una vez que llega el momento de utilizar estos métodos sobre entradas de mayor tamaño, podemos deducir, a la luz de los resultados obtenidos, que los tiempos de ejecución de los métodos secuenciales serían difícilmente admisibles. No obstante, teniendo en cuenta que cada rama del árbol creado por los métodos constituye un problema en sí mismo, podemos pensar en resolverlos simultáneamente utilizando diferentes recursos. Esta visión del problema sobre una ejecución paralela, proporcionada por los métodos basados en la lógica, da lugar a desarrollar una nueva versión paralela del método de generadores minimales.

Aunque GenMinGen ha demostrado tener un mejor rendimiento que MinGenPr (y ambos a su vez un mejor rendimiento que MinGen), sólo vamos a desarrollar una versión paralela del método MinGenPr, que denominaremos *MinGenPar*. Esto se debe al hecho de que no hay necesidad de comunicación entre los subproblemas cuando se usa el método MinGenPr. Sin embargo, cuando se usa GenMinGen, recordemos que la poda aplicada

depende de los resultados obtenidos previamente en cada subproblemas, ya que en cada paso, tenemos que comparar con el conjunto actual de generadores minimales generados hasta el momento. Esto rompe la filosofía *MapReduce* de nuestra implementación, donde cada nodo del árbol (es decir, cada subproblema) está destinado a ser resuelto de forma independiente y sin existir comunicación entre los nodos del árbol.

No obstante, incluso teniendo en cuenta que los mejores resultados entre los métodos de generadores minimales son los logrados por GenMinGen, al introducir ahora la estrategia paralela, cambian las tornas. En efecto, los resultados de los métodos en su forma secuencial, incluso para GenMinGen no pueden compararse con los números que podemos alcanzar cuando usamos computación paralela con MinGenPr. La computación paralela nos brinda la posibilidad de tratar con problemas de gran tamaño, lo cual es, en la mayoría de los casos, más valioso que posibles mejoras desarrolladas en versiones secuenciales.

Antes de continuar, debemos señalar que los recursos y la arquitectura concreta que se ha utilizado para ejecutar los experimentos paralelos es, una vez más, la que nos aporta el Centro de Supercomputación y Bioinnovación de la Universidad de Málaga[2]. En particular, para este experimento, hemos utilizado 32 nodos cluster SL230, contando con 16 núcleos y 64GB de memoria RAM y 7 nodos cluster DL980, contando con 80 núcleos y 2TB de memoria RAM. Las comunicaciones se realizan a través de una red Infiniband Net FDR y QDR. En el momento de la ejecución de las pruebas, cada uno de los núcleos está reservado, de forma exclusiva, para nuestro experimento.

4.4.1 Algoritmo paralelo: *MinGenPar*

La implementación paralela del método de cálculo de generadores minimales *MinGenPar* se ha desarrollado siguiendo el marco introducido por los autores en [13] y que es exactamente el mismo que el definido para el caso del cálculo de las claves minimales 3.4. De forma breve, la imple-

[2]https://www.scbi.uma.es/

mentación paralela se lleva a cabo en dos etapas, siguiendo el paradigma *MapReduce* [33].

La primera etapa divide el problema de entrada en varios subproblemas equivalentes pero reducidos de la siguiente forma. Se utiliza un solo núcleo para construir el espacio de búsqueda hasta que se alcanza un cierto nivel de profundidad del árbol. El objetivo es dividir el problema original en varios problemas que puedan ser tratados por múltiples núcleos, desempeñando el papel de un procedimiento *Map*. El proceso continúa hasta que el tamaño del nodo actual sea inferior a un tamaño determinado β respecto al número de implicaciones.

En la segunda etapa cada uno de estos subproblemas se resuelve en paralelo usando múltiples núcleos hasta que lleguemos a las hojas del árbol de búsqueda. Como último paso, se utiliza un solo núcleo para componer el resultado global, mezclando los diferentes resultados parciales de cada subproblema en uno final (etapa *Reduce*). Este paso es imprescindible para eliminar redundancias y por tanto, obtener los generadores *minimales*. Podemos ver un esquema gráfico del proceso en la Figura 4.6 y el pseudocódigo de la implementación *MinGenPar* en MinGenPar.

Para demostrar la mejora al pasar de la implementación secuencial a la paralela, esta vez vamos a realizar una serie de experimentos donde los archivos de entrada que vamos a usar van a alcanzar valores mucho mayores que para la prueba secuencial, concretamente, contarán con hasta 150 atributos y 150 implicaciones.

No obstante, podemos comprobar que incluso con estos números, tres veces más altos que el experimento secuencial, la versión paralela nos reporta resultados en un tiempo admisible y que sobrepasan ampliamente a los obtenidos por su homóloga secuencial, como podemos observar en los valores recogidos en la Tabla 4.2.

La información mostrada en la Tabla 4.2 sigue la misma estructura que la Tabla 4.1, pero esta vez, debido a la estrategia paralela, se introducen nuevos parámetros. Por un lado, el encabezado incluye el número de atributos e implicaciones, el valor de parada (BOV) y la cantidad de núcleos utilizados. Por otro lado, el cuerpo de la tabla contiene seis columnas con

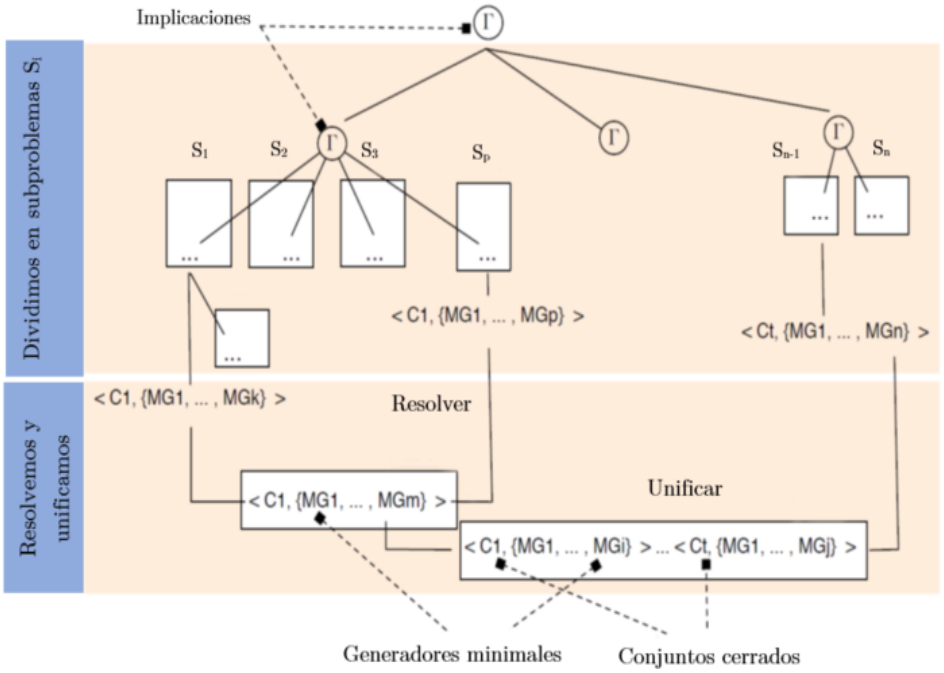

Figura 4.6: Esquema de funcionamiento de la implementación paralela de los métodos de generadores minimales utilizando el paradigma *MapReduce* [33].

la información de los valores del método que desglosamos a continuación:

(i) Nombre del problema y el método utilizado para resolverlo.

(ii) Número de subproblemas generados en la etapa primera del algoritmo.

(iii) Tiempo transcurrido para la etapa primera.

(iv) Tiempo total de ejecución (etapa parcial + etapa paralela).

(v) Número de nodos del árbol.

(vi) Número de generadores minimales obtenidos.

Optamos por mantener el número de nodos y la cantidad de generadores minimales en la tabla solamente con la intención de tener una idea del tamaño del problema, ya que es obvio que ambos implementaciones logran los mismos números.

Como conclusión, gracias a la aplicación de la implementación paralela, se han podido reducir los tiempos de ejecución de horas y días (experimentos 'MinGenPr-{2,3,7,8,10}-secuencial') a sólo unos pocos minutos. En otras palabras, la computación paralela nos permite tratar con conjuntos de datos de gran tamaño dentro de unos márgenes de tiempo admisibles. No obstante, hay dos aspectos fundamentales que merecen un estudio propio: el cálculo del valor de corte o BOV y la estimación del número idóneo de cores a utilizar en los experimentos; a ellos dedicamos los siguientes apartados.

4.4.2 Cálculo del valor de corte o BOV

Para el caso de los generadores minimales nos encontramos con el mismo problema que en el caso de las claves minimales respecto al valor de corte de la etapa de división . En la primera etapa del algoritmo paralelo hay que decidir cuándo detener la expansión de una rama del árbol de búsqueda y generar el subproblema actual correspondiente para que se resuelva en

paralelo posteriormente. Para ello, vamos a utilizar el BOV, que representa el cardinal del conjunto de implicaciones del nodo actual. El BOV se calcula como un porcentaje del tamaño de la entrada y debe ser estimado para equilibrar el trabajo realizado por cada núcleo al actuar en paralelo.

Sin embargo, se nos presenta la misma problemática que en el caso de las claves minimales, es decir, si decidimos detenernos en un nivel cercano a la raíz del árbol seleccionando un BOV bajo (es decir, un alto número de implicaciones), ciertamente estamos reduciendo el tiempo de ejecución de la etapa de división y sólo se crearían unos pocos subproblemas. En consecuencia, dado que el árbol no habría sido capaz de expandirse aún, no tendremos suficiente material para aprovechar la computación paralela usando diferentes núcleos. Por otro lado, si detenemos la división en un nivel lejos de la raíz, el proceso de división seguramente creará una gran cantidad de subproblemas que pueden sacar partido del paralelismo, pero el tiempo de ejecución de esta primera etapa seguramente crecerá. Esta tesitura es la que nos lleva a seleccionar un BOV empíricamente, ya que es realmente difícil obtener el valor más óptimo con sólo analizar la entrada teóricamente. Sin embargo, después de hacer muchos experimentos, hay varios aspectos que vale la pena mencionar y esta sección se presenta con esa intención.

Para explorar las situaciones que se pueden dar cuando elegimos diferentes valores de BOV, hemos repetido nuestros experimentos paralelos usando diversos BOV. Tomamos BOV tales que la etapa de división alcance una mayor profundidad en el árbol, en concreto, utilizaremos 130 y 100 como valores de BOV, es decir, el $\approx 86{,}67\,\%$ y $\approx 66{,}67\,\%$ del conjunto original de implicaciones, respectivamente. Esta selección de valores no se hace por casualidad sino con el objetivo de provocar ciertas situaciones que hemos encontrado después de llevar a cabo una gran cantidad de experimentos que analizaremos a continuación. Los resultados relativos a los tiempos de ejecución son muestran en la Tabla 4.3.

A la luz de estos resultados, cabe destacar en primer lugar que, excepto para el problema MinGenPar-1 donde existe una diferencia sutil, todos los problemas se comportan peor con respecto al tiempo de ejecución cuando

demoramos el punto de división. Para encontrar una explicación, ponemos nuestra atención en el número de subproblemas, donde van a surgir cuatro comportamientos diferentes cuando variamos el BOV. Procedemos a enumerar cada uno de ellos con el apoyo de los resultados obtenidos en la Tabla 4.3.

Sea BOV_1, BOV_2 dos BOV diferentes para el mismo experimento tal que $BOV_1 > BOV_2$, y sea Sp_1, Sp_2 el número de subproblemas generados en cada ejecución, respectivamente. Entonces, pueden acontecer diferentes escenarios donde:

(i) $Sp_1 < Sp_2$. Esta puede ser la situación más natural ya que con $BOV_2 < BOV_1$ el algoritmo tiene más margen para expandir el árbol y generar más subproblemas. De esta forma, el paralelismo tomaría ventaja, pero al aumentar el tiempo necesario para dividir, el tiempo de ejecución global empeoraría. Los experimentos Problemas MinGenPar-{2,6,7,8,10} reflejan esta situación.

(ii) $Sp_1 > Sp_2$. Puede suceder que incluso profundizando en el árbol obtengamos menos nodos, es decir, menos subproblemas. Esto se debe a que hay ramas que terminan antes de alcanzar los BOV y se resuelven dentro de la etapa de división, por lo que el tiempo de ejecución aumenta. Podemos ver este caso en el problema MinGenPar-4.

(iii) $Sp_1 \neq 0 \wedge Sp_2 = 0$. Siguiendo con el punto anterior, podemos caer en una situación extrema donde no se generan subproblemas dentro de la etapa de división porque cada rama del árbol termina antes del punto de división BOV, por lo tanto el problema original se resuelve por completo dentro de la etapa de división sin alcanzar el escenario paralelo. Esta situación se puede considerar como el peor de los casos, ya que la implementación paralela funciona igual que la secuencial. Esto sucede para cada problema cuando se usa un $BOV \approx 66{,}67\,\%$ y también en problemas como MinGenRd-{1,3,5} incluso con un $BOV \approx 86{,}67\,\%$.

(iv) $Sp_1 = Sp_2$. Finalmente, aunque utilicemos diferentes BOV, podemos obtener el mismo número de subproblemas. Esto refleja una intrincada situación en la que los niveles inferiores del árbol pueden contar con el mismo número de nodos que los superiores. La cuestión clave es que cuando profundicemos en el árbol, podemos generar más nodos a medida que el árbol se expande y así crecerá el número de subproblemas, pero también pueden ser varias las ramas que terminan antes de seguir profundizando, y por lo tanto reduce el número de nodos. Por lo tanto, con estas adiciones y sustracciones, el cálculo global de subproblemas puede terminar en un empate aunque se utilicen diferentes BOV como se muestra en el problema MinGenRd-9. Pero eso no es todo. El Problema MinGenRd-9 muestra, efectivamente, el mismo número de subproblemas para $BOV \approx 93{,}33\,\%$ y $BOV \approx 86{,}67\,\%$, sin embargo, el tiempo de ejecución no es el mismo, de hecho, es peor para un $BOV \approx 86{,}67\,\%$ ya que hemos prolongado la etapa de división.

4.4.3 Estimación del número óptimo de cores

Hasta ahora, podemos pensar que es sólo una cuestión de recursos el poder tratar con problemas de mayor tamaño y aún así obtener resultados dentro de un tiempo razonable. Sin embargo, no es tan obvio y este apartado está dedicado a analizar este hecho. Con este propósito, los problemas de gran tamaño usados anteriormente en la sección 4.4.1 se analizarán nuevamente usando: 16, 32, 48, 64 y 80 núcleos cada vez. Los resultados se muestran en la Tabla 4.4.

Aunque los recursos son mejores ahora, los resultados no acompañan. Se puede ver que mejoramos el rendimiento mediante el uso de más núcleos (por ejemplo, de 16 a 64 núcleos), sin embargo, llega un momento donde no se obtiene mejora; los resultados son bastante similares para 48, 64 y 80 núcleos. Esto se debe al hecho de que hay varias ramas en el árbol (tal vez sólo una) que tarda mucho en terminar y por tanto, no importa cuánto mejoremos los recursos, el tiempo global sigue siendo casi el mismo.

El problema es que por el momento, no podemos predecir si una rama resultará en una larga o corta. Esta situación detiene nuestras primeras intenciones de aumentar ciegamente la cantidad de recursos y muestra que, el número de núcleos para este problema se puede establecer en 64 como un valor óptimo, logrando un equilibrio entre los recursos necesarios y los beneficios obtenidos.

4.4.4 Generadores minimales para un *dataset* real

Para nalizar adecuadamente esta sección, traemos ahora los resultados obtenidos al aplicar el algoritmo de cálculo de generadores minimales a un conjunto de datos del mundo real. En particular, hemos elegido *Mushroom Data Set*[3] accesible desde el sitio web de la Universidad de California, Irvine (UCI)[4]. Básicamente, este conjunto de datos incluye descripciones de hipotéticas muestras correspondientes a 8.124 especies de setas utilizando 22 atributos diferentes.

En este conjunto de datos, se realiza una adaptación para convertir la información de entrada cuyo formato contempla múltiples valores, en información binaria de manera que pueda ser tratada por el algoritmo MinGenPar. Como resultado, obtenemos un *dataset* con 126 atributos y un conjunto de implicaciones para este *dataset* de 1.587. Queremos declarar que con tal número de implicaciones, la versión secuencial del algoritmo no termina. Para llevar a cabo este experimento usamos las conclusiones alcanzadas anteriormente, es decir, utilizaremos 64 núcleos diferentes y un $BOV \approx 93{,}33\,\%$, es decir $BOV = 1{,}481$, ya que son ser los mejores valores para proceder.

Los resultados de este experimento se muestran en la tabla 4.5, donde se aprecia claramente la forma en que nuestro algoritmo cumple con las expectativas cuando se trata de un conjunto de datos de gran tamaño y del mundo real.

[3] https://archive.ics.uci.edu/ml/datasets/mushroom
[4] http://archive.ics.uci.edu/ml/

Function MinGenPar(M,Label,Guide,Σ,BOV)

input : Conjunto de atributos M.
Un conjunto auxiliar para construir un generador minimal, Label.
Un conjunto auxiliar para construir un conjunto cerrado, Guide.
Un sistema de implicaciones Σ en M.
BOV: Break of Value for the splitting stage

output: El mapeo mg_Σ

begin

- **foreach** $X \subseteq M$ **do** $mg_\Sigma(X) := \varnothing$
- (Guide,Σ):=`Cls`(Guide,Σ)
- $M := M \setminus$Guide
- Premises := `Minimals`$\{A \subseteq M \mid A \to B \in \Sigma$ for some $B \subseteq M\}$
- ClosedSets:= $\{X \subseteq M \mid A \not\subseteq X$ for all $A \in$ Premises$\}$
- **foreach** $X \in$ ClosedSets **do** $mg_\Sigma(\text{Guide} \cup X) := \{\text{Label} \cup X\}$
- **If** $\mid \Sigma \mid \leq BOV$ **then do in parallel**
  - `// [MAP]`
  - **foreach** $A \in$ Premises **do**
    - `// [REDUCE]`
    - $mg_\Sigma := mg_\Sigma \sqcup \text{MinGenRd}(M, \text{Label} \cup A, \text{Guide} \cup A, \Sigma)$
  - **return** mg_Σ
- **else**
  - `// [Splitting stage]`
  - **foreach** $A_k \in$ Premises **do**
    - $mg^k_\Sigma = \text{MinGenRdPar}(M, \text{Label} \cup A_k, \text{Guide} \cup A_k, \Sigma, BOV)$
  - **forall the** mg^k_Σ **do**
    - `// [REDUCE]`
    - $mg_\Sigma := mg_\Sigma \sqcup mg^k_\Sigma$
  - **return** mg_Σ

Cuadro 4.2: Comparación entre las versiones secuencial y paralela del algoritmo MinGen-Par aplicado a problemas de gran tamaño.

Problema & Método	Subp	División$_t$(s)	Paralelo$_t$(s)	Total$_t$(s)	Nodos	MinGens
MinGenPar-1-sequential	-	-	-	43	374	216
MinGenPar-1-parallel	11	3	1	4	374	216
MinGenPar-2-sequential	-	-	-	17.352	54.375	6.273
MinGenPar-2-parallel	347	220	55	275	54.375	6.273
MinGenPar-3-sequential	-	-	-	33.338	68.531	6.529
MinGenPar-3-parallel	822	2.338	251	2.589	68.531	6.529
MinGenPar-4-sequential	-	-	-	4.612	25.477	2.478
MinGenPar-4-parallel	344	350	97	447	25.477	2.478
MinGenPar-5-sequential	-	-	-	1.585	12.522	1.159
MinGenPar-5-parallel	168	432	30	462	12.522	1.159
MinGenPar-6-sequential	-	-	-	1.653	8.110	1.436
MinGenPar-6-parallel	79	35	7	42	8.110	1.436
MinGenPar-7-sequential	-	-	-	107.238	262.621	9.113
MinGenPar-7-parallel	1.754	958	242	1.200	262.621	9.113
MinGenPar-8-sequential	-	-	-	61.381	257.267	5.538
MinGenPar-8-parallel	966	253	188	441	257.267	5.538
MinGenPar-9-sequential	-	-	-	372	1.726	683
MinGenPar-9-parallel	24	7	2	9	1.726	683
MinGenPar-10-sequential	-	-	-	7.484	45.962	2.969
MinGenPar-10-parallel	277	186	65	251	45.962	2.969

Cuadro 4.3: Experimentos utilizando diferentes valores de BOV con la implementación paralela *MinGenPar* sobre problemas de gran tamaño.

Problema	Subp	División$_t$(s)	Paralelo$_t$(s)	Total$_t$(s)	BOV
MinGenPar-1	0	44	-	44	66.67 %
	0	41	-	41	86.67 %
	11	3	1	4	93.33 %
MinGenPar-2	0	18.533	-	18.533	66.67 %
	885	8.532	58	8.590	86.67 %
	347	220	55	275	93.33 %
MinGenPar-3	0	33.377	-	33.377	66.67 %
	0	33.285	-	33.285	86.67 %
	822	2.338	251	2.589	93.33 %
MinGenPar-4	0	4.858	-	4.858	66.67 %
	308	3.703	22	3.725	86.67 %
	344	350	97	447	93.33 %
MinGenPar-5	0	1.601	-	1.601	66.67 %
	0	1.547	-	1.547	86.67 %
	168	432	30	462	93.33 %
MinGenPar-6	0	772	-	772	66.67 %
	144	492	11	503	86.67 %
	79	35	7	42	93.33 %
MinGenPar-7	0	167.451	-	167.451	66.67 %
	5.412	96.433	295	96.728	86.67 %
	1.754	958	242	1.200	93.33 %
MinGenPar-8	0	75.060	-	75.060	66.67 %
	5.344	41.404	375	41.779	86.67 %
	966	253	188	441	93.33 %
MinGenPar-9	0	82	-	82	66.67 %
	24	42	2	44	86.67 %
	24	7	2	9	93.33 %
MinGenPar-10	0	9.438	-	9.438	66.67 %
	697	6.569	50	6.619	86.67 %
	277	186	65	251	93.33 %

Cuadro 4.4: Tiempos de ejecución (en segundos) de los resultados obtenidos incrementando el número de cores para la ejecución en paralelo.

Problema	Número de cores				
	16	32	48	64	80
MinGenRd-1	5	4	3	3	3
MinGenRd-2	310	275	199	190	197
MinGenRd-3	2.742	2.589	2.122	2.078	2.075
MinGenRd-4	512	447	444	440	446
MinGenRd-5	598	462	416	445	442
MinGenRd-6	50	42	34	35	34
MinGenRd-7	1.457	1.200	1.078	1.010	1.012
MinGenRd-8	499	441	432	430	425
MinGenRd-9	9	9	7	7	7
MinGenRd-10	267	251	196	194	195

Cuadro 4.5: Algoritmo MinGenPar aplicado sobre un *dataset* real.

	Atrib	Implicaciones	BOV	Cores		
	126	1.587	1.481	64		
Problema & Método	Subp	División$_t$(s)	Paralelo$_t$(s)	Total$_t$(s)	Nodos	MinGens
mushrooms-parallel	224	152	9	161	81.363	17.127

Parte II

Sistemas de Recomendación Conversacionales

Capítulo 5

Sistemas de Recomendación

En la actualidad, suele ser difícil encontrar un día en el que no hayamos participado en situaciones donde intervenga algún tipo de recomendación. En este sentido, la ciencia y la tecnología han colaborado en el desarrollo y expansión de los sistemas de recomendación (SR) y a establecer un claro campo de conocimiento dentro de la gestión de la información.

Un SR es un sistema inteligente que proporciona a los usuarios una serie de sugerencias personalizadas (recomendaciones) sobre un determinado tipo de elementos (ítems). De forma general, los SR estudian las características de cada usuario e ítem, y mediante un procesamiento de los datos, encuentra un subconjunto de ítems que pueden resultar de interés para el usuario. Una de las referencias más notables en el campo de los SR la encontramos en el libro de Adomavicius y otros [2].

Desde que el primer SR hizo su aparición en el mundo de las tecnologías de la información [56, 101], los SR han estado en continua evolución durante los últimos años [1]. Sin embargo, es con la expansión de las nuevas tecnologías cuando han tenido un acercamiento más directo a la mayor parte de la sociedad debido a su capacidad para realizar todo tipo de recomendaciones sobre diversos elementos al alcance de todos (libros [31],

documentos [97], música [69], turismo [19], películas[1], etc.).

En la actualidad, los SR constituyen un claro campo de investigación y estudio como demuestran el gran número de trabajos que se están realizando [35, 115] y cuya cantidad continúa aumentando día a día. Además, la relevancia de estos sistemas no se limita al ámbito investigador. Actualmente, muchos SR ya han sido implantados con éxito en fuertes entornos comerciales a nivel mundial. Este es el caso de empresas líderes en el sector como pueden ser Amazon [74], LinkedIn [100] o Facebook [121], que han realizado fuertes inversiones con el fin de generar mejores SR. Estas situaciones ponen de manifiesto la gran importancia de estos sistemas en ambas vertientes de la sociedad actual.

Tras esta introducción al concepto SR y la importancia que abarca en los sistemas actuales, durante el resto de este capítulo vamos a ahondar en las características que intervienen en este tipo de sistemas. En este sentido, nuestra intención es doble; por un lado conseguimos aumentar la naturaleza autocontenida de la tesis, y por otro introducimos una serie de conceptos que serán claves para razonar los capítulos posteriores. En concreto, en primer lugar, se presenta una breve clasificación de las diferentes técnicas de recomendación que existen y la justificación de la elegida para el desarrollo de un SR en nuestro caso particular 5.1. A continuación, se presenta una serie de problemas de los que suelen adolecer este tipo de sistemas 5.2. Finalmente, la última parte está dedicada a presentar las medidas que se utilizan para evaluar el rendimiento de estos sistemas 5.3.

5.1 Técnicas de recomendación

Existen numerosos tipos diferentes de SR que normalmente se clasifican atendiendo a cómo se llevan a cabo las recomendaciones. Los más conocidos y extendidos son los sistemas de filtrado colaborativo (denominados CF, por sus siglas en inglés: *Collaborative Filtering*) y los sistemas basados en contenido (denominados CB, por sus siglas en inglés: *Content-Based*).

[1] https://www.movielens.org

Los primeros SR de filtrado colaborativo apuntan a GroupLens [66] o Video Recommender [56]. Este tipo de SR basa su funcionamiento fundamentalmente en las valoraciones que otros usuarios han otorgado a los elementos disponibles, y en realidad, la mayoría de los SR actuales utilizan técnicas de filtrado colaborativo en algún aspecto. Por su parte, los SR basados en contenido se basan en categorizar los ítems a recomendar, proporcionando resultados que tengan características similares a otros que han sido bien valorados anteriormente por el usuario. La mención de los SR basados en contenido es esencial por dos motivos; en primer lugar, es un tipo de SR que ha alcanzado una gran importancia en la actualidad debido a la tremenda expansión de las redes sociales, donde los usuarios introducen multitud de información de forma masiva, y en segundo lugar, aportará características al sistema desarrollado en esta tesis.

Además, en los últimos años ha habido un gran crecimiento de los SR contextuales [106], capaces de tener en cuenta información relevante para la recomendación como puede ser la hora, el lugar, la compañía, la ubicación, etc. Existe otro grupo denominado SR demográficos [7] que clasifican a los usuarios según diferentes parámetros personales (edad, localización, etc.) y realizan las recomendaciones teniendo en cuenta el grupo demográfico al que pertenece el usuario. Estos sistemas no requieren información histórica, sin embargo, requieren información que pueda ser sensible para la privacidad del usuario.

Por otro lado encontramos un tipo de recomendador de vital importancia para el trabajo de esta tesis, son los denominados SR basados en conocimiento (KB, por sus siglas en inglés: *Knowledge-Based*) [79]. Estos sistemas gestionan el conocimiento inherente a los datos y revelan cómo un ítem puede satisfacer la necesidad del usuario, es decir, utilizan un método de razonamiento para inferir la relación entre una necesidad y una posible recomendación.

Llegamos ahora a los SR más importantes desde el punto de vista de este trabajo, los denominados SR conversacionales [47, 70]. Estos están estrechamente relacionados con los conceptos de recomendador basado en críticas [22] y recomendaciones de información [122].

En estos sistemas, se aplica un proceso iterativo en el cual el usuario selecciona una o más características que desea que los ítems verifiquen. Esta forma de proceder a través de la interacción con el usuario, contrasta con la tendencia histórica de los SR de dar una recomendación con sólo una consulta. A partir de esta interacción y usando diferentes técnicas, el sistema progresa eligiendo (o incluso prediciendo) elementos que concuerdan con las preferencias del usuario. La ventaja principal de este tipo de SR es que a partir de las respuestas del usuario, el sistema es capaz de refinar la recomendación en cada ronda sucesiva del diálogo. Resaltamos este tipo de SR porque será la estrategia principal sobre el que se sustenta el desarrollo de SR realizado y que ha dado lugar a una de las contribuciones que avalan esta tesis [14].

Es posible estudiar más estrategias de recomendación; para ello, podemos encontrar una recopilación más detallada, junto con las principales tendencias, en dos de las referencias más importantes de este campo de conocimiento [1, 17].

En párrafos anteriores hemos podido apreciar que existen diferentes tipos de estrategias para los SR, sin embargo, la historia ha demostrado ampliamente que la mejor alternativa consiste en combinar características de diferentes tipos de SR para generar híbridos que se beneficien de las ventajas de cada uno de ellos [32]. Ésta ha sido exactamente nuestra intención principal, la cual, una vez introducidos los diferentes tipos de recomendadores, ya estamos en condiciones de establecer.

Nuestro trabajo ha culminado en un SR híbrido que combina las siguientes técnicas de recomendación:

- **SR basados en conocimiento.** En virtud de nuestro estudio de extracción de conocimiento de los datos utilizando FCA, los conjuntos de implicaciones y los operadores de cierre.

- **SR basados en contenido.** Necesario ya que para aplicar las técnicas lógicas empleadas necesitamos en primera instancia información sobre la que trabajar.

- **SR conversacionales.** Se presenta como estrategia central sobre la que aplicar y utilizar las dos anteriores respectivamente.

Como veremos a continuación, esta combinación de estrategias nos va a permitir abordar el denominado problema de la dimensionalidad que podemos encontrar frecuentemente al trabajar con SR.

5.2 Problemas comunes

Si bien es cierto el éxito que están alcanzando los SR, también tenemos que añadir que estos sistemas no están exentos de problemas y dificultades. En esta sección vamos a hacer una breve recopilación de aquellos que tienen una mayor presencia e importancia. Tal es el caso de uno de los problemas más comunes, el denominado arranque en frío o *cold-start* [41], que aparece cuando un nuevo usuario o ítem se incluye en el sistema. Estos nuevos elementos carecen de información propia y por tanto no es posible hacer predicciones sobre ellos. Un estudio muy completo al respecto de las soluciones propuestas a este problema puede verse en [116].

En otros casos, el problema del SR llega de la mano de la escasez de datos [51], ya que no siempre se posee la cantidad de información deseable para los ítems del sistema. Otros problemas pueden venir de la mano de la suplantación de la identidad (en inglés *Shilling attacks*). En estos problemas o ataques, el usuario puede hacer un uso malicioso del sistema para obtener beneficio propio. Es un problema muy difícil de detectar [132] o impedir, sin embargo, cada vez se desarrollan más técnicas para hacerle frente y es un gran campo de investigación dentro de los SR [139, 141].

Además, existen muchos otros problemas como son: privacidad [42], oveja girs/negra o *black-sheep* [46], sobreespecialización [75], escalabilidad [60], postergación (en inglés, *long-tail items*) [120], dimensionalidad [105], etc.

En concreto, nuestro trabajo ha estado orientado a resolver este último problema, la dimensionalidad cuando estamos trabajando sobre SR. Este problema, también conocido como *the curse of dimensionality phe-*

nomenon [89, 105] aparece cuando es necesario trabajar sobre datasets con un alto número de características (variables o atributos). De forma intuitiva, podríamos introducirlo de la siguiente manera. Cuando hay pocas columnas de datos, es relativamente fácil para los algoritmos realizar tareas de tratamiento inteligente de la información como: aprendizaje automático, *clustering*, clasificación, etc. Sin embargo, a medida que aumentan las columnas o características de nuestros ítems, se vuelve exponencialmente más difícil hacer labores predictivas con un buen nivel de precisión. El número de filas de datos necesarias para realizar cualquier modelado útil aumenta exponencialmente a medida que agregamos más columnas a una tabla.

La maldición de la dimensión representa un obstáculo importante a la hora de resolver problemas de gestión de la información que se plantean en el contexto del aprendizaje automático. Como ejemplo concreto encontramos el algoritmo de los k-vecinos, muy utilizado en SR [111], en el cual al aumentar la dimensión, la distancia al vecino más próximo crece.

Este problema también se hace latente en el campo de los SR al administrar grandes volúmenes de información sobre los que poder realizar recomendaciones al usuario teniendo en cuenta que el número de alternativas es muy elevado. Para abordar este problema, podemos encontrar muchos trabajos en la literatura sobre la reducción de la dimensión de la información, especialmente mediante selección de características *(feature selection)*, que pueden ayudarnos a descartar aquellas características que no son merecedoras de ser considerados según diferentes criterios. De hecho, estas técnicas ya se aplican en otras áreas como son: algoritmos genéticos o redes neuronales, normalmente centrándose en la aplicación de un proceso automatizado que se aplique de una vez *(batch mode)* mediante selección de características.

Suele ser habitual que para realizar una selección de características por parte del SR, el usuario tenga que introducir y seleccionar información del sistema una y otra vez. Esto constituye un problema dado que pueden existir artículos para los cuales el número de características que los definen sea muy elevado y en consecuencia, incomode la correcta interacción del

usuario con el sistema, poniendo de manifiesto de nuevo cómo la alta dimensionalidad constituye un problema para los SR.

En definitiva, nuestro objetivo es abordar el problema de la alta dimensionalidad en los SR a través de un proceso de selección de atributos por parte del usuario mediante un SR conversacional que utilice características de los SR basados en contenido y en conocimiento.

Un trabajo interesante en esta área es [61], que establece la idoneidad de los enfoques basados en el conocimiento para los procesos conversacionales. En particular, estos autores utilizan el razonamiento basado en restricciones, en lugar de nuestro enfoque basado en la lógica. Además, este trabajo trata sobre concepto de optimización de consultas, análogo al aplicado en nuestra propuesta. Otro trabajo notable es [122], que comparte nuestro objetivo de disminuir el número de pasos de la conversación. Los autores proponen métricas acerca del número de pasos de la conversación y tasas de poda, ambos muy similares a los utilizados en nuestro trabajo como veremos más adelante. Por otro lado, en [21], los autores demuestran cómo la posibilidad de que sea el usuario el encargado de la selección de atributos supera al hecho de que sea el sistema mismo el encargado de dicha selección. Este hecho respalda nuestro enfoque en el cual el humano experto guía la conversación y el proceso de selección de características.

5.3 Evaluación de los sistemas de recomendación

Desde que se inició la investigación de los SR, la evaluación de las predicciones y recomendaciones se ha convertido en un aspecto muy importante [20, 54]. Los SR requieren medidas de calidad y métricas de evaluación [49] para conocer la calidad de las técnicas, métodos y algoritmos para las predicciones y recomendaciones. Las métricas de evaluación [55] y los *frameworks* de evaluación [16] facilitan la comparación de varias soluciones para el mismo problema.

No obstante, teniendo en cuenta la gran variedad de factores que pueden intervenir en el funcionamiento de los SR, la primera tarea a llevar a cabo es analizar qué tipo de métrica es la que mejor se adapta al SR que quer-

emos evaluar. Con el fin de medir la calidad de los resultados de las recomendaciones de un SR, es habitual utilizar el cálculo de algunas de las métricas de predicción de errores más comunes. El uso de estas métricas nos da una forma de comprobar la eficacia de SR calculando una medida del error en las predicciones. Entre estas métricas destacan el error absoluto medio (MAE) y sus métricas relacionadas: error cuadrático medio, raíz del error cuadrático medio (RMSE), el error medio absoluto normalizado (NMAE) [137].

También son muy comunes aquellas métricas aplicadas a la precisión en las clasificaciones [6]. Dado que gran parte de los SR basa su funcionamiento en el concepto de valoración, la aplicación de estas medidas para evaluar la precisión del recomendador es una tarea prácticamente imprescindible es gran cantidad de SR. Las métricas más destacadas en este grupo son Precisión, Memoria y F1 [109].

Además de las métricas mencionadas, existen otras tantas que también pueden ser de aplicación como el coeficiente de correlación de Pearson, (en inglés *Pearson Correlation Coefficient*, PCC), coeficiente de correlación de Spearman, estabilidad, fiabilidad, originalidad, diversidad, ...

No obstante, hay que tener en cuenta que dependiendo del SR con el que estemos trabajando, la evaluación habrá que llevarla a cabo utilizando aquellas métricas, que por su naturaleza y significado, tengan cabida en relación al SR que se desea evaluar. En este sentido, en el capítulo siguiente 6.3 aplicaremos una serie de métricas relacionadas con los SR conversacionales para evaluar el rendimiento del SR desarrollado y justificaremos el porqué otras métricas tales como algunas de las introducidas en esta sección, no son susceptibles de aplicación en nuestro marco concreto de trabajo.

Capítulo 6

Desarrollo de SR conversacional

Este capítulo va a estar dedicado a mostrar el proceso, las mejoras conseguidas y los principales resultados que han acompañado a la consecución del SR desarrollado y que ha sido la parte fundamental de uno de los trabajos publicados que sustentan esta tesis [14]. Es por tanto intención principal de este capítulo el mostrar cómo se ha utilizado la base teórica adquirida para culminar en una aplicación de ingeniería, y en definitiva, para transmitir el conocimiento a las aplicaciones.

El texto va a estar repartido en tres secciones. La primera 6.1 y la segunda 6.2 están dedicadas a presentar el SR desarrollado y su funcionamiento respectivamente, mientras que en la última 6.3 presentaremos las medidas de evaluación utilizadas para el caso concreto del sistema desarrollado.

6.1 Presentación: *sicumaRS*

Si bien cronológicamente el último paso del desarrollo fue establecer un nombre para nuestro SR, por comodidad para con el texto, esta vez vamos a proceder al contrario; presentamos en primera instancia a *sicumaRS* como el SR creado a partir del trabajo de esta parte de la tesis.

Como hemos mencionado con anterioridad, *sicumaRS* va a ser un SR híbrido que va a combinar las estrategias de los SR conversacionales, basa-

dos en conocimiento y basados en contenido. Su finalidad va a ser abordar el problema de la dimensionalidad por medio un tratamiento eficiente de la información haciendo uso de FCA, las implicaciones y los operadores de cierre. En concreto haremos uso de la SL_{FD} y del algoritmo del cierre Cls.

Antes de entrar en el funcionamiento de *sicumaRS*, hay que tener en cuenta que lo primero que vamos a necesitar son *datasets* con elementos sobre el que poder hacer recomendaciones. Es necesario que estos *datasets* contengan la información según una representación binaria, es decir, el valor de cada uno de los atributos asociados a cada elemento del sistema será un valor binario que representará si ese elemento verifica ese atributo o no lo hace. Para entender este aspecto, veamos el siguiente ejemplo.

Ejemplo 6.1.1. *Supongamos el dataset Auto MPG Data Set*[1] *accesible desde la página web de la Universidad de California, Irvine (UCI)*[2] *que básicamente contiene características del motor de un vehículo con la intención de predecir el consumo (las siglas MPG proceden del inglés Miles Per Gallon, o millas por galón.). Sobre este dataset se hace la adaptación necesaria para convertir la información multivaluada en información binaria, de forma que se establecieron unos umbrales que asignaban a cada característica del dataset un valor de 1 si superaba el umbral, 0 en otro caso. De esta forma sicumaRS ya puede utilizar este dataset modificado. La Tabla 6.1 muestra un extracto de la información original utilizada y su correspondiente adaptación.*

Una vez tengamos los *datasets* sobre los que vamos a trabajar, *sicumaRS* necesita conocer las relaciones que hay entre sus atributos por medio de implicaciones lógicas, pero para ello existen aplicaciones capaces de extraer el conjunto de implicaciones que se verifican en un determinado *dataset* [57, 133, 134].

[1] http://archive.ics.uci.edu/ml/datasets/Auto+MPG
[2] http://archive.ics.uci.edu/ml/

Cuadro 6.1: Extracto del *dataset* Auto MPG Data Set de la UCI

Nombre	MPG	Cilindrada	Potencia	Peso	Aceleración
Chevrolet Monte Carlo	15.0	8	150.0	3761	9.5
Buick Estate Wagon	14.0	8	225.0	3086	10.0
Toyota Corolla Mark II	24.0	4	95.00	2372	15.0
Plymouth Duster	22.0	6	95.00	2833	15.5
...					
Chevrolet Monte Carlo	1	1	0	1	0
Buick Estate Wagon	1	1	0	1	0
Toyota Corolla Mark II	0	0	1	0	1
Plymouth Duster	0	1	1	0	1
...					

6.2 Funcionamiento

Una vez presentado *sicumaRS* y la información de entrada que debe tener para funcionar, pasamos ahora a explicar el proceso conversacional junto con un esquemas conceptuales que faciliten su comprensión.

Fundamentalmente, la naturaleza de sistema conversacional que se manifiesta en *sicumaRS* nos lleva a que el proceso se desarrolle según una serie de etapas que pasamos a enumerar:

(i) La interacción del usuario con el sistema comienza cuando el usuario elige un atributo con el que realizar la búsqueda. En principio, *sicumaRS* limita la elección de atributos a uno en cada paso de la conversación. De esta forma, podemos apreciar más fácilmente cómo funciona el sistema. No obstante, pruebas más avanzadas en las que hemos permitido la elección de múltiples atributos por cada paso han demostrado que el sistema mantiene su correcto funcionamiento e incluso aumenta más rápidamente la poda de información superflua.

(ii) Una vez seleccionado el atributo, el proceso entra en el algoritmo Cls para calcular el cierre del conjunto de atributos y al mismo tiempo, el conjunto de implicaciones que quedan fuera del cierre.

(iii) Una vez el Cls termina, se muestra una primera recomendación. Esta recomendación es una lista de resultados con los elementos del

dataset que verifican el atributo seleccionado. Para conseguir esta lista el sistema realiza una consulta a base de datos solicitando aquellos elementos que verifiquen el atributo.

(iv) En este punto, el usuario puede terminar el diálogo en caso de que ya esté satisfecho con el resultado o bien, puede continuar interactuando con el sistema por medio de la elección de un nuevo atributo. La ganancia en este momento se produce de la siguiente manera.

Para los sucesivos pasos del diálogo, hemos reducido el número de atributos disponibles eliminando aquellos que estén incluidos en el cierre debido a que son implícitos con respecto a la selección realizada. Como consecuencia, esos atributos implícitos no aparecerán en las sucesivas interacciones con el usuario, liberándolo así de tener que tratar con información redundante, lo cual actúa en beneficio directo a aliviar el problema de la alta dimensionalidad de la información.

No obstante, si bien es cierto que esta mejora puede alcanzarse con el algoritmo clásico del cierre [78], la mayor ventaja de usar el Cls es que *simultáneamente* estamos reduciendo el número de implicaciones, y por tanto, en cada nuevo paso del diálogo no necesitamos volver a extraer el nuevo conjunto de implicaciones, lo cual es una tarea de *data-mining* con coste exponencial, sino que podemos continuar la interacción a partir aquí, donde tanto atributos como implicaciones han sido reducidas, y por tanto con coste *lineal*.

(v) Finalmente, el usuario puede decidir si queda satisfecho con la recomendación obtenida, o bien si quedan más atributos disponibles para elegir, puede volver al paso 1 para continuar el diálogo y refinar la recomendación; en caso contrario el diálogo termina por no quedar más atributos que seleccionar.

Las Figuras 6.2 y 6.1 nos ayudan a entender con mayor facilidad la sucesión de pasos que da el sistema conversacional para interactuar con el usuario y proponer recomendaciones. La primera representa el diagrama de

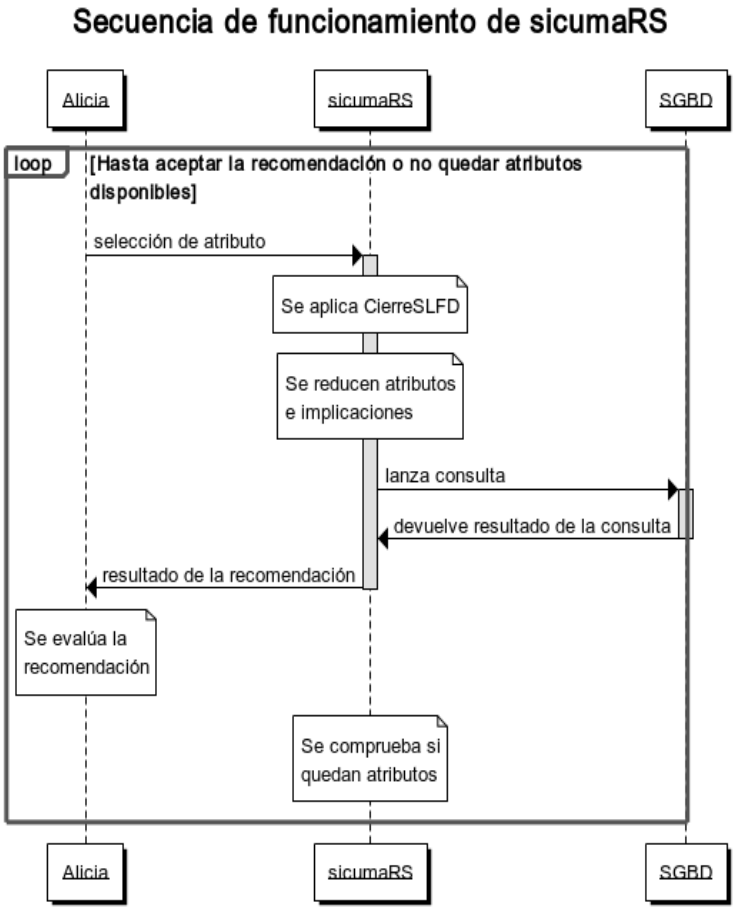

Figura 6.1: Diagrama de secuencia del funcionamiento de *sicumaRS*.

secuencia del proceso en el cual intervienen el usuario (Alicia), el SR (en este caso *sicumaRS*) y el sistema gestor de bases de datos (SGBD). Por otro lado, la Figura 6.2 nos muestra un esquema conceptual aplicado a un caso determinado de la investigación realizada. En concreto, ese esquema representa el funcionamiento de *sicumaRS* sobre un *dataset* que agrupa información sobre anomalías hematológicas y fenotipos. Por tanto en el esquema podemos ver como se desarrolla un hipotético diálogo hasta alcanzar una lista de anomalías aceptable. Esta figura tiene una importancia mayor añadida ya que además de servir para ilustrar el funcionamiento del SR, muestra la ganancia que obtenemos al utilizar el Cls frente a implementaciones clásicas. Los resultados de aplicación sobre este *dataset* concreto pueden verse con mayor detalle en [14].

Finalmente, vamos a terminar este apartado mostrando un ejemplo de aplicación del SR conversacional utilizando el *dataset* de restaurantes que muestran los autores en [71].

Este *dataset* va a contar con 6 tipos diferentes de establecimientos que se relacionan con 11 facilidades que pueden ofrecer. La Tabla 6.2 muestra un extracto del aspecto del *dataset*.

Cuadro 6.2: *Dataset* de restaurantes (extracto)

Tipo de local	Abierto	Cerrado	Tranquilo	Animado	Pintoresco	Barato
Restaurante común	✓	✓		✓		
Estrella Michelín		✓	✓			
Burger		✓		✓		✓
Bar Tapas		✓		✓	✓	✓
Pizzería		✓		✓		✓
Chiringuito Playa	✓			✓	✓	

Como consecuencia, vamos a tener un conjunto de atributos $U = \{$*Abierto, Cerrado, Tranquilo, Animado, Pintoresco, Barato, Moderado, Caro, Aire Acondicionado (AC), Vistas, Terraza*$\}$. Además, una vez que hayamos utilizado alguna de las técnicas existentes para extraer las implicaciones que se verifican en la Tabla 6.2 obtenemos el siguiente conjunto de implicaciones:

- $Terraza \rightarrow Animado$;

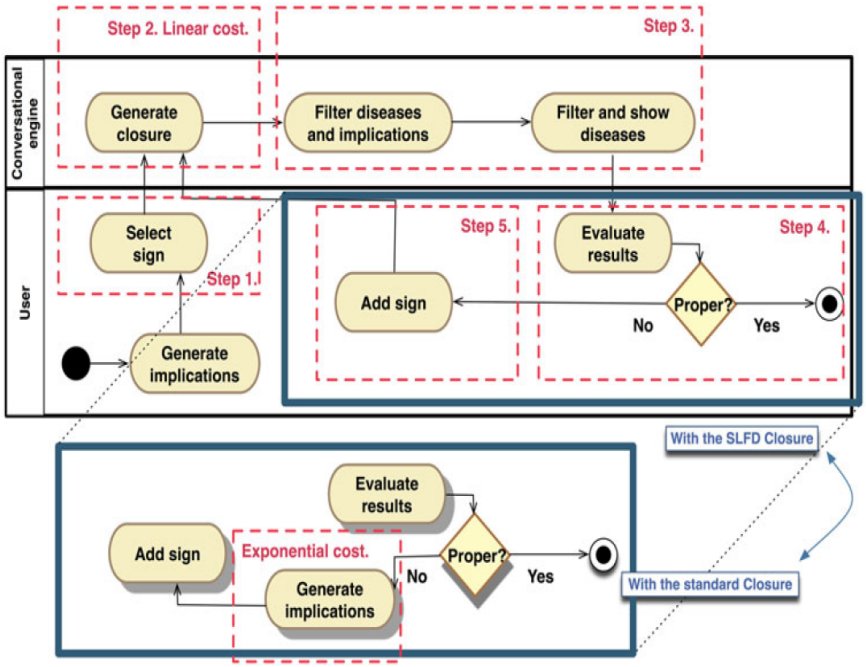

Figura 6.2: Diálogo entre el usuario y *sicumaRS* que muestra el funcionamiento del sistema y la ganancia obtenida por el algoritmo Cls frente al algoritmo clásico del cierre. Fuente [14].

- $Vistas \rightarrow Abierto, Animado, Pintoresco, Moderado, Terraza$;
- $AC \rightarrow Cerrado$;
- $Caro \rightarrow Cerrado, Tranquilo, AC$;
- $Moderado \rightarrow Pintoresco$;
- $Barato \rightarrow Cerrado, Animado$;
- $Animado, Pintoresco, Terraza \rightarrow Abierto, Moderado, Vistas$;
- $Animado, Pintoresco, Moderado \rightarrow Abierto, Vistas, Terraza$;
- $Tranquilo \rightarrow Cerrado, AC$;
- $Cerrado, Pintoresco, AC \rightarrow Tranquilo, Moderado$;
- $Cerrado, Pintoresco, Moderado \rightarrow Tranquilo, AC$;
- $Cerrado, Animado \rightarrow Barato$;
- $Cerrado, Animado, Barato, Terraza \rightarrow AC$;
- $Cerrado, Animado, Barato, AC \rightarrow Terraza$;
- $Abierto \rightarrow Animado, Pintoresco, Moderado, Vistas, Terraza$;

En definitiva, el SR conversacional va a contar con unos datos de partida formados por: un conjunto de 6 restaurantes, 11 posibles atributos y 15 implicaciones que se verifican en los datos. Por tanto, pasemos ahora a realizar una simulación de posible diálogo.

Ejemplo 6.2.1. *Supongamos que un usuario busca una recomendación sobre un restaurante para cenar. En primer lugar, busca que el lugar cuente con un atmósfera animada. Además, debido al buen clima reinante esta noche, es preferible que el establecimiento cuente con terraza. Para terminar, un precio moderado pondría la guinda al pastel.*

De esta forma, el usuario comienza la interacción con el sistema introduciendo sus preferencias: Animado*,* Terraza *y* Moderado *de forma sucesiva y los resultados que se obtienen los podemos ver desglosados en la Tabla 6.3.*

Cuadro 6.3: Desglose paso a paso del diálogo entre el usuario y *sicumaRS*.

Iter.	Selección	Cierre	Atributos	Implics.	Resultados
1	Animado	{Animado}	{CS, Pintoresco, Barato, OS, Tranquilo, Moderado, Caro, AC, Vistas, Terraza}	14	{Burguer, Pizzería, Tapas, Playa}
2	Terraza	{Animado, Terraza}	{CS, Pintoresco, Barato, OS, Tranquilo, Moderado, Caro, AC, Vistas}	13	{Pizzería, Playa}
3	Moderado	{Animado, Terraza, Moderado, OS, Pintoresco, Vistas}	{CS, Barato, Tranquilo, Caro, AC}	7	{Playa}

De este ejemplo podemos extraer algunas conclusiones interesantes que recogemos en la lista siguiente:

- Primero y más importante; hemos conseguido que la interacción con el usuario sea más sencilla ya que no hay necesidad de tener que tratar con todos los atributos posibles a cada paso de la conversación. En cada momento, el sistema ha ido eliminando aquellos atributos que por la selección de otros se encuentran implícitamente incluidos. Gracias a ello, el diálogo entre el usuario y el sistema se hace más dinámico por dos razones:

  (i) Se evita que los usuarios tengan que navegar entre atributos redundantes.

  (ii) La lista de atributos disponible depende de las elecciones previas, por tanto, los resultados pueden diferir entre un acceso al sistema y otro, evitando de esta forma obtener siempre los mismos resultados una y otra vez.

- Gracias al algoritmo Cls, conseguimos que en cada paso de la conversación, una vez aplicado, se reduzcan tanto el número de atributos disponible como el número de implicaciones, lo que redunda en una reducción de los tiempos de respuesta del sistema.

- También hemos de comentar que puede ocurrir que el usuario obtenga una recomendación satisfactoria antes de que la lista de resultados tenga una longitud manejable. En estos casos, el diálogo entre el usuario y el sistema puede acabar con una lista de posibles alternativas todavía muy extensa, pero en cualquier caso el sistema habrá guiado de forma eficiente al usuario hasta este punto. De hecho, en caso de no estar del todo satisfecho todavía, la solución pasa por continuar eligiendo atributos hasta que la lista de resultados sea más tratable, pero en cualquier caso, el sistema depende de la iniciativa del usuario.

6.3 Medidas de evaluación

Como es razonable, todo sistema debe someterse a una serie de evaluaciones que confirmen su viabilidad y su utilidad. Con *sicumaRS* hemos realizado este tipo de evaluaciones y para ello vamos a pasar a definir qué medidas se han utilizado para llevar a cabo esta evaluación.

Recordemos que en apartados sucesivos se analizaron muchas de las diferentes métricas de evaluación de los SR que existen en la literatura. Sin embargo, no todas son adecuadas para todos los SR; dependiendo del SR habrá unas que sean razonables de aplicar y otras que no tengan cabida. En nuestro caso, dado que estamos tratando con un SR conversacional, puede ser evidente que la primera medida que podemos aplicar es calcular el número de pasos que se producen en la conversación [81] como hemos estado viendo en los ejemplos anteriores. Por contra, otras métricas tan populares como son *Precision* y *Recall* [50] no son adecuadas de aplicar en nuestro caso porque obtendríamos siempre valores máximos en ambas métricas y la razón es la siguiente:

- En primer lugar, cualquier ítem de la lista de resultados, verifica los

atributos seleccionados ya que la consulta que se lanza a la base de datos para obtener la lista de ítems contiene esas restricciones.

- Y en segundo lugar, a cada paso del diálogo, el sistema devuelve todos los ítems que verifiquen la selección de atributos establecida por el usuario.

Sucede una situación similar cuando estamos hablando de otras métricas muy utilizadas como son MAE o RMSE. Estas métricas basadas en valoraciones no tienen cabida en nuestro sistema puesto que no existen valoraciones con la que el sistema trabaje.

Asimismo, no existe la necesidad de de considerar métricas referentes a la exactitud de los resultados ya que *sicumaRS* no es un modelo de predicción, su funcionamiento está basado en implicaciones y eso nos asegura un 100 % de exactitud en las respuestas.

Afortunadamente, existen otras medidas que pueden reflejar los resultados de los experimentos. Las explicamos a continuación:

Número de pasos (N)

Como su nombre indica, esta métrica registra el número de pasos que se dan en el diálogo y en consecuencia, el número de atributos seleccionados por el usuario. Nos proporciona una visión clara de si el sistema ha necesitado mucha o poca interacción para satisfacer la demanda del usuario. Hay que aclarar, sin pérdida de generalidad, que en los experimentos realizados, dependiendo del *dataset* sobre el que trabajemos, se ha establecido un tamaño máximo de la lista de resultados como indicador de aceptación por parte del usuario; por ejemplo, se establece que el usuario se considera satisfecho cuando una recomendación de hoteles contenga como máximo 10 hoteles.

$$N = |\text{Atributos seleccionados}|, \quad \text{donde } |A| \text{ representa el cardinal de A.}$$

Velocidad de poda en cada paso i (S_i)

Esta métrica evalúa el porcentaje de atributos que el sistema libera al usuario de tener en cuenta a lo largo de la conversación y de forma acumulativa de un paso al siguiente. Con esta métrica buscamos averiguar si los ratios de poda del algoritmo son mejores al principio de la conversación o en los pasos posteriores. Es una métrica para medir cuán rápido es el sistema reduciendo la sobrecarga de información.

$$S_i = \frac{|\text{Atributos del cierre}|_i - i}{|\text{M}|}$$

Siendo $i = 1, ..., N$, y M el conjunto global de atributos.

Reducción de atributos (P)

Esta última métrica nos informa de la reducción global de atributos que ha realizado el sistema al terminar el diálogo. La reducción de estos atributos es consistente ya que es fruto de la aplicación del algoritmo del cierre a partir de las selecciones realizadas por el usuario. Formalmente:

$$P = S_N$$

Capítulo 7

Experimentos realizados

A lo largo del texto, se han mostrado ejemplos 2.2.16 6.2.1 de aplicación del algoritmo Cls y en general del SR desarrollado, no obstante, en estos momentos en los que contamos con *sicumaRS* como un SR ya definido, abordamos en este capítulo una serie de experimentos más elaborados que nos permitan comprobar el funcionamiento de nuestro sistema y evaluar su rendimiento. La forma en que vamos a proceder con los experimentos la exponemos a continuación.

Para cada experimento vamos a realizar un test que consiste en simular 100 diálogos siguiendo al pie de la letra el proceso detallado en 6.2. No obstante, las simulaciones van a llevarse a cabo como diálogos aleatorios, es decir, cada diálogo se desarrollará eligiendo atributos de forma aleatoria del conjunto de atributos disponible a cada paso de la conversación. Esto puede verse desde diferentes perspectivas. En primera instancia, parece adecuada la estrategia aleatoria ya que así se garantiza la honestidad de los resultados al no haber posibilidad de inducir situaciones favorables para que el sistema obtenga mejores resultados. Sin embargo, proceder de manera aleatoria también puede obscurecer las virtudes de nuestro sistema.

Por un lado, imaginemos una situación en la que las elecciones aleatorias impliquen atributos que no guardan relación entre ellos en el *dataset*, entonces, el diálogo terminará rápidamente ya que puede que haya muy

pocos ítems (o incluso ninguno) que verifiquen tal selección de atributos y por tanto, el proceso terminará sin poder aplicar ninguna reducción de atributos. Ahora bien, supongamos un diálogo más realista en el que la elección de atributos tenga un relación más sensata. En este caso, durante el diálogo, el sistema será capaz de ir aplicando sucesivas podas para reducir la sobrecarga de información ya que existirá relación entre los atributos seleccionados. Por tanto, esta forma de proceder ensalzaría los beneficios de nuestro sistema.

En relación a la evaluación del sistema, las métricas que vamos a utilizar sobre cada simulación van a a ser: número de pasos de la conversación, velocidad de poda de atributos en cada paso y reducción total de atributos; tal y como se han presentado en 6.3.

Finalmente, se realizan una serie de repeticiones sobre cada uno de los experimentos de manera que cada número mostrado es fruto de un estudio estadístico a partir de los resultados obtenidos en esas repeticiones que nos permite extraer los resultados más fiables [45].

Dicho eso, el capítulo va a estar dividido en 4 bloques principales. Comenzaremos con un experimento sobre un *dataset* muy conocido en el campo de los SR: MovieLens10M 7.1. Es un *dataset* que contiene una gran cantidad de información sobre películas de la que vamos a usar sólo los títulos y géneros. En virtud de esto, trabajamos sobre una cantidad importante de elementos, pero sin contar con demasiados atributos. Continuaremos con otro *dataset* que contiene un menor número de elementos pero muy definidos mediante muchos atributos (Costa del Sol Hotels) y que además contiene información real extraída directamente desde la red 7.2. Para culminar esta sección, el último experimento utiliza un *dataset* (World Wide POIs) que supera ampliamente a los anteriores en cuanto al número de elementos y atributos y que al igual que el anterior, la información que presenta también es real 7.3.

Finalmente, la última sección nos trae la presentación y los resultados obtenidos sobre el *dataset* 7.4 utilizado en una de las publicaciones que avalan esta tesis [14]. Varias tablas y figuras acompañarán cada experimento con la intención de ilustrar más fácilmente los resultados obtenidos.

7.1 MovieLens *datasets*

MovieLens[1] es un proyecto desarrollado por el equipo de investigación GroupLens[2] del Departamento de Ciencias de la Computación e Ingeniería de la Universidad de Minnesota especializado en SR, comunidades online, bibliotecas digitales, ... En su dirección web se ayuda a la gente a elegir películas que deseen ver; grosso modo es un SR de películas con un tinte evidente de CF. Cuenta con cientos de miles de usuarios y de películas almacenadas en una colección muy rica de *datasets* que han sido un referente a la hora de probar el funcionamiento de otros sistemas en diferentes entornos [53].

En nuestro caso, nos vamos a centrar en su MovieLens10M *dataset*. Es un *dataset* totalmente accesible de forma gratuíta desde la página web y contiene información de más de 10.000 películas con sus respectivos géneros, duración, país, valoraciones, elenco, etc.

De toda esta información, vamos a generar un conjunto reducido para utilizarlo en nuestro *sicumaRS*. Concretamente, vamos a crear una tabla para emparejar cada una de las películas con todos los posibles géneros que existen en el *dataset* original. De esta forma, tendremos las películas en las filas de la tabla y los géneros como columnas; una película tendrá una marca de verificación en aquellos géneros en los que esté clasificada y nada en otro caso. La Tabla 7.1 muestra un pequeño extracto del conjunto final.

Cuadro 7.1: Extracto del *dataset* MovieLens 10M.

Título	Acción	Comedia	Crimen	Drama	Romance	...
Little City (1998)		✓			✓	
Driver, The (1978)	✓		✓			
Father of the Bride (1950)		✓				
Bio-Dome (1996)		✓				
Fast Runner, The (2001)				✓		
Overboard (1987)		✓			✓	
Get Rich or Die Tryin' (2005)	✓		✓	✓		
...						

[1] http://movielens.org
[2] https://grouplens.org

Para convertir esta información a valores de 1 ó 0 para nuestro SR sólo tenemos que cambiar los marcas de verificación por 1 y el resto por 0.

Los números de este *dataset* adaptado según se ha indicado, alcanzan los siguientes valores:

- Filas (items): 10.681 películas.
- Columnas (atributos): 19 géneros.
- Implicaciones: 245

Con esta información ya estamos en disposición de iniciar la interacción entre el usuario y *sicumaRS*. No obstante, antes de pasar al experimento general, veamos un ejemplo concreto de diálogo.

Ejemplo 7.1.1. *Supongamos en este caso que el usuario está buscando una película de acción, con experiencia IMAX y con algunos toques de misterio. Entonces, el usuario interactúa con el sistema introduciendo estas preferencias en la conversación. El resultado lo podemos apreciar viendo cuál ha sido diálogo que se ha producido en la Tabla 7.2.*

Cuadro 7.2: Resultados del diálogo entre *sicumaRS* y el usuario sobre el *dataset* MovieLens 10M.

Iter.	Selección	Cierre	Atribs.	Implics.	Items
1	Acción	{Acción, Thriller, Aventura}	16	121	1.473
2	IMAX	{Acción, Thriller, Aventura, Sci-Fi, IMAX, Comedia, Fantasía}	12	43	108
3	Misterio	{Acción, Thriller, Aventura, Sci-Fi, IMAX, Comedia, Fantasía, Misterio Crimen, Romance, Cine-Negro}	8	1	6

En el ejemplo anterior, podemos darnos cuenta de que incluso cuando interactuamos con un *dataset* muy grande como MovieLens10M, se hace más cómodo para el usuario el obtener una recomendación, ya que en cada paso estamos reduciendo el espacio de búsqueda. Efectivamente, 3 pasos son suficientes para obtener la recomendación entre todo el *dataset*.

En este sentido, si nos centramos en la primera iteración en la que se introduce la preferencia *Acción*, el conjunto de cierre contiene dos atributos más aparte del seleccionado. La siguiente iteración sigue la misma línea y el cardinal del conjunto cierre aumenta sustancialmente. Como consecuencia, podemos liberar el usuario de tratar con esos atributos en las iteraciones sucesivas, y así estamos reduciendo la sobrecarga de información. Además, como hemos mencionado antes, al mismo tiempo, también reducimos el número de implicaciones involucradas de forma significativa, lo cual acelera las consultas subsiguientes.

Una vez visto un caso concreto de diálogo con este *dataset*, pasamos ahora a realizar el experimento completo tal y como se ha establecido anteriormente, es decir, simulando 100 diálogos diferentes en los cuales los atributos se eligen de forma aleatoria. La Figura 7.1 muestra el resultado de este experimento dividido en 3 secciones:

a) Representa por medio de un diagrama de sectores el porcentaje de experimentos que han necesitado un determinado número de pasos para alcanzar una recomendación admisible.

b) Muestra el porcentaje total de atributos que se han reducido en el diálogo.

c) Muestra el porcentaje de atributos que se ha reducido a cada paso de la conversación, es decir, la velocidad de poda.

Al analizar la Figura 7.1(a) podemos apreciar como *sicumaRS* es capaz de guiar al usuario hasta una recomendación aceptable en 2 ó 3 pasos en la mayoría de los casos. Si tenemos en cuenta el tamaño de este *dataset*, estos resultados son altamente prometedores.

Respecto a la reducción de atributos global, la Figura 7.1(b) muestra que el sistema ha liberado al usuario de tener que interactuar con el 5-20 % de los atributos. Incluso tenemos algunas pruebas en las que la reducción ha llegado a alcanzar el 60 % de los atributos.

La velocidad de poda acompaña el resultado anterior moviéndose en un rango de entre el 2-10 % como muestra la Figura 7.1(c).

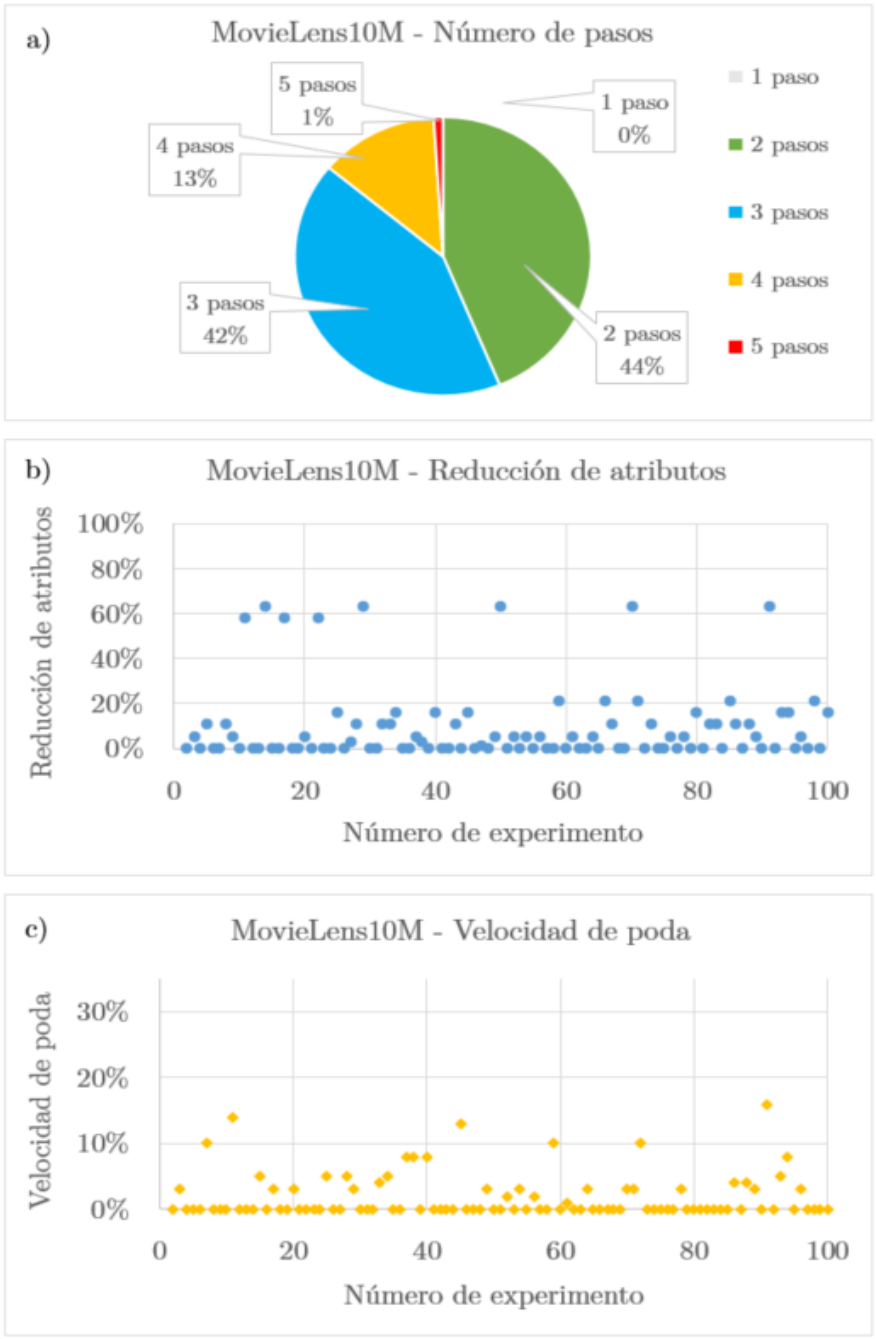

Figura 7.1: Resultado del experimento sobre el *dataset* MovieLens10M.

A la luz de estos resultados, podemos concluir que *sicumaRS* se comporta de forma notable a la hora de aliviar la sobrecarga de información en la interacción con el usuario.

Nota 7.1.2. *Aparte de los dataset que podemos encontrar disponibles en la web, a la hora de realizar experimentos nos vimos en la necesidad de poder crear nuestros propios datasets, igualmente utilizando información real, pero ahorrando los pasos de análisis y preparación de la información al formato de sicumaRS.*

A raíz de ello, se investigó el web-scrapping como una de las técnicas disponibles para extraer información de la red [18, 62]. El aprendizaje de esta técnica nos abrió la puerta a crear datasets con los que poder trabajar. Como ejemplo de algunos de ellos, tenemos el dataset de Hoteles Costa del Sol cuyo experimento pasaremos a analizar, y el de puntos de interés turístico (POIs Mundial) que veremos en la sección 7.3.

7.2 Hoteles Costa del Sol *dataset*

Este experimento tiene un gran interés para el usuario ya que la elección de un hotel, ya sea por vacaciones, trabajo o cualquier otro motivo, suele ser una decisión con un alto margen de maniobra en primera instancia. Por lo tanto, contar con un sistema que acelere la búsqueda de nuestras necesidades puede ser muy útil para ahorrar tiempo y esfuerzo. Para ello, presentamos un *dataset* que contiene más de 300 hoteles (extraídos del proyecto Costa del Sol Occidental[3]) cada uno de los cuales con 37 atributos diferentes. Aunque el *dataset* es muy disperso, la tabla final obtenida es digna de tener en consideración ya que está formada por información real que se utiliza actualmente por muchos turistas que visitan el sur de España.

Al igual que en el experimento anterior, vamos a generar una tabla que contiene los hoteles en las filas y las características o servicios en las columnas tal y como se muestra en la Tabla 7.3.

[3]http://www.costadelsoloccidental.org

Cuadro 7.3: Hoteles Costa del Sol *dataset* (extracto)

Nombre Hotel	AC	Bar	Gym	Internet	Masajes	Parking	...
Fuerte Estepona Suites	✓	✓	✓	✓	✓	✓	
Hotel Buenavista		✓				✓	
Hotel Paraiso	✓	✓				✓	
Apts Marriot Playa				✓			
Hotel Piedra Paloma		✓				✓	
Hostal Hospederia V Cent	✓	✓		✓		✓	
...							

Si bien es cierto que este *dataset* no contiene tantos elementos como el de MovieLens10M 7.1, sí que lo supera en número de atributos para cada ítem, y aún más importante, en el cardinal del conjunto de implicaciones. Los números finales son:

- Filas (items): 361 hoteles.
- Columnas (atributos): 37 servicios.
- Implicaciones: 1.507

Al igual que en el experimento anterior, antes de entrar en el experimento general, veamos un caso concreto.

Ejemplo 7.2.1. *Supongamos que una pareja joven quiere pasar un fin de semana descansando en un hotel y están visitando sitios web en un teléfono móvil. Los requisitos que buscan por parte del hotel son Spa y servicio de Belleza. Al introducir estos parámetros en el sistema, se produce un resultado con 16 hoteles. Un resultado así puede ser incómodo para la pantalla de un teléfono móvil común. Sin embargo, consideran incluir una sesión de Masaje también, por tanto, añaden la nueva preferencia y obtienen una lista de recomendación con 7 hoteles, eso ya es aceptable. El proceso se representa en la Tabla 7.4.*

En este experimento se aprecia claramente cómo el mecanismo de poda hace que la interacción sea más rápida, más cómoda y dinámica, ya que podría ser una ardua tarea tratar de elegir por un hotel entre cientos de

Cuadro 7.4: Resultados del diálogo entre *sicumaRS* y el usuario sobre el *dataset* Hoteles Costa del Sol

Iter.	Selección	Cierre	Atribs.	Implics.	Items
1	Spa	{Spa, Bar, Restaurante, Cafetería, Piscina, Jardines, Parking, AC}	29	799	96
2	Belleza	{Spa, Bar, Restaurante, Cafetería, Piscina, Jardines, Parking, AC} Belleza, Reuniones, Deporte, Animación}	25	559	16
3	Masaje	{Spa, Bar, Restaurante, Cafetería, Piscina, Jardines, Parking, AC, Belleza, Reuniones, Deporte, Animación, Masaje, Médico, Lavandería, Internet}	21	126	7

posibilidades con más de 30 servicios posibles cada una. Por el contrario, 3 pasos son suficientes para guiar a la pareja a sus mejores opciones. Estos resultados superan con creces los estudios previos [122] donde, incluso contando con un *dataset* hotelero más pequeño, el número de pasos necesarios es mayor.

Con la intención de preservar la completitud con el experimento anterior, la Figura 7.2 muestra el resultado para el experimento completo.

Respecto a este experimento, podemos ver que los resultados son ligeramente diferentes a los alcanzados sobre MovieLens10M 7.1. El primer punto notable del experimento es que esta vez hay casos en que la duración de la conversación ha aumentado y esto se debe a que el número de atributos posibles es mucho mayor ahora. Aún así, en la mayoría de los casos son necesarios sólo 2-3 pasos una vez más para conseguir una recomendación admisible.

La reducción de atributos es mayor en este caso con respecto a MovieLens10M, alcanzando tasas de poda entre 10-40 % en la mayoría de los casos, y de entorno al 85 % en algunos otros.

Del mismo modo, la velocidad de poda también es mayor como podemos apreciar en la Figura 7.2(c); en definitiva, consiguiendo valores que reflejan la utilidad del mecanismo de conversacional.

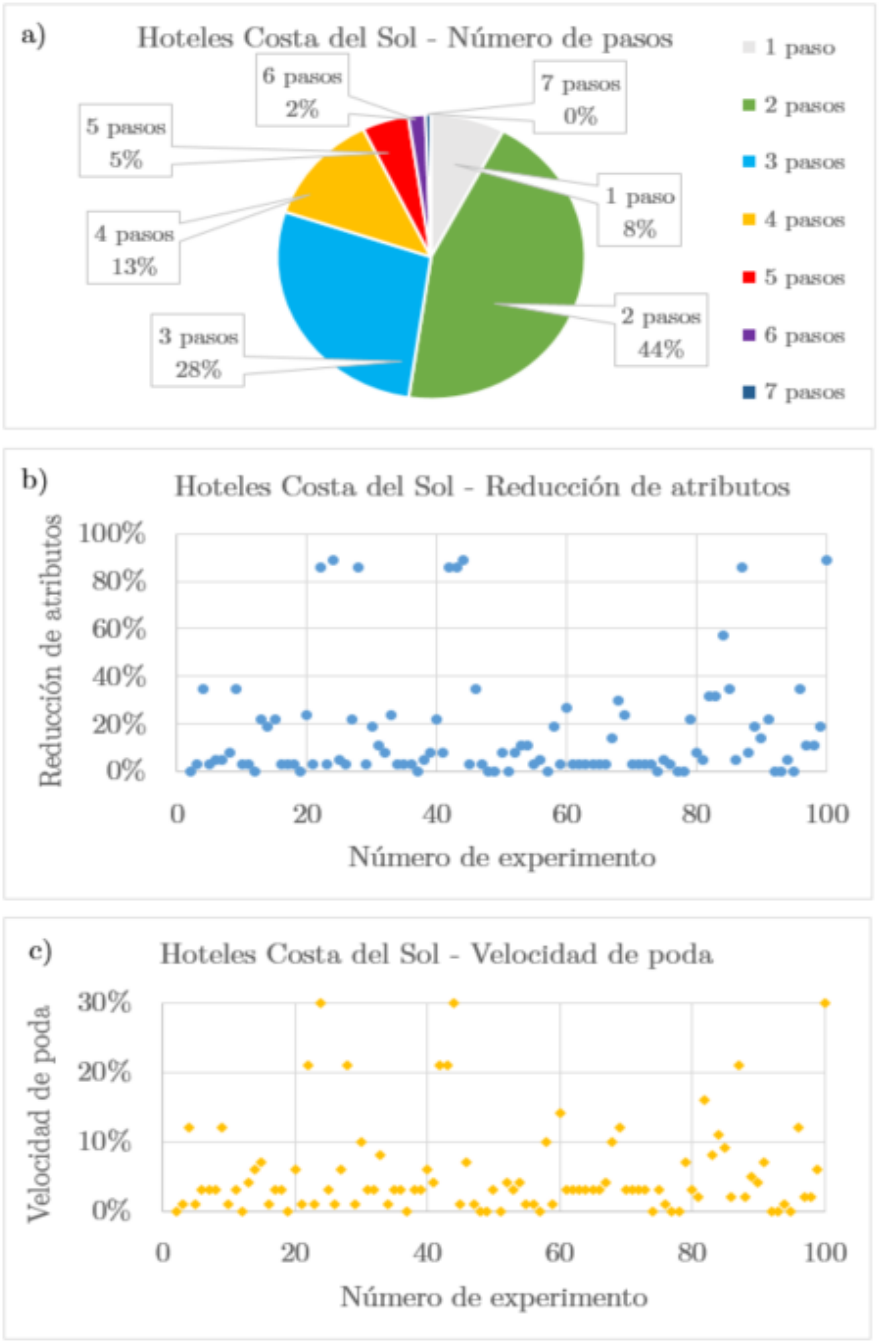

Figura 7.2: Resultado del experimento sobre el *dataset* Hoteles Costa del Sol.

7.3 POIs Mundial *dataset*

Finalmente, con la intención de crear un *datasete* más amplio que los anteriores tanto en el número de ítems como en el número de atributos, llegamos al último *dataset* generado, al cual hemos denominado POIs Mundial *dataset*. Manteniendo la idea de realizar pruebas más fidedignas, la información del *dataset* es información real, extraída del portal web Tripadvisor[4] con las técnicas mencionadas anteriormente. Este *dataset* va a contener información de multitud de puntos de interés (en inglés *Points Of Interest*) turístico de todo el mundo (Torre Eiffel, Estatua de la Libertad, Big Ben, Plaza Roja, Coliseo, etc.). En esta clasificación entran monumentos, parques, museos, iglesias, galerías de arte, teatros, complejos deportivos, ..., así hasta un total de más de 100 categorías diferentes. En la Figura 7.3 mostramos un ejemplo de algunos de los puntos de interés que podemos encontrar en Roma según la web y las categorías a las que pertenecen.

La generación de este *dataset* arrojó una serie de resultados muy satisfactorios. El primero fue constatar la capacidad adquirida para la generación de *datasets* de gran envergadura y con información real. De hecho POIs Mundial con un total de 17.400 puntos de interés (ítems), cantidad que sobrepasa notablemente a las 10.000 películas almacenadas en el histórico *dataset* de MovieLens10 analizado en 7.1. Además, cada elemento puede pertenecer a más de 100 categorías diferentes (en concreto 115 categorías diferentes), lo cual genera una tabla con un total de más de 2 millones de celdas. Aún así, el número de implicaciones generado no es excesivo debido a que el conjunto es altamente disperso.

En resumen, los números para este último *dataset* son:

- Filas (items): 17.400
- Columnas (atributos): 115
- Implicaciones: 675

[4] https://www.tripadvisor.es

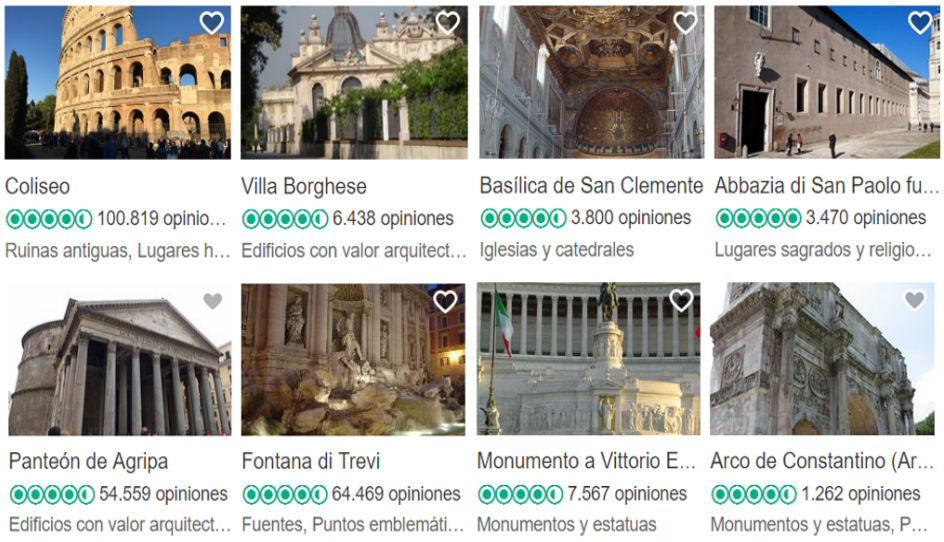

Figura 7.3: Ejemplo de puntos de interés de la ciudad de Roma que muestra el portal Tripadvisor. En la parte inferior de cada uno podemos ver a qué grupo pertenece (Iglesias, Monumentos, Estatuas, Fuentes, ...).

Pasamos ahora comprobar el resultado obtenido tras lanzar el test sobre el mayor de los *datasets* generados en esta investigación.

Recordemos ahora que estamos tratando con un *dataset* que contiene más de 17.000 puntos de interés que pueden verificar más de 100 atributos diferentes. Si bien los números de este *dataset* son mayores que los de el *dataset* MovieLens10M, vemos que los resultados obtenidos, *a priori* parecen aún más espectaculares (2 ó 3 pasos en la conversación). Sin embargo, en este caso tenemos que hacer unas aclaraciones.

Al principio de este apartado indicábamos que los tests se llevarían a cabo simulando diálogos donde los atributos se seleccionarían de forma aleatoria y analizamos los pros y contras de esta decisión. Bien, pues experimento pone en evidencia tales circunstancias. Por un lado, vemos como la mayoría de los test finalizan en 2 ó 3 pasos de conversación (ver Figura 7.4(a)). Esto puede considerarse un gran éxito del sistema al necesitar un diálogo tan corto a la hora de hacer una recomendación entre tantísimos elementos, y de hecho en muchos de los tests la situación se refleja fielmente.

Sin embargo, algunos otros experimentos que también han terminado tan rápido, se debe a que la selección de atributos no ha sido coherente. Por ejemplo, supongamos que entre todos los atributos se ha seleccionado que el punto de interés sea del tipo *Museo Histórico* y *Playas*, entonces a la hora de buscar elementos del *dataset* que verifiquen simultáneamente atributos tan dispares, el resultado contendrá muy pocos elementos y por tanto en pocos pasos habremos finalizado el diálogo. En definitiva, esto sucede debido a dos razones fundamentales:

(i) La selección aleatoria de atributos.

(ii) Y muy importante, el *dataset* es altamente disperso, ya que de entre los más de 100 atributos posibles, un punto de interés llega a verificar simultáneamente a la sumo 7 de ellos.

Por otro lado, en la Figura 7.4(b) y (c) podemos apreciar que en este caso la reducción de atributos y la velocidad de poda alcanzan niveles aún más notables que en el caso de MovieLens10M, y es que aunque los valores

sean similares (reducción entre el 5-25 % y velocidad de poda entre el 2-10 %), hay que tener en cuenta que debido al tamaño de este *dataset*, esos porcentajes implican un número de elementos eliminados muy superior.

En resumen, podemos constatar que incluso con *datasets* tan ricos en contenido, *sicumaRS* es capaz de entablar un diálogo con el usuario brindándole unas reducciones de información muy importantes, aliviando de esta forma de manera notable la sobrecarga de información.

7.4 Enfermedades y Síntomas *dataset*

Otro de los *datasets* generados es el que hemos denominado por cuenta propia *Diseases & Symptoms dataset*. Este *dataset* tiene una importancia mayor ya que ha sido el utilizado en una de las publicaciones que avalan esta Tesis Doctoral [14]. A continuación, pasamos a describir la fuente de los datos y la estructura del *dataset* generado.

La fuente original de la cual se ha extraído la información es el Phenotype Ontology Consortium[5] (HPO). Como podemos leer en su página web: "HPO [6] pretende proveer un lenguaje estandarizado de las fenotipos encontrados por anomalías en enfermedades humanas. Cada término en HPO describe una anomalía fenotípica. HPO está en desarrollo utilizando literatura médica, Orphanet[7], DECIPHER[8], y Online Mendelian Inheritance in Man (OMIM)[9].". A partir de toda esta información pudimos generar un *dataset* sobre el que desempeñar nuestros experimentos.

Debido a que la cantidad de información en el HPO es enorme, en esta primera aproximación sólo vamos a utilizar un extracto de toda la información disponible. Haciendo uso de OMIM para diferenciar entre diferentes tipos de fenotipos que aparecen en el HPO, hemos generado una tabla que empareja anomalías hematológicas y fenotipos, consiguiendo un conjunto

[5] http://www.human-phenotype-ontology.org
[6] HumanPhenotypeOntology
[7] http://www.orpha.net/consor/cgi-bin/index.php
[8] https://decipher.sanger.ac.uk
[9] http://www.omim.org

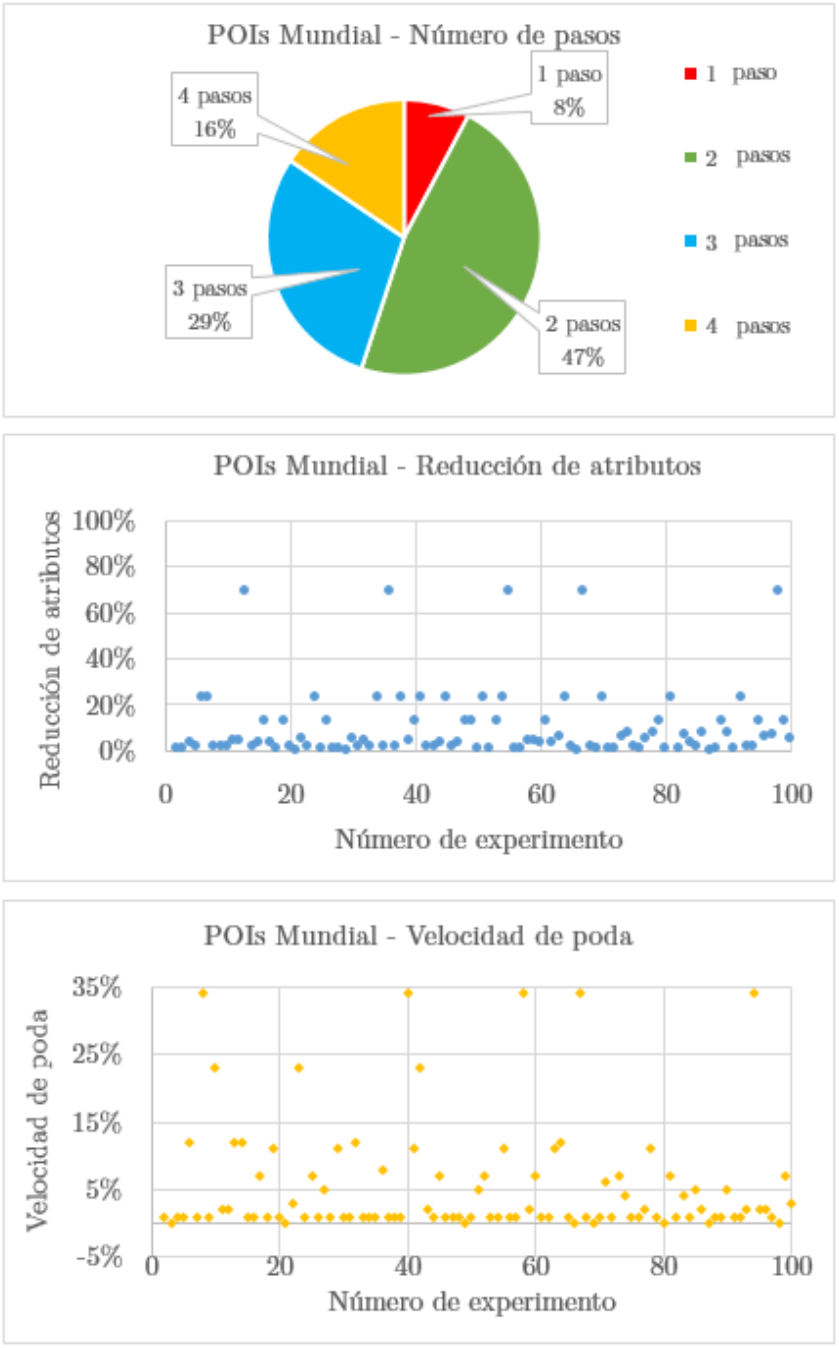

Figura 7.4: Resultado del experimento sobre el *dataset* POIs Mundial.

de datos considerable. Podemos ver un extracto de la información generada en la Tabla 7.5.

Cuadro 7.5: Diseases & Symptoms dataset (extract)

Disease ID	HPO_1249	HPO_1250	HPO_1251	HPO_1252	HPO_1254	...
274000		✓				
275630	✓		✓			
277380				✓	✓	
300884		✓		✓		
300322	✓			✓		
...						

En caso de que queramos obtener información detallada de cada anomalía que se muestra, encomendamos al lector a visitar la web de OMIM y utilizar su motor de búsqueda con el identificador mostrado en la columna *Disease ID*. Como ejemplo, la anomalía *Disease ID 275630* corresponde con el síndrome *Chanarin-Dorfman*.

En definitiva, vamos a trabajar sobre un *dataset* con 446 enfermedades, 100 fenotipos diferentes y todas las implicaciones que se verifican. Desgraciadamente, no podemos mostrarlas todas en el documento pues el conjunto alcanza un número superior a 6.000 implicaciones.

Este experimento además de su relevancia en cuanto a figurar en una de las publicaciones que avalan esta tesis [14], tiene una importancia añadida debido al alto número de implicaciones que contiene. Recordemos que en un *dataset* de referencia en el campo como lo es MovieLens10M 7.1 el número de implicaciones que obteníamos era 245, mientras que ahora *sicumaRS* trabajará y con resultados satisfactorios, con este *dataset* en el que el cardinal del conjunto de implicaciones es superior a 6.000 implicaciones.

A continuación recopilamos los números de este *dataset*:

- Filas (items): 446 anomalías.
- Columnas (atributos): 100 fenotipos.
- Implicaciones: 6.468

En este punto tenemos que hacer un inciso importante y que es de aplicación para todos los *datasets* y pruebas que se han realizado.

Nota 7.4.1. *Si nos centramos en el Enfermedades y Síntomas dataset y calculamos la llamada base Duquenne-Guigues [48] de implicaciones que se verifican en el contexto, el número de implicaciones asciende hasta las 8.811 implicaciones. Entonces, ¿por qué hablamos de que el sistema va a tratar unas 6.000 implicaciones con este dataset? Bien, pues la principal característica de la base Duquenne-Guigues de implicaciones es que esta base tiene el menor número de implicaciones posibles entre todas las posibles bases de implicaciones que se verifican en el contexto. Sin embargo, existirán algunas implicaciones para las que no haya ningún objeto que las verifique, y esas implicaciones significan que el conjunto de objetos que pueden cumplir con la premisa de la regla, no existen en el contexto. Además, esas implicaciones incluyen todos los atributos del contexto. Por tanto, el sentido de esas implicaciones es simplemente teórico y no presenta ningún valor cuando estamos tratando con aplicaciones reales; es por ello por lo que obviamos tales implicaciones.*

En este caso, los resultados del experimento y sus conclusiones pueden consultarse en [14].

Capítulo 8

Conclusiones y Trabajos Futuros

Fundamentalmente, podemos decir que esta Tesis ha englobado dos vertientes principales. Por un lado, se ha realizado un profundo estudio de los métodos basados en la lógica para el tratamiento eficiente de la información utilizando los conjuntos de implicaciones que se verifican en un determinado dataset. Y por otro lado, de forma más extensa, se han realizado las tareas de implementación necesarias para poder llevar estos métodos teóricos a la práctica. Hemos trabajado sobre tres campos diferentes: claves minimales, generadores minimales y sistemas de recomendación. Para cada uno de ellos se han realizado multitud de experimentos que demuestran la utilidad y la validez del trabajo realizado.

En esta tesis hemos podido apreciar el hecho de que contar con una sólida teoría basada en la lógica y las matemáticas nos concede la base para la creación de métodos automatizados con los que poder afrontar el desarrollo de aplicaciones de ingeniería. Como hemos podido comprobar, existe una gran cantidad de información implícita en los datos que solemos manejar. El descubrimiento de toda esta información y su gestión inteligente es sin duda un claro tema de investigación con fuerte actividad y repercusión en la actualidad. Esta ha sido nuestra intención a la hora de trabajar con FCA, los conjuntos de implicaciones y los operadores de cierre. Pasar de la teoría a la práctica y viceversa ha sido uno de los principales desafíos tanto

de esta tesis como lo es para el propio FCA si se pretende que se convierta en una herramienta fructífera para la representación, gestión y análisis del conocimiento en situaciones reales.

Antes de entrar plenamente en el apartado de conclusiones, queremos indicar la manera de certificar la validez de los resultados obtenidos a lo largo de la tesis. Como hemos podido advertir en los capítulos anteriores, nuestra labor se centra en actuar sobre conjuntos de implicaciones. En ese sentido, para los experimentos realizados hemos contado con unos ficheros de entrada que contenían la información necesaria, y sobre ellos hemos obtenido unos resultados. Ahora bien, la forma de verificar que esos resultados son correctos es la siguiente. En primer lugar y con respecto a los resultados de claves y generadores minimales, se han realizado y corregido numerosos ejercicios en papel intentando buscar casos límites donde la implementación pudiera no ser precisa y se ha comprobado que los resultados obtenidos en papel coincidían exactamente con los calculados por las implementaciones en la máquina. Además, dado que para muchos de los ejemplos probados en los que se llegaban a calcular millones de nodos de un árbol no era posible comprobar si cada uno de esos cálculos era correcto, para el caso concreto de los experimentos relacionados con claves minimales, la validez de los experimentos viene dada al haber cotejado los resultados con aquellos obtenidos sobre un amplio abanico de ficheros utilizados en trabajos anteriores [8, 9] donde su validez quedó demostrada. Básicamente, la validez de los ejercicios más grandes se ha extrapolado de los resultados correctos obtenidos para los ejercicios más pequeños. Asimismo, los resultados se corroboran igualmente al alcanzar las mismas soluciones para diferentes métodos cuando cada uno de ellos hace un tratamiento de la información diferente con respecto al otro. En relación a los experimentos con SR conversacionales, dado que los experimentos no alcanzan números tan altos, la validez puede demostrarse de forma más asequible siguiendo un desarrollo explícito en papel.

Y el otro punto que queremos remarcar es el siguiente. A lo largo de todo el proyecto siempre hemos hablado de trabajar sobre sistemas de implicaciones. No obstante, queda fuera del ámbito de esta tesis el procedimiento

mediante el cual se obtiene esos elementos para un sistema de datos. Para ello, encomendamos al avezado lector a visitar [57, 133, 134]. En nuestro caso, nuestro cometido comienza con el tratamiento de la información una vez de ha extraído el conjunto de implicaciones de un sistema concreto.

Aclarados estos puntos, nos encontramos ahora en el último capítulo de la Tesis, en el cual vamos a recopilar la conclusiones más importantes que hemos alcanzado como resultado del trabajo de investigación realizado. Seguidamente, cerrarán el capítulo una serie de tareas con las que continuar a partir de este punto y que se introducen como trabajos futuros.

Conclusiones

Como hemos mostrado anteriormente, conocer las claves es fundamental en cualquier modelo de datos. En este sentido, hemos presentado una serie de métodos que nos permiten averiguar el conjunto de claves a partir del conjunto de implicaciones y haciendo uso de métodos de razonamiento automatizado, en concreto, la SL_{FD}. Se ha investigado, pasando desde la teoría a la práctica, los diferentes métodos implementados haciendo uso del paradigma de Tableaux y se ha comprobado como, hasta donde sabemos, los resultados obtenidos superan las aproximaciones anteriores.

Por otro lado, enumerar todos los conjuntos cerrados y sus generadores minimales es un problema muy complejo pero esencial en varias áreas de conocimiento y una oportunidad para mostrar los beneficios de FCA cuando trabajamos para aplicaciones reales. Para abordar esta tarea, se han presentado dos métodos de poda para mejorar el rendimiento de la enumeración de los generadores minimales. Para ello se ha hecho un uso intensivo de la SL_{FD} sobre conjuntos de implicaciones. Finalmente, se han creado, analizado y probado enfoques diferentes (MinGen, MinGenPr, GenMinGen), mostrando claramente las mejoras alcanzadas por cada uno.

En ambas situaciones, es decir, tanto para claves minimales como para generadores minimales, se han desarrollado los códigos necesarios para poder actuar sobre grandes cantidades de información. No obstante, para resolver problemas reales donde la cantidad de información de entrada sea

considerable, es incuestionable la necesidad de unos recursos enormes tales como los que ha proporcionado el Centro de Supercomputación y Bioinnovación de la Universidad de Málaga; sin ellos habría sido inviable haber podido realizar gran parte de las pruebas. De acuerdo con esto, es absolutamente necesario que las implementaciones tengan en cuenta el correcto uso de recursos de memoria. Incluso para problemas pequeños, la cantidad de memoria que se puede necesitar puede dispararse escandalosamente.

Asimismo, el parecido y la cualidad de independencia que tiene cada nodo del árbol tanto para los métodos de claves minimales como para los de generadores minimales, ponen de manifiesto que la utilización de estrategias paralelas sea la mejor opción para abordar estos problemas. No obstante, el primer punto que queremos aclarar es que el concepto de paralelismo que hemos utilizado se refiere a un paralelismo de tipo *hardware*. Con esto nos referimos a que los beneficios que obtenemos del paralelismo se deben a la utilización de un conjunto de procesadores que se encargan de ir resolviendo cada uno de los subproblemas de forma simultánea. Por lo tanto, no estamos ante un caso de desarrollo de código paralelo desde una visión más purista, sino que es más acertado considerarlo como una aplicación basada en una estrategia *MapReduce* [33].

Para la mayoría de los casos, en relación a claves y generadores minimales, existe un serio inconveniente. En primera instancia es prácticamente imposible, por el momento, prever cuál va a ser la magnitud que va a alcanzar la resolución del problema a la vista de la información de entrada. Esto nos va a obligar a realizar una serie de experimentos previos de forma que podamos aproximar las necesidades que va a tener un determinado experimento.

Para cada una de las implementaciones realizadas, existía la necesidad de establecer criterios que nos permitieran evaluar el rendimiento de las pruebas de forma que pudiéramos comparar unos métodos con otros. En el caso de los experimentos relacionados con claves y generadores minimales, cuando nos planteamos la idea de la comparación de resultados, lo primero que se nos ocurrió fue la medición de los tiempos que necesitaba cada uno de los métodos para obtener los resultados. No obstante, advertimos como

este parámetro está íntimamente ligado a la arquitectura que estemos utilizando para ejecutar el experimento, lo cual hace que el resultado dependa en gran medida de los recursos que se están utilizando y no tanto de la calidad o eficiencia del propio algoritmo. En consecuencia, se oscurecía la utilidad teórica de los resultados obtenidos. Por tanto, se decidió contabilizar magnitud del árbol y la cantidad de resultados redundantes que se obtienen. De esta forma, si alguien hiciera otro método con un código en cualquier otro lenguaje o utilizase recursos *hardware* diferentes que desembocaran en una mejora del tiempo, siempre podríamos atenernos al tamaño del árbol pudiendo defender si realmente es una mejora en el método o en la ejecución debido a la arquitectura.

En cuanto a los SR conversacionales, también son varias las conclusiones a las que hemos llegado. La más importante es que, efectivamente, el tratamiento que realizamos de la información por medio de implicaciones y la SL_{FD} puede aplicarse con éxito en este campo de conocimiento. Esto ya quedaba patente, como hemos mencionado con anterioridad, por la existencia de multitud de trabajos en la literatura de SR que utilizan conceptos de FCA; nuestro trabajo viene a reforzar este hecho.

Concretamente, presentamos una novedosa aplicación del algoritmo de cierre Cls para afrontar el problema de la sobrecarga de la información que a menudo presentan los SR. Nuestra solución propone un proceso conversacional de selección de atributos por parte del usuario. Este trabajo combina características de sistemas basados en conteido con sistemas badados en conocimiento mediante una gestión inteligente de las implicaciones a través del cierre Cls. Nuestro sistema constituye un marco que puede integrarse en diversos datasets y los resultados siguen siendo admisibles. Esto es, ciertamente, una gran contribución de este trabajo, pues ofrece la posibilidad de aplicarse a aplicaciones en diferentes áreas.

Otra conclusión importante es que existen numerosas técnicas de recomendación. De esta forma, una tarea fundamental en el desarrollo o en el análisis de un SR será identificar qué técnica es la más adecuada para su funcionamiento según el contexto de uso esperado. Del mismo modo, es excepcionalmente difícil abarcar todos los aspectos que involucran a un SR

para evitar los diferentes problemas que pueden aparecer, por tanto, hay que tener presente con qué expectativas queremos que actúe. Además, hay que tener en cuenta que son muy pocos los SR que basan su funcionamiento en una única técnica de recomendación; lo habitual es que sean sistemas híbridos con la intención de poder beneficiarse de las ventajas que ofrecen unas estrategias de recomendación y otras. Nuestro caso en un claro ejemplo; el tipo de SR desarrollado mezcla las estrategias de recomendación basada en contenido, basada en conocimiento y conversacional.

Por otro lado, existen numerosas opciones a la hora de evaluar el funcionamiento de un SR. Esta es una conclusión razonable en tanto en cuanto el número de técnicas diferentes con las que trabajan los SR es igualmente alto. Es fundamental decidir qué métricas son oportunas de aplicar dependiendo del tipo de SR que queramos evaluar, pues evidentemente habrá casos en los que una métrica no tenga cabida para un tipo de SR determinado. Asimismo, es recomendable no pretender obtener valores óptimos en la mayoría de las métricas. Hay que ser realista y darse cuenta de que es extremadamente difícil cumplir plenamente con unas y otras simultáneamente. La clave está en decidir cuál es el cometido principal de nuestro sistema, aplicarle las métricas de evaluación apropiadas y sacar las conclusiones pertinentes dentro de ese marco de actuación.

Finalmente, queremos remarcar que el sistema desarrollado es capaz de ir más allá del ámbito académico o de investigación. Las pruebas realizadas sobre *datasets* reales demuestran la viabilidad y la utilidad que el sistema puede aportar en entornos empresariales. De esta forma, apostamos de nuevo por una eficaz transmisión de conocimiento que acerque la empresa a la academia.

Trabajos Futuros

Finalmente, nuestros resultados motivan una serie de direcciones importantes para futuras investigaciones.

Respecto a las aportaciones referentes al tema de claves y generadores minimales existen varios aspectos con los que continuar a partir de este

trabajo de investigación. Los iremos introduciendo uno a uno sin perjuicio de que el orden implique una mayor o menos importancia.

En primera instancia es prácticamente imposible prever cuál va a ser la magnitud que va a alcanzar la resolución del problema a la vista de la información de entrada. Esto va a constituir en la mayoría de los casos un serio inconveniente.

Tanto en claves minimales como en generadores minimales hemos podido ver que existe un elemento fundamental en el diseño de las implementaciones paralelas: el valor de corte o BOV. Recordamos que el BOV es el valor a partir del cual la ejecución secuencial del código paralelo termina y se forman los diferentes subproblemas que serán resueltos en paralelo. Nos encontramos entonces con que establecer un valor de corte adecuado se convierte en una tarea realmente compleja. En primera instancia, hay que tener en cuenta la forma de expansión que tiene el Tableaux del experimento que estemos realizando y esto sólo lo podemos averiguar empíricamente. En definitiva, hay que buscar estrategias para encontrar un valor de corte tal que el aprovechamiento de los recursos computacionales sea óptimo.

Es necesario avanzar en el diseño de un *benchmark* que tenga en cuenta los diferentes aspectos y la naturaleza de estos problemas y algoritmos de manera que podamos dirigir las búsquedas de forma más eficiente.

Hay que profundizar en el hecho de que aumentar el número de cores para la resolución de un problema no siempre redunda en una mejora del rendimiento. Hay casos en los que aumentar los recursos utilizados puede ser incluso contraproducente como hemos comentado anteriormente.

Nuestra intención futura es aplicar el cálculo de claves y generadores minimales sobre datos reales, en especial a casos donde la información de entrada sea considerable, de forma que podamos valorar el rendimiento alcanzado gracias al paralelismo.

Otra tarea sobre la que continuar el trabajo consiste en investigar cómo realizar un nuevo diseño de los códigos paralelos que nos permita establecer comunicación entre las diferentes resoluciones paralelas de un mismo problema de forma que podamos mejorar en la reducción de los cálculos redundantes. Para el caso concreto del cálculo de los generadores minimales,

este mismo objetivo nos serviría para obtener una versión paralela del mejor método secuencial estudiado (GenMinGen).

En relación a los SR conversacionales, aparecen dos aspectos interesantes a tener en cuenta en el futuro próximo. En primer lugar, sería muy valioso poder identificar desde un primer momento qué elementos del dataset presentan una importancia superior entre los demás, ya sea por aparecer con mayor frecuencia, ser único, tener relación con un mayor número de elementos del conjunto, etc. En segundo lugar y siguiendo la misma idea, sería sin duda crucial saber qué características del dataset (tamaño, escasez, etc.) son las más influyentes en el rumbo que toman las conversaciones con el usuario. Tener un mayor conocimiento de estos dos factores nos permitiría poder influir de manera positiva en la conversación del sistema con el usuario y de esta forma, deparar en una mejor experiencia de usuario.

Asimismo, hay aspectos de los SR que sería recomendable investigar si sería acertado incluir en el sistema conversacional; tal puede ser el caso de proporcionar explicaciones que justifiquen las recomendaciones que el usuario ha recibido. Este es un aspecto importante en un SR, ya que ayuda a mantener un mayor grado de confianza del usuario en los resultados generados por el sistema [110].

Índice de figuras

Índice de cuadros

Bibliografía

[1] Adomavicius, G., and Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng. 17*, 6 (June 2005), 734–749.

[2] Adomavicius, G., and Tuzhilin, A. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer, 2011, pp. 217–253.

[3] Armstrong, W. W. Dependency structures of data base relationships. In *IFIP Congress* (1974), pp. 580–583.

[4] Atencia, M., David, J., and Scharffe, F. Keys and pseudo-keys detection for web datasets cleansing and interlinking. In *Knowledge Engineering and Knowledge Management* (Berlin, Heidelberg, 2012), A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Acquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, Eds., Springer Berlin Heidelberg, pp. 144–153.

[5] Atzeni, P., and De Antonellis, V. *Relational Database Theory*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.

[6] Avazpour, I., Pitakrat, T., Grunske, L., and Grundy, J. *Dimensions and Metrics for Evaluating Recommendation Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 245–273.

[7] Beel, J., Langer, S., Nurnberger, A., and Genzmehr, M. The impact of demographics (age and gender) and other user-characteristics on evaluating recommender systems. In *TPDL* (2013), vol. 8092 of *Lecture Notes in Computer Science*, Springer, pp. 396–400.

[8] Benito-Picazo, F. *Minimal Key-Par. Una versión paralela de los algoritmos de búsqueda de claves minimales basados en Tableaux.* Dpto. Lenguajes y Ciencias de la Computación, Universidad de Málaga, 2013.

[9] Benito-Picazo, F. *Study of minimal keys algorithms based on tableaux methods. Increasing the range of treated problems by means of parallelism and prune strategies based on minimal subsets.* Languages and Computer Science Department, Universidad de Málaga, 2014.

[10] Benito-Picazo, F., Cordero, P., Enciso, M., and Mora, A. Increasing the efficiency of minimal key enumeration methods by means of parallelism. In *ICSOFT-EA 2014 - Proceedings of the 9th International Conference on Software Engineering and Applications, Vienna, Austria, 29-31 August, 2014* (2014), pp. 512–517.

[11] Benito-Picazo, F., Cordero, P., Enciso, M., and Mora, A. Keys for the fusion of heterogeneous information. In *Proceedings of the Fifteenth International Conference on Computational and Mathematical Methods in Science and Engineering* (Cádiz, Spain, 2015), pp. 201–211.

[12] Benito-Picazo, F., Cordero, P., Enciso, M., and Mora, A. Closed sets enumeration: a logical approach. In *Proceedings of the Seventeenth International Conference on Computational and Mathematical Methods in Science and Engineering* (Cádiz, Spain, 2017), pp. 287–292.

[13] Benito-Picazo, F., Cordero, P., Enciso, M., and Mora, A. Reducing the search space by closure and simplification paradigms. *Journal of Supercomputing 73*, 1 (Jan. 2017), 75–87.

[14] Benito-Picazo, F., Enciso, M., Rossi, C., and Guevara, A. Enhancing the conversational process by using a logical closure operator in phenotypes implications. *Mathematical Methods in the Applied Sciences* (2017).

[15] Bertet, K., Demko, C., Viaud, J.-F., and Guérin, C. Lattices, closures systems and implication bases: A survey of structural aspects and algorithms. *Theoretical Computer Science* (2016).

[16] Bobadilla, J., Hernando, A., Ortega, F., and Bernal, J. A framework for collaborative filtering recommender systems. *Expert Syst. Appl. 38*, 12 (Nov. 2011), 14609–14623.

[17] Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. Recommender systems survey. *Knowledge-Based Systems 46*, 0 (2013), 109 – 132.

[18] Bonifacio, C., Barchyn, T. E., Hugenholtz, C. H., and Kienzle, S. W. Ccdst: A free canadian climate data scraping tool. *Computers and Geosciences 75* (2015), 13 – 16.

[19] Borras, J., de la Flor, J., Pérez, Y., Moreno, A., Valls, A., Isern, D., Orellana, A., Russo, A., and Clavé, S. A. Sigtur/e-destination: A system for the management of complex tourist regions. R. Law, M. Fuchs, and F. Ricci, Eds., Springer Vienna, pp. 39–50.

[20] Burke, R. Evaluating the dynamic properties of recommendation algorithms. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (New York, NY, USA, 2010), RecSys '10, ACM, pp. 225–228.

[21] Chen, L., and Pu, P. Hybrid Critiquing-based Recommender Systems. In *Proceedings of the 12th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2007), IUI '07, ACM, pp. 22–31.

[22] Chen, L., and Pu, P. Critiquing-based recommenders: Survey and emerging trends. *User Modeling and User-Adapted Interaction 22*, 1-2 (Apr. 2012), 125–150.

[23] Codd, E. F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (1970), 377–387.

[24] Codocedo, V., and Napoli, A. Formal concept analysis and information retrieval – a survey. In *Formal Concept Analysis* (Cham, 2015), J. Baixeries, C. Sacarea, and M. Ojeda-Aciego, Eds., Springer International Publishing, pp. 61–77.

[25] Cohen, E., Datar, M., Fujiwara, S., Gionis, A., Indyk, P., Motwani, R., Ullman, J. D., and Yang, C. Finding interesting associations without support pruning. *IEEE Trans. on Knowl. and Data Eng. 13*, 1 (Jan. 2001), 64–78.

[26] Cordero, P., Enciso, M., and Mora, A. Automated reasoning to infer all minimal keys. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (2013), IJCAI'13, AAAI Press, pp. 817–823.

[27] Cordero, P., Enciso, M., Mora, A., and de Guzmán, I. P. SLFD logic: Elimination of data redundancy in knowledge representation. In *IBERAMIA 2002: Proceedings of the 8th Ibero-American Conference on AI* (London, UK, 2002), Springer-Verlag, pp. 141–150.

[28] Cordero, P., Enciso, M., Mora, A., and de Guzmán, I. P. A tableaux-like method to infer all minimal keys. *Logic Journal of the IGPL 22*, 6 (2014), 1019–1044.

[29] Cordero, P., Enciso, M., Mora, A., and Ojeda-Aciego, M. Computing minimal generators from implications: a logic-guided approach. In *Proceedings of The Ninth International Conference on Concept Lattices and Their Applications, Fuengirola (Málaga), Spain, October 11-14, 2012* (2012), pp. 187–198.

[30] Cordero, P., Enciso, M., Mora, A., Ojeda-Aciego, M., and Rossi, C. Knowledge discovery in social networks by using a logic-based treatment of implications. *Knowledge-Based System 87* (2015), 16–25.

[31] Crespo, R. G., Martínez, O. S., Lovelle, J. M. C., García-Bustelo, B. C. P., Gayo, J. E. L., and de Pablos, P. O. Recommendation system based on user interaction data applied to intelligent electronic books. *Computers in Human Behavior 27*, 4 (2011), 1445 – 1449. Social and Humanistic Computing for the Knowledge Society.

[32] de Campos, L. M., Fernández-Luna, J. M., Huete, J. F., and Rueda-Morales, M. A. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning 51*, 7 (2010), 785 – 799.

[33] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation* (2004), USENIX Association.

[34] du Boucher-Ryan, P., and Bridge, D. Collaborative recommending using formal concept analysis. *Knowledge-Based Systems 19*, 5 (2006), 309 – 315. {AI} 2005 {SI}.

[35] Eirinaki, M., Gao, J., Varlamis, I., and Tserpes, K. Recommender systems for large-scale social networks: A review of challenges

and solutions. *Future Generation Computer Systems 78* (2018), 413 – 418.

[36] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering 19*, 1 (Jan 2007), 1–16.

[37] Elmasri, R., and Navathe, S. *Fundamentals of Database Systems*, 6 ed. Prentice Hall International, 2010.

[38] Endres, D., Adam, R., Giese, M. A., and Noppeney, U. Understanding the semantic structure of human fmri brain recordings with formal concept analysis. In *Proceedings of the 10th International Conference on Formal Concept Analysis* (Berlin, Heidelberg, 2012), ICFCA'12, Springer-Verlag, pp. 96–111.

[39] Fadous, R., and Forsyth, J. Finding candidate keys for relational data bases. In *SIGMOD Conference* (1975), W. F. King, Ed., ACM, pp. 203–210.

[40] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. From data mining to knowledge discovery in databases. *AI Magazine* (1996), 37–54.

[41] Feil, S., Kretzer, M., Werder, K., and Maedche, A. Using gamification to tackle the cold-start problem in recommender systems. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion* (New York, NY, USA, 2016), CSCW '16 Companion, ACM, pp. 253–256.

[42] Friedman, A., Knijnenburg, B. P., Vanhecke, K., Martens, L., and Berkovsky, S. *Privacy Aspects of Recommender Systems*. Springer US, Boston, MA, 2015, pp. 649–688.

[43] Ganter, B., and Wille, R. *Formal Concept Analysis: Mathematical Foundations*, 1st ed. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[44] Giannella, C., and Wyss, C. Finding minimal keys in a relation instance, 1999.

[45] Goh, T. N. An information management paradigm for statistical experiments. *Quality and Reliability Eng. Int. 26*, 5 (2010), 487–494.

[46] Gras, B., Brun, A., and Boyer, A. Identifying grey sheep users in collaborative filtering: A distribution-based technique. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization* (New York, NY, USA, 2016), UMAP '16, ACM, pp. 17–26.

[47] Griol, D., and Molina, J. M. Building multi-domain conversational systems from single domain resources. *Neurocomputing 271* (2018), 59 – 69.

[48] Guigues, J. L., and Duquenne, V. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines 95* (1986), 5–18.

[49] Gunawardana, A., and Shani, G. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res. 10* (Dec. 2009), 2935–2962.

[50] Gunawardana, A., and Shani, G. *Evaluating Recommender Systems*. Springer US, Boston, MA, 2015, pp. 265–308.

[51] Guo, G. Resolving data sparsity and cold start in recommender systems. In *Proceedings of the 20th International Conference on User Modeling, Adaptation, and Personalization* (Berlin, Heidelberg, 2012), UMAP'12, Springer-Verlag, pp. 361–364.

[52] Hamrouni, T., Valtchev, P., Yahia, S. B., and Nguifo, E. M. About the Lossless Reduction of the Minimal Generator. *Lecture Notes in Computer Science 4390* (2007), 130–150.

[53] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst. 5*, 4 (Jan. 2016), 19:1–19:19.

[54] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst. 22*, 1 (Jan. 2004), 5–53.

[55] Hernández del Olmo, F., and Gaudioso, E. Evaluation of recommender systems: A new approach. *Expert Syst. Appl. 35*, 3 (Oct. 2008), 790–804.

[56] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1995), CHI '95, ACM Press/Addison-Wesley Publishing Co., pp. 194–201.

[57] Huhtala, Y., Karkkainen, J., Porkka, P., and Toivonen, H. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J. 42*, 2 (1999), 100–111.

[58] Ibaraki, T., Kogan, A., and Makino, K. Functional dependencies in horn theories. *Artificial Intelligence 108*, 1 (1999), 1 – 30.

[59] Ignatov, D. I. Introduction to formal concept analysis and its applications in information retrieval and related fields. In *Information Retrieval* (Cham, 2015), P. Braslavski, N. Karpov, M. Worring, Y. Volkovich, and D. I. Ignatov, Eds., Springer International Publishing, pp. 42–141.

[60] Isinkaye, F., Folajimi, Y., and Ojokoh, B. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal 16*, 3 (2015), 261 – 273.

[61] Jannach, D., Zanker, M., and Fuchs, M. Constraint-Based Recommendation in Tourism: A Multiperspective Case Study. *Information Technology & Tourism 11*, 2 (2009), 139–155.

[62] Johansson, L., Epitropou, V., Karatzas, K., Karppinen, A., Wanner, L., Vrochidis, S., Bassoukos, A., Kukkonen, J., and

Kompatsiaris, I. Fusion of meteorological and air quality data extracted from the web for personalized environmental information services. *Environmental Modelling and Software 64* (2015), 143 – 155.

[63] Karnaugh, M. The map method for synthesis of combinational logic circuits. *American Institute of Electrical Engineers, Part I: Communication and Electronics, Transactions of the 72*, 5 (Nov. 1953), 593–599.

[64] Kaytoue, M., Kuznetsov, S. O., Napoli, A., and Duplessis, S. Mining gene expression data with pattern structures in formal concept analysis. *Information Sciences 181*, 10 (2011), 1989 – 2001. Special Issue on Information Engineering Applications Based on Lattices.

[65] Kemper, A., and Moerkotte, G. Query optimization in object bases: Exploiting relational techniques. In *Query Processing for Advanced Database Systems, Dagstuhl*. Morgan Kaufmann, 1991, pp. 63–98.

[66] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM 40*, 3 (Mar. 1997), 77–87.

[67] Krajca, P., Outrata, J., and Vychodil, V. Parallel recursive algorithm for fca. In *6th Int. Conf. on Concept Lattices and Their Applications (CLA)* (2008), vol. 433, CEUR WS, pp. 71–82. Olomouc (ZC).

[68] Kuznetsov, S., and Obiedkov, S. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence 14* (2002), 189–216.

[69] Lampropoulos, A. S., Lampropoulou, P. S., and Tsihrintzis, G. A. A cascade-hybrid music recommender system for mobile ser-

vices based on musical genre classification and personality diagnosis. *Multimedia Tools Appl. 59*, 1 (2012), 241–258.

[70] Lee, S., and Choi, J. Enhancing user experience with conversational agent for movie recommendation: Effects of self-disclosure and reciprocity. *International Journal of Human-Computer Studies 103* (2017), 95 – 105.

[71] Leiva, J. L., Enciso, M., Rossi, C., Cordero, P., Mora, A., and Guevara, A. Context-aware recommendation using fuzzy formal concept analysis. In *ICSOFT* (2013), J. Cordeiro, D. A. Marca, and M. van Sinderen, Eds., SciTePress, pp. 617–623.

[72] Leiva, J. L., Enciso, M., Rossi, C., Cordero, P., Mora, A., and Guevara, A. Improving recommender systems with simplification logic to manage implications with grades. In *Software Technologies - 8th International Joint Conference, ICSOFT 2013, Reykjavik, Iceland, July 29-31, 2013, Revised Selected Papers* (2013), pp. 290–305.

[73] Lèvy, G., and Baklouti, F. A distributed version version of the ganter algorithm for general galois lattices. In *3rd. Int. Conf. on Concept Lattices and Their Applications (CLA)* (2005), vol. 162, CEUR WS, pp. 207–221. Olomouc (ZC).

[74] Linden, G., Smith, B., and York, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing 7* (2003), 76–80.

[75] Lops, P., de Gemmis, M., and Semeraro, G. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer, 2011, pp. 73–105.

[76] Lucchesi, C. L., and Osborn, S. L. Candidate keys for relations. *J. Comput. Syst. Sci. 17*, 2 (1978), 270–279.

[77] Maier, D. *The Theory of Relational Databases*. Computer Science Press, Rockville, 1983.

[78] Maier, D. *Theory of Relational Databases*. Computer Science Pr, 1983.

[79] Mandl, M., Felfernig, A., Teppan, E., and Schubert, M. Consumer decision making in knowledge-based recommendation. *Journal of Intelligent Information Systems 37* (2011), 1–22.

[80] Manolopoulos, Y., Theodoridis, Y., and Tsotras, V. J. *Advanced Database Indexing*, vol. 17 of *Advances in Database Systems*. Kluwer, 1999.

[81] McSherry, D. Minimizing dialog length in interactive case-based reasoning. In *Procs of the 17th Int Joint Conf on AI, IJCAI, Seattle, Washington, USA, August 4-10, 2001* (2001), pp. 993–998.

[82] Mimouni, N., Nazarenko, A., and Salotti, S. *A Conceptual Approach for Relational IR: Application to Legal Collections*. Springer International Publishing, Cham, 2015, pp. 303–318.

[83] Missaoui, R., Nourine, L., and Renaud, Y. An inference system for exhaustive generation of mixed and purely negative implications from purely positive ones. In *Proceedings of the 7th International Conference on Concept Lattices and Their Applications, Sevilla, Spain, October 19-21, 2010* (2010), pp. 271–282.

[84] Missaoui, R., Nourine, L., and Renaud, Y. Computing implications with negation from a formal context. *Fundam. Inf. 115*, 4 (Dec. 2012), 357–375.

[85] Mora, A., Cordero, P., Enciso, M., Fortes, I., and Aguilera, G. Closure via functional dependence simplification. *Int. J. Comput. Math. 89*, 4 (2012), 510–526.

[86] Mora, Á., Enciso, M., Cordero, P., and Pérez de Guzmán, I. *An Efficient Preprocessing Transformation for Functional Dependencies Sets Based on the Substitution Paradigm*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 136–146.

[87] Morgan, C. G. An automated theorem prover for relational logic (abstract). In *Workshop Theorem Proving with Analytic Tableaux and Related Methods, Lautenbach. Universität Karlsruhe, Fakultät für Informatik, Institut für Logik, Komplexität und Deduktionssysteme, Interner Bericht 8/92, March 18-20, 1992* (1992), pp. 56–58.

[88] Motameny, S., Versmold, B., and Schmutzler, R. Formal concept analysis for the identification of combinatorial biomarkers in breast cancer. In *Proceedings of the 6th International Conference on Formal Concept Analysis* (Berlin, Heidelberg, 2008), ICFCA'08, Springer-Verlag, pp. 229–240.

[89] Nagler, T., and Czado, C. Evading the curse of dimensionality in nonparametric density estimation with simplified vine copulas. *Journal of Multivariate Analysis 151* (2016), 69 – 89.

[90] Niemelä, I. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25*, 3-4 (Feb. 1999), 241–273.

[91] Nishio, N., Mutoh, A., and Inuzuka, N. On computing minimal generators in multi-relational data mining with respect to 0-subsumption. *CEUR Workshop Proceedings 975* (2012), 50–55.

[92] Ore, O. Galois connections. *Transactions of the American Mathematical Society 55* (1944), 493–513.

[93] Paredaens, J., Bra, P., Gyssens, M., and Gucht, D. V., Eds. *The structure of the relational database model*. EATCS Monographs on Theoretical Computer Science, 1989.

[94] Paredaens, J., Bra, P., Gyssens, M., and Gucht, D. V., Eds. *The structure of the relational database model.* EATCS Monographs on Theoretical Computer Science, 1989.

[95] Pernelle, N., Saïs, F., and Symeonidou, D. An automatic key discovery approach for data linking. *Web Semantics: Science, Services and Agents on the WWW 23* (2013), 16–30.

[96] Poelmans, J., Ignatov, D. I., Kuznetsov, S. O., and Dedene, G. Formal concept analysis in knowledge processing: A survey on applications. *Expert Systems with Applications 40*, 16 (2013), 6538 – 6560.

[97] Porcel, C., Tejeda-Lorente, A., Martínez, M. A., and Herrera-Viedma, E. A hybrid recommender system for the selective dissemination of research resources in a technology transfer office. *Inf. Sci. 184*, 1 (Feb. 2012), 1–19.

[98] Priss, U. Formal concept analysis in information science. *Annual Rev. Info. Sci & Technol. 40*, 1 (Dec. 2006), 521–543.

[99] Qu, K., Zhai, Y., Liang, J., and Chen, M. Study of decision implications based on formal concept analysis. *International Journal of General Systems 36*, 2 (2007), 147–156.

[100] Reda, A., Park, Y., Tiwari, M., Posse, C., and Shah, S. Metaphor: a system for related search recommendations. In *CIKM* (2012), X. wen Chen, G. Lebanon, H. Wang, and M. J. Zaki, Eds., ACM, pp. 664–673.

[101] Resnick, P., and Varian, H. R. Recommender systems. *Commun. ACM 40*, 3 (Mar. 1997), 56–58.

[102] Risch, V., and Schwind, C. Tableaux-based theorem proving and non-standard reasoning. In *Workshop Theorem Proving with Analytic Tableaux and Related Methods, Lautenbach. Universität Karlsruhe,*

Fakultät für Informatik, Institut für Logik, Komplexität und Deduktionssysteme, Interner Bericht 8/92, March 18-20, 1992 (1992), pp. 76–78.

[103] Saiedian, H., and Spencer, T. An efficient algorithm to compute the candidate keys of a relational database schema. *Comput. J. 39*, 2 (1996), 124–132.

[104] Sali, A. *Minimal Keys in Higher-Order Datamodels*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 242–251.

[105] Salimi, A., Ziaii, M., Amiri, A., Zadeh, M. H., Karimpouli, S., and Moradkhani, M. Using a feature subset selection method and support vector machine to address curse of dimensionality and redundancy in hyperion hyperspectral data classification. *The Egyptian Journal of Remote Sensing and Space Science* (2017).

[106] Sassi, I. B., Mellouli, S., and Yahia, S. B. Context-aware recommender systems in mobile environment: On the road of future research. *Information Systems 72* (2017), 27 – 61.

[107] Senatore, S., and Pasi, G. Lattice navigation for collaborative filtering by means of (fuzzy) formal concept analysis. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2013), SAC '13, ACM, pp. 920–926.

[108] Shah Khusro, Zafar Ali, I. U. Recommender systems: Issues, challenges, and research opportunities. In *Information Science and Applications (ICISA) 2016* (2016), vol. 376, IEEE, pp. 1179–1189.

[109] Shani, G., and Gunawardana, A. *Evaluating Recommendation Systems*. Springer US, Boston, MA, 2011, pp. 257–297.

[110] Sharma, R., and Ray, S. Explanations in recommender systems: An overview. *Int. J. Bus. Inf. Syst. 23*, 2 (Jan. 2016), 248–262.

[111] Shirude, S. B., and Kolhe, S. R. *Machine Learning Using K-Nearest Neighbor for Library Resources Classification in Agent-Based Library Recommender System.* Springer Singapore, Singapore, 2016, pp. 17–29.

[112] Simsion, G. C., and Witt, G. C. *Data modeling essentials*, 3rd ed. Amsterdam; Boston, 2005.

[113] Sismanis, Y., Brown, P., Haas, P. J., and Reinwald, B. Gordian: efficient and scalable discovery of composite keys. In *In Proc. International Conference on Very Large Data Bases (VLDB* (2006), pp. 691–702.

[114] Snelting, G., and Tip, F. Reengineering class hierarchies using concept analysis. *SIGSOFT Softw. Eng. Notes 23*, 6 (Nov. 1998), 99–110.

[115] Son, J., and Kim, S. B. Academic paper recommender system using multilevel simultaneous citation networks. *Decision Support Systems 105* (2018), 24 – 33.

[116] Son, L. H. Dealing with the new user cold-start problem in recommender systems: A comparative review. *Information Systems 58* (2016), 87 – 104.

[117] Sridhar, R., and Iyengar, S. S. Efficient parallel algorithms for functional dependency manipulations. In *Procs of the 2nd Int. Symposium on Databases in Parallel and Distributed Systems* (1990), DPDS '90, pp. 126–137.

[118] Stumme, G. *Efficient Data Mining Based on Formal Concept Analysis.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 534–546.

[119] Sun, X. *Construction Data Mining Information Management System Based on FCA and Ontology.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 19–24.

[120] Sundaresan, N. Recommender systems at the long tail. In *Proceedings of the Fifth ACM Conference on Recommender Systems* (New York, NY, USA, 2011), RecSys '11, ACM, pp. 1–6.

[121] Tiroshi, A., Kuflik, T., Kay, J., and Kummerfeld, B. Recommender systems and the social web. In *UMAP Workshops* (2011), L. Ardissono and T. Kuflik, Eds., vol. 7138 of *Lecture Notes in Computer Science*, Springer, pp. 60–70.

[122] Trabelsi, W., Wilson, N., Bridge, D. G., and Ricci, F. Preference dominance reasoning for conversational recommender systems: a comparison between a comparative preferences and a sum of weights approach. *International Journal on Artificial Intelligence Tools 20*, 4 (2011), 591–616.

[123] Valtchev, P., and Duquenne, V. Towards scalable divide-and-conquer methods for computing concepts and implications. In *4th Int. Conf. Journèes de l'Informatique Messine (JIM03): Knowledge Discovery and Discrete Mathematics. Metz (FR)* (2003), pp. 3–14.

[124] Valtchev, P., and Duquenne, V. On the merge of factor canonical bases. In *International Conference on Formal Concept Analysis (ICFCA)* (2008), vol. 4933 of *Lecture Notes in Computer Science*, Springer, pp. 182–198.

[125] Wastl, R. Linear derivations for keys of a database relation schema. *Journal of Universal Computer Science 4*, 11 (1998), 883–897.

[126] Wastl, R. On the number of keys of a relational database schema. *Journal of Universal Computer Science 4*, 5 (1998), 547–559.

[127] Wille, R. *Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts.* Springer Netherlands, Dordrecht, 1982, pp. 445–470.

[128] Wille, R. *Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 1–33.

[129] Wille, R. *Formal Concept Analysis as Applied Lattice Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 42–67.

[130] Witten, I., Frank, E., and Hall, M. *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.

[131] Worland, P. B. An efficient algorithm for 3nf determination. *Information Sciences 167*, 1 (2004), 177 – 192.

[132] Yang, Z., and Cai, Z. Detecting abnormal profiles in collaborative filtering recommender systems. *Journal of Intelligent Information Systems* (2016), 1–20.

[133] Yao, H., Hamilton, H. J., and Butz, C. J. Fd mine: Discovering functional dependencies in a database using equivalences. In *ICDM* (2002), IEEE Computer Society, pp. 729–732.

[134] Yevtushenko, S., Tane, J., Kaiser, T. B., Obiedkov, S., Hereth, J., and Reppe, H. Conexp - the concept explorer.

[135] Yu, C. T., and Johnson, D. T. On the complexity of finding the set of candidate keys for a given set of functional dependencies. *Inf. Process. Lett. 5*, 4 (1976), 100–101.

[136] Yu, K. A Scalable and Accurate Online Feature Selection for Big Data.

[137] Zhang, F., Gong, T., Lee, V. E., Zhao, G., Rong, C., and Qu, G. Fast algorithms to evaluate collaborative filtering recommender systems. *Knowledge-Based Systems 96* (2016), 96 – 103.

[138] Zhang, W., Du, Y. J., and Song, W. Recommender system with formal concept analysis. In *2015 International Conference on Information and Communications Technologies (ICT 2015)* (April 2015), pp. 1–6.

[139] Zhang, X.-L., Lee, T. M. D., and Pitsilis, G. Securing recommender systems against shilling attacks using social-based clustering. *Journal of Computer Science and Technology 28*, 4 (2013), 616–624.

[140] Zhang, Y. Determining all candidate keys based on karnaugh map. *IEEE International Conference on Information Management, Innovation Management and Industrial Engineering 04* (2009), 226–229.

[141] Zhou, W., Wen, J., Gao, M., Liu, L., Cai, H., and Wang, X. *A Shilling Attack Detection Method Based on SVM and Target Item Analysis in Collaborative Filtering Recommender Systems.* Springer International Publishing, Cham, 2015, pp. 751–763.

[142] Zobel, J. How reliable are the results of large-scale information retrieval experiments? In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 1998), SIGIR '98, ACM, pp. 307–314.

[143] Zou, C., Zhang, D., Wan, J., Hassan, M. M., and Lloret, J. Using concept lattice for personalized recommendation system design. *IEEE Systems Journal 11*, 1 (March 2017), 305–314.