Universidad del Bío-Bío

Facultad de Ciencias Empresariales

Departamento de Ciencias de la Computación y
Tecnologías de la Información

Método para la Obtención de Dependencias Funcionales Basado en Dualidad de Hipergrafos

Por

Joel Fuentes López

Tesis para optar al grado de Magister
en Ciencias de la Computación

Profesor director : **Pablo Sáez**
Profesor co-director : **Gilberto Gutiérrez**

Chillán - Chile

Enero de 2012

A mis padres y mi novia, que sin esperar nada a cambio,
han sido pilares en mi camino y así, forman parte de este logro
que espero abra nuevos senderos a mi desarrollo profesional.

Agradecimientos

A Dios, por demostrame tantas veces su existencia y con ello darme fuerzas para cumplir este anhelo.

A mis padres, Sebastián y Carmen. Con su fuerza y sacrificio me inspiran cada día a dar lo mejor de mí.

A mi novia Lidia, gracias por todo el apoyo y por estar siempre ahí.

A mis directores de tesis el Dr. Pablo Sáez y el Dr. Gilberto Gutiérrez, por su constante apoyo, comprensión y disposición para este trabajo de investigación. Destaco de ambos su gran conocimiento en la Ciencia de la Computación, pero también su inmensa calidad humana.

Finalmente, a la Universidad del Bío-Bío a través del programa de Magíster en Ciencias de la Computación, por permitirme continuar mi desarrollo como profesional.

A todos ellos, infinitas gracias.

Resumen

Una de las principales etapas en las distintas metodologías enfocadas en la obtención del modelo relacional a partir de sistemas heredados es la extracción de las dependencias funcionales por medio de técnicas de minería de datos. Resulta importante para los diversos enfoques de migración propuestos, realizar la labor de extracción de dependencias funcionales desde el conjunto de datos del sistema heredado. Para esto, el hecho de contar con una herramienta que realice esta labor de forma automática pasa a ser un factor fundamental. Es así como en el último tiempo se han propuesto variados métodos para este fin, siendo las soluciones más conocidas exponenciales en tiempo en el número de atributos de la relación, considerando que en situaciones reales es frecuente encontrar base de datos en que la cantidad de atributos de sus relaciones es alta (más de 20 ó 30 atributos).

La presente tesis está enfocada a resolver este problema utilizando el conocimiento del que se dispone en la actualidad en relación con el problema de dualidad de hipergrafos, para el cual se conocen algoritmos cuasi-polinomiales ($O(n^{\log n})$), que hasta ahora no han sido comúnmente considerados para resolver el problema de la obtención de dependencias funcionales. Se puede mostrar que este problema, si se parte de las refutaciones para estas dependencias existentes en los conjuntos de datos de los sistemas heredados, es equivalente al mencionado problema de dualidad de hipergrafos.

En concreto, y luego del estudio los algoritmos, tanto para la obtención de dependencias funcionales como de teoría de hipergrafos, se logró construir una herramienta de la cual se

pueden beneficiar los procesos de migración de los sistemas heredados. Experimentaciones muestran que esta herramienta responde bien cuando la cantidad de atributos de la relación es grande, que es el caso en el cual muestran debilidades los enfoques comúnmente usados en la actualidad.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Capítulo 1

Introducción

1.1 Introducción General

Las dependencias funcionales (DF de aquí en adelante) son restricciones de integridad sobre los datos. Conocer las DF en el momento del diseño de base de datos permite crear mecanismos para evitar la redundancia, con los potenciales problemas de integridad que ella conlleva, además de permitir mejorar la eficiencia de los programas.

El descubrimiento de DFs en una instancia de una relación es una importante técnica en la minería de datos. Las DFs descubiertas son utilizadas en la administración de bases de datos [RG00], diseño de base de datos y optimización de consultas [CDGL01], reparación de archivos XML [FFGZ03], ingeniería inversa [TZ04], entre otros. También cumplen una labor fundamental en el trabajo con sistemas de información heredados, y es en este último ámbito donde se enfoca este trabajo de investigación, con el objetivo de proponer una herramienta capaz de extraer las DFs de forma automática desde estos sistemas. Por ejemplo, resulta útil descubrir las DFs de un sistema heredado cuyo conjunto de datos contiene componentes químicos, lo que resulta valioso para descubrir atributos de los componentes que son dependientes funcionalmente en una cierta estructura química [HKPT99].

Estudios como [Lin08, VCG10, And94] proponen métodos y estrategias para obtener el modelo de base de datos relacional desde sistemas heredados, destacando enfoques basados en ingeniería inversa y de recuperación de la arquitectura de los sistemas. En estos estudios destaca como fundamental la etapa de la obtención automática de las DFs, lo que resalta la importancia de contar con herramientas eficientes para este fin.

Actualmente existen algoritmos y herramientas que pueden realizar esta labor, pero las soluciones más difundidas requieren un tiempo exponencial en el número de atributos de la relación. Es por ello que nace la inquietud de intentar mejorar las técnicas hasta ahora utilizadas por estos algoritmos. Para esto se mostrará en las siguientes secciones que el problema de minería de DFs se puede llevar eficientemente al de dualidad de hipergrafos, para el cual se conocen algoritmos de complejidad $O(n^{\log n})$, es decir sub exponenciales (o cuasi-polinomiales).

En esta tesis se presentan los resultados de la investigación y experimentación sobre el método propuesto para la obtención de DFs, trabajo que también fue presentado por los autores en los proceedings de las XVI jornadas del software y las bases de datos, JISBD, celebradas en La Coruña, España, del 5 al 7 de septiembre del 2011 [FSG11].

1.2 Hipótesis

Los nuevos algoritmos para resolver el problema de dualidad de hipergrados permiten mejorar los tiempos de respuesta en la obtención de dependencias funcionales desde una instancia de una relación cuando el número de atributos es mayor a 20.

1.3 Objetivos

Objetivo General

Proponer un método para la obtención de dependencias funcionales desde sistemas de información heredados utilizando la teoría de hipergrafos. Con este método se espera obtener mejor rendimiento que los algoritmos conocidos cuando el conjunto de atributos R es grande, considerando que en el estudio de dualidad de hipergrafos se han propuesto variados algoritmos.

Objetivos Específicos

Para lograr el objetivo general se consideran los siguientes objetivos específicos:

- Analizar y evaluar algoritmos existentes para la generación de DFs.
- Analizar e implementar los principales algoritmos de Dualidad de Hipergrafos, tales como [MI98, BEGK00, FK96, KS05].
- Construir, en base al método ideado, una herramienta de software que sea capaz de extraer de forma automática todas las DFs existentes en una instancia de una relación.
- Realizar pruebas de obtención de DFs desde instancias de relaciones para medir la eficiencia de la herramienta construida.

1.4 Organización

Esta tesis contempla 5 capítulos. El primer Capítulo es el actual y corresponde a la introducción, objetivos y organización de la presente investigación.

El Capítulo 2 se centra en describir los trabajos relacionados, especificando en primera instancia las definiciones preliminares, luego el nexo de la obtención de DFs con los sistemas heredados y finalmente se describen los algoritmos existentes para la obtención de dependencias funcionales desde una instancia de una relación.

El Capítulo 3 está destinado a describir la teoría de hipergrafos y principalmente la dualización de hipergrafos. En primer lugar se especifican las definiciones y operadores sobre hipergrafos y luego se aborda el problema de dualidad de hipergrafos, donde además se exponen los principales algoritmos que se utilizarán para la implementación del enfoque propuesto.

En el Capítulo 4 se detalla el método que se propone para resolver el problema de obtención de DFs. Se consideran aspectos teóricos de la propuesta, como también aspectos técnicos. En la parte final del capítulo se exponen las pruebas y resultados experimentales realizados con la implementación del método, considerando los datos de prueba utilizados, el ambiente de prueba y por los resultados finales obtenidos.

Finalmente, en el último capítulo se exponen las conclusiones de esta tesis y se definen algunos posibles trabajos futuros a realizar.

Capítulo 2

Trabajos relacionados

Se presenta aquí parte importante del resultado del estudio del estado del arte. En este capítulo se especifican las definiciones, que serán fundamentales para comprender secciones posteriores, la problemática de obtener DFs desde sistemas heredados y finalmente la descripción de los principales algoritmos que existen actualmente para este fin.

2.1 Definiciones preliminares

2.1.1 Esquemas, estados e instancias de una base de datos

La descripción de una base de datos se denomina esquema de la base de datos, que se especifica durante la fase de diseño y no se espera que cambie con frecuencia. La mayoría de los modelos de datos tienen ciertas convenciones para la visualización de los esquemas a modo de diagramas.

Los datos reales de una base de datos pueden cambiar con frecuencia. Los datos en un momento concreto se denominan estado de la base de datos, también reciben el nombre de conjunto actual de ocurrencias o instancias. En un estado de la base de datos, cada estructura de esquema tiene su propio conjunto actual de instancias; por ejemplo, una relación determinada contendrá el conjunto de entidades individuales (registros) como sus instancias [ENC+02].

2.1.2 Dependencia funcional

Una DF es una restricción que se establece entre dos conjuntos de atributos de la relación. Sea R el conjunto de n atributos de una relación $R = \{A_1, A_2, ..., A_n\}$.

Definición. Una dependencia funcional, denotada por $X \rightarrow Y$, entre dos conjuntos de atributos X e Y que son subconjuntos de R, especifica una restricción en las posibles tuplas que pueden formar un estado de relación r de R. La restricción dice que dos tuplas t_1 y t_2 en r que cumplen que $t_1[X] = t_2[X]$, deben cumplir también que $t_1[Y] = t_2[Y]$ [ENC$^+$02].

Esto significa que los valores del componente Y de una tupla de r dependen de, o están determinados por, los valores del componente X; alternativamente, los valores del componente X de una tupla únicamente (o funcionalmente) determinan los valores del componente Y. Se dice también que existe una dependencia funcional de X hacia Y, o que Y es funcionalmente dependiente de X.

Por lo tanto, X determina funcionalmente Y si para toda instancia r del esquema de relación R, no es posible que r tenga dos tuplas que coincidan en los atributos de X y no lo hagan en los atributos de Y. En base a esto se puede observar lo siguiente:

1. Si una restricción de R indica que no puede haber más de una tupla con un valor X concreto en cualquier instancia de relación $r(R)$, es decir, que X es una clave candidata de R, se cumple que $X \rightarrow Y$ para cualquier subconjunto de atributos Y de R (ya que la restricción de clave implica que dos tuplas en cualquier estado legal $r(R)$ no tendrán el mismo valor de X).
2. Si $X \rightarrow Y$ en R, esto no supone que $Y \rightarrow X$ en R.

Ejemplo 1. *Dada la instancia de la relación de la Tabla 2.1, se pueden obtener las siguientes DFs:*

- $A \rightarrow ABCDEFGH$. *Por lo tanto A es clave candidata.*
- $B \rightarrow H$

Tabla 2.1: Instancia de una relación y sus DFs

A	B	C	D	E	F	G	H
1	a	eth	cdr	cdr	0	0x00	10
2	a	eth	car	cdr	0	0xf0	10
3	b	usb	cdr	car	0	0xff	10
4	b	com	car	car	1	0x68	10
5	c	lpt	cddr	car	1	0xa0	12

- $C \rightarrow B$
- $C \rightarrow F$
- $D,E \rightarrow ABCDEFGH$. *Por lo tanto D, E es clave candidata.*
- *etcétera.*

Se observa por ejemplo que, en el caso de $B \rightarrow H$, la primera y segunda tupla poseen el mismo el valor en el atributo B y en el atributo H, y lo mismo ocurre con el par de tuplas 3 y 4. Por lo tanto, se verifica que el atributo B determina al atributo H.

Cabe destacar que las DFs que se pueden deducir de la Tabla 2.1 son aquellas que solamente se verifican en la instancia y no necesariamente se cumplen en todas las instancias, o son parte del dominio del modelado.

2.1.3 Axiomas de Armstrong

Los Axiomas de Armstrong son un conjunto de axiomas que se utilizan para deducir todas las DFs de una base de datos relacional. Fueron desarrollados por William W. Armstrong en 1974 en [Arm74].

Los Axiomas de Armstrong corresponden a los siguientes tres axiomas de inferencias para DFs definido en conjuntos de atributos X, Y y Z.

1. *Reflexividad:* Si $Y \subseteq X$, entonces $X \rightarrow Y$.
2. *Aumentatividad:* Si $X \rightarrow Y$, entonces $XZ \rightarrow YZ$.

3. *Transitividad:* Si $X \rightarrow Y$ y $Y \rightarrow Z$, entonces $X \rightarrow Z$.

Los axiomas de inferencia 1, 2 y 3 son correctos y completos [Mai83]. Correcto indica que dado un conjunto F de DFs que son satisfechos por la relación r, cualquier DF inferida desde F usando los axiomas 1, 2 ó 3 es satisfecha también por r. Completo indica que los tres axiomas pueden ser aplicados repetidamente para inferir todas las DFs logicamente implicadas por el conjunto F de DFs.

Los siguientes dos axiomas de inferencia pueden ser derivados desde los Axiomas de Armstrong [RG02].

- *Unión:* Si $X \rightarrow Y$ y $X \rightarrow Z$, entonces $X \rightarrow YZ$.
- *Descomposición:* Si $X \rightarrow YZ$, entonces $X \rightarrow Y$ y $X \rightarrow Z$.

Ejemplo 2. *Dada la relación $R(A,B,C,D,E)$ y sus DFs $A \rightarrow B$, $C \rightarrow D$, $D \rightarrow E$. Se puede demostrar que $A,C \rightarrow A,B,C,D,E$:*

1. *$A \rightarrow B$ es dada*
2. *$A,C \rightarrow A,B,C$ se obtiene por aumentatividad de 1 por A,C*
3. *$C \rightarrow D$ es dada*
4. *$D \rightarrow E$ es dada*
5. *$C \rightarrow E$ se obtiene por transitividad de 3 y 4*
6. *$C \rightarrow D,E$ por la unión de 3 y 5*
7. *$A,B,C \rightarrow A,B,C,D,E$ por la aumentatividad de 6 por A,B,C, y finalmente*
8. *$A,C \rightarrow A,B,C,D,E$ por la transitividad de 2 y 7*

2.2 Obtención de dependencias funcionales desde Sistemas Heredados

Los Sistema de Información Heredados son aquellos sistemas críticos para una organización y que generalmente operan la mayor parte del día, en efecto, suelen ser la columna vertebral y el principal vehículo para la consolidación de información. Estos Sistemas Heredados poseen normalmente una infraestructura técnica obsoleta, de bajo rendimiento y costosa en su mantenimiento debido a la documentación escasa haciendo necesaria la comprobación del código fuente para entender su funcionalidad, además de poseer limitadas o nulas posibilidades de interactuar con plataformas modernas orientadas a interfaces de usuario amigables [VCG10].

A raíz de lo anterior, el mayor problema surge con la independencia entre los datos y la lógica de negocio. Dada la estrecha relación que existe, resulta difícil modificar los programas sin que los datos se vean afectados y viceversa. Por otro lado, en la actualidad la migración de datos de los Sistemas Heredados se suele hacer de forma manual, transformándolo en un proceso largo y tedioso, que tiene directa relación con el número de fuentes de datos y tamaño de estas.

La migración de un sistema heredado a sistemas relacionales es un procedimiento muy costoso para las organizaciones que conlleva un riesgo a fracasar, más aún lograr comprender completamente la lógica de desarrollo y la estructura de su base de datos es a veces compleja. No obstante se ha convertido en una opción de transición para muchas organizaciones que siguen empleando modelos de datos no relacionales como archivos planos, bases de datos jerárquica o de red para sus sistemas heredados [Lin08, Sne95].

Para lograr la migración desde sistemas heredados es necesario realizar la extracción del modelo de datos, para lo cual se han propuesto diversas alternativas, tales como la ingeniería inversa descrita en [And94] donde se propone un método para la extracción de un modelo entidad relación a partir de una base de datos relacional. Otras investigaciones como [HRCH07] proponen la obtención del esquema de base de datos heredada CODASYL (modelo de red) a

partir de un proceso semiautomático mediante instrucciones DDL (Data Description Languaje) que no considera la normalización adicional del esquema de base de datos relacional.

Varias de estas metodologías de obtención del modelo relacional desde sistemas heredados poseen en común las siguientes etapas:

1. Extracción: Consiste en la extracción de la estructura y datos desde las fuentes de origen. Estas fuentes pueden ser bases de datos relacionales y no relacionales. En este proceso es necesario analizar los datos extraídos, es decir verificar si los datos cumplen la estructura de origen.
2. Obtención de DFs: Proceso automático que dado un conjunto de datos acotado realiza el análisis para indicar las DFs que se cumplen en él.
3. Generación de Modelo: Etapa final cuyo objetivo es la generación automática del modelo de datos a partir de las DFs obtenidas en la etapa anterior, esta se lleva a cabo mediante herramientas ad-hoc de manera automática y bajo la supervisión de usuarios expertos de manera iterativa hasta completar un modelo que los satisfaga.

Es en la segunda etapa donde la herramienta de obtención de DFs presentada en esta tesis ayudará de una manera automática y eficiente a obtener las DFs desde la fuente de datos del sistema heredado en análisis. Es así como esta etapa cuenta también con las siguientes labores que son fundamentales para un buen proceso de extracción [VCG10]:

- Identificación de DFs del dominio del sistema heredado: Establecimiento y registro de dependencias funcionales que son propias del dominio general del problema que aborda el sistema heredado, es decir, dependencias conocidas por los expertos que además poseen un cierto grado de obviedad semántica.
- Obtención automática de DFs: El resultado de la etapa anterior entrega un problema más acotado respecto de la realidad debido a que las DFs obtenidas previamente permitirán disminuir el tiempo de procesamiento y mejorar el rendimiento debido a la menor cantidad de atributos a analizar.

- Refinamiento de Expertos: Concluida la actividad anterior, es necesario revisar sus resultados, debido a que pueden resultar DFs que sean resultado (coincidencia) de los mismos datos y que no correspondan a la lógica del negocio.

2.3 Algoritmos existentes para la búsqueda de dependencias funcionales

El estudio de Data Mining [FPSSU96] y específicamente la obtención de dependencias funcionales desde un conjunto de datos ha llamado la atención de muchos investigadores en el último tiempo. A mediados de los 80's Mannila y Raiha recogen estudios sobre las relaciones de Armstrong y expone en [MR86] un método para obtener de éstas las dependencias funcionales en una relación, lo que representó la base para idear nuevos algoritmos en los años posteriores.

Los Algoritmos que se han propuesto para resolver este problema se pueden clasificar en tres categorías o enfoques [YH08]: El primer enfoque es el de generación-prueba de DFs candidatas, donde destacan los algoritmos llamados TANE [HKPT99] y FUN [NC01a, NC01b], el segundo es el enfoque del cubrimiento minimal, donde destacan los algoritmos propuestos por Flach et al.[FS99], Lopes et al. [LPL00] y Wyss et al. con su algoritmo FastFDs [WGR01], y finalmente el enfoque de análisis de conceptos formales (FCA) con los algoritmos de Baixeries [Bai04] y el de Lopes et al. [LPL02].

De la clasificación mencionada anteriormente nos interesa conocer en detalle el funcionamiento del algoritmo llamado TANE, que fue propuesto en 1999 por Huthala et al. [HKPT99, HKPT98] y que corresponde a uno de los algoritmos más utilizados de obtención de DFs desde una relación, por lo que varios investigadores han obtenido nuevas variantes de dicho algoritmo. A continuación se presentan los aspectos más relevantes del funcionamiento del algoritmo.

2.3.1 Algoritmo TANE.

Propuesto por Huhtala et al. en [HKPT99]. Corresponde a un algoritmo de la categoría generación y prueba, el cual puede obtener el conjunto de dependencias minimales de una relación. Realiza la búsqueda de DF en base a particiones. Estas particiones se generan de acuerdo a clases de equivalencia (valores que puede tomar un atributo en una tupla).

Denotamos la *clase de equivalencia* de una tupla $t \in r$ con respecto al conjunto $X \subseteq R$ por $[t]_X$, es decir $[t]_X = \{u \in r \mid t[A] = u[A] \text{ para todo } A \in X\}$. Entonces, el conjunto $\pi_X = \{[t]_X \mid t \in r\}$ de clases de equivalencia es una *partición* de r sobre X. Por lo tanto, π_X es una colección de conjuntos disjuntos (clases de equivalencia) de tuplas, en la cual, cada una tiene un único valor para el conjunto de atributos X y la unión de los conjuntos es igual a la relación r. Finalmente, el ranking $|\pi|$ de una partición π es el número de clases de equivalencia en π.

Ejemplo 3. *En la relación del Tabla 2.2, el atributo A tiene valor 1 sólo en las tuplas 1 y 2, entonces estos forman una clase de equivalencia* $[1]_A = [2]_A = \{1,2\}$*, por lo que las particiones respecto al atributo A son* $\pi_A = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}\}$*. A su vez, las particiones respecto a B,C son* $\pi_{\{B,C\}} = \{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$.

Las particiones de cada atributo son:

$\pi_{\{A\}} = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}\}$
$\pi_{\{B\}} = \{\{1\}, \{2, 3, 4\}, \{5, 6\}, \{7, 8\}\}$
$\pi_{\{C\}} = \{\{1, 3, 4, 6\}, \{2, 5, 7\}, \{8\}\}$
$\pi_{\{D\}} = \{\{1, 4, 7\}, \{2\}, \{3\}, \{5\}, \{6\}, \{8\}\}$

Dada entonces la definición de clase de equivalencia y partición podemos definir que una *dependencia funcional* $X \to A$ existe si y sólo si $|\pi_X| = |\pi_{X \cup \{A\}}|$.

El algoritmo también calcula DFs aproximadas, que son encontradas de acuerdo a un error $e(X \to A)$ que es la fracción mínima de tuplas que pueden ser removidas desde la relación para

Tabla 2.2: Ejemplo de una relación y sus particiones

ID Tupla	A	B	C	D
1	1	a	top	Flower
2	1	A		Tulip
3	2	A	top	Daffodil
4	2	A	top	Flower
5	2	b		Lily
6	3	b	top	Orchid
7	3	C		Flower
8	3	C	hill	Rose

que $X \rightarrow A$ se mantenga. Esto provoca una mejora en los tiempos de obtención de DFs con respecto a la no utilización de un factor de error.

En lo que respecta al algoritmo en sí, sus principales características son:

1. Realiza la búsqueda nivel por nivel, tal como muestra la Figura 2.1, donde el cálculo de cada nivel se realiza utilizando los resultados de los niveles previos, reduciendo la cantidad de cálculos necesarios.
2. Tiene dos métodos para reducir la complejidad en tiempo y espacio:
   (a) El primero es reemplazando las particiones por una representación más compacta. Esto lo realiza compactando los atributos en enteros y utilizando tabla hash (posible mejora).
   (b) El segundo es introduciendo un ligero error e para encontrar las dependencias funcionales aproximadas.
3. Utilización de Pruning (técnicas para optimizar la búsqueda):
   (a) Descartar los superconjuntos de conjuntos ya encontrados como DFs. Por ejemplo, si $X \subseteq Y$ $X \rightarrow A$ es candidato, luego $Y \rightarrow A$ no es una DF minimal puesto que Y es un superconjunto de X.

Para encontrar todas las dependencias minimales no triviales, TANE realiza una búsqueda nivel por nivel. Un nivel L_l es la colección de conjuntos de atributos de tamaño l que puede

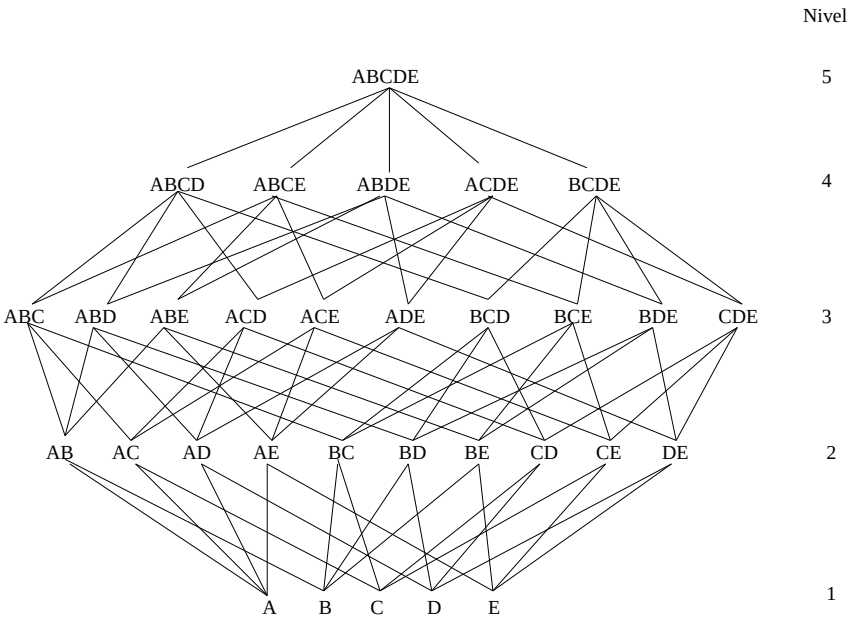

Figura 2.1: Posibles combinaciones de los atributos A, B, C, D y E

ser potencialmente usado para construir dependencias basado en las consideraciones señaladas anteriormente. TANE comienza con $L_1 = \{\{A\}|A \in R\}$, y computa L_2 desde L_1, L_3 desde L_2, etcétera. El Algoritmo 2.1 muestra este hecho.

El procedimiento COMPUTAR_DEPENDECIAS(L_l) encuentra las dependencias minimales a partir de L_{l-1}. El procedimiento PRUNE(L_l) realiza acotamiento del espacio de búsqueda y el procedimiento GENERAR_SIGUIENTE_NIVEL(L_l) forma el siguiente nivel a partir del nivel actual.

Resulta interesante analizar como es la verificación que realiza TANE de la dependecia minimal. El procedimiento COMPUTAR_DEPENDECIAS(L_l), Algoritmo 2.2, realiza esta labor y se basa en en el siguiente lema [HKPT99]:

Lemma 2.1. $C^+(X) = \{A \in R|\forall B \in X : X \setminus \{A,B\} \rightarrow \{B\}\}$

Lemma 2.2. *Sea $A \in X$ y $X \setminus \{A\} \rightarrow A$ una dependencia válida. $X \setminus \{A\} \rightarrow A$ es minimal si y sólo si, para todo $B \in X$, tenemos que $A \in C^+(X \setminus \{B\})$, donde C^+ es el cierre del descriptor definido*

```
   input  : relacion r del esquema R
   output: Dependencias funcionales minimales no triviales que se verifican en R
2  L_0 := {∅}
3  C+(∅) := R
4  L_1 := {{A}|A ∈ R}
5  l := 1
6  while l ≠ ∅ do
8      COMPUTAR_DEPENDENCIAS(L_l)
9      PRUNE(L_l)
10     L_{l+1}:= GENERAR_SIGUIENTE_NIVEL(L_l)
11     l := l + 1
12 end
```

Algoritmo 2.1: Algoritmo TANE

como el subconjunto de atributos máximo de R en el sentido de que la adición de cualquier atributo vulneraría la condición de DF.

```
   input  : L_l
2  foreach X ∈ L_l do
4      C+(X) := ⋂_{A∈X} C+(X \ {A}
5      foreach X ∈ L_l do
7          foreach A ∈ X ∩ C+(X) do
9              if X \ {A → A es válido then
11                 imprimir X \ {A → A}
12                 eliminar A desde C+(X)
13                 eliminar todo B en R \ X desde C+(X)
14             end
15         end
16     end
17 end
```

Algoritmo 2.2: Procedimiento COMPUTAR_DEPENDENCIAS(L_l)

Los pasos 2, 5 y 7 garantizan que el procedimiento genera exactamente las dependencias minimales desde $X \setminus \{A\} \to A$, donde $A \in L_l$ y $A \in X$.

Como se mencionó anteriormente, el funcionamiento de TANE se basa en la búsqueda de DFs por niveles dentro del retículo booleano (formado por la combinación de atributos), obtención de DFs aproximadas de acuerdo a un factor de error y la utilización de técnicas de

pruning para optimizar la búsqueda. El problema es que su rendimiento es bajo cuando en una relación el conjunto R es grande (por ejemplo, más de 20 ó 30 atributos), tal como lo muestran los resultados obtenidos en [HKPT99, YHB02]. Esto, debido a que el tamaño del retículo booleano crece exponencialmente con la cantidad de atributos de la relación. Otros algoritmos propuesto tienen problemas similares.

Por ejemplo, si tenemos un conjunto de atributos R, por ejemplo $\{A,B,C,D,E,F\}$, el espacio de búsqueda de determinantes de DFs de la forma $X \rightarrow U$, para un $U \in R$ determinado, es exponencial en el tamaño de R (llamémoslo n), ya que se deben formar $2^{n-1}-1$ combinaciones, descartando el nivel 0 (en el ejemplo $2^5-1=31$ posibles subconjuntos de atributos candidatos). En la Figura 2.1 se muestran todas las posibles combinaciones si tomamos $U=F$. Estas combinaciones forman un retículo booleano.

2.3.2 FUN.

Es un algoritmo similar al TANE, útiliza el mecanismo de búsqueda de dependencias nivel por nivel. Se diferencia básicamente en las reglas de pruning que utiliza para eliminar DFs candidatas [NC01a, NC01b].

2.3.3 FDEP.

Es un algoritmo del enfoque de cubrimiento minimal. Consiste en tres algoritmos: el algoritmo bottom-up, el algoritmo bi-direccional y el algoritmo top-down. Los experimentos mostrados en [FS99] muestran que el método bottom-up es el más eficiente. Este método primero computa el cubrimiento negativo, que es un cubrimiento que contiene todas las dependencias que son inconsistentes con el conjunto de datos. Luego, en el segundo paso, el método itera sobre estas dependencias, generando el cubrimiento positivo, desde el cual se obtienen las DFs.

2.3.4 FastFDs y Dep-Miner.

Realizan la búsqueda considerando pares de tuplas. Primero, los segmentos de partición de la relación es extraída desde la relación inicial, luego usando estas particiones los conjuntos son computados y se generan los conjuntos maximales. Así, un cubrimiento mínimal de DFs es encontrado desde el conjunto maximal. La diferencia entre FastFDs y Dep-Miner es sólo que este último emplea una búsqueda nivel por nivel, miestras que FastFDs usa la estrategia de búsqueda depth-first [YH08].

Capítulo 3

Dualidad de Hipergrafos

En este capítulo se estudiarán los hipergrafos, que corresponde a una generalización de un grafo, cuyas aristas aquí se llaman hiperaristas, y pueden relacionar a cualquier cantidad de vértices, en lugar de sólo un máximo de dos como en el caso de los grafos.

Se mostrará en las siguientes secciones que el problema de obtención de DFs se puede llevar eficientemente al de dualidad de hipergrafos, para el cual se conocen algoritmos de complejidad $O(n^{\log n})$, es decir sub exponenciales (o cuasi-polinomiales).

Se utilizará en este capítulo esencialmente la notación de [Pol08].

3.1 Hipergrafo

Un *hipergrafo* se define como un grafo generalizado $H = (A, E)$, donde A es un conjunto finito y $E \subseteq \mathcal{P}(A)$ un conjunto de hiperaristas ($\mathcal{P}(C)$ es el conjunto de partes de C). Este concepto fue propuesto por Claude Berge en 1970 [Ber89] y se puede considerar como una generalización del concepto de grafo, en el sentido de que no se requiere que las aristas tengan siempre dos nodos.

Ejemplo. Sea $A = \{a, b, c, d, e, f\}$, y sea H'el conjunto de hiperaristas formado por $\{\{a, b\}, \{b, c\}, \{c, d, e\}, \{f\}\}$. La Figura 3.1 representa el hipergrafo mediante un Diagrama

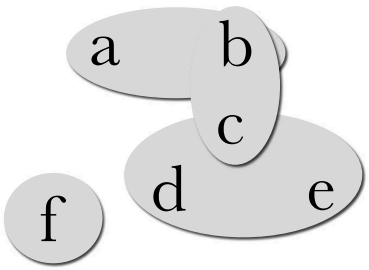

Figura 3.1: Hipergrafo $H' = \{\{a,b\},\{b,c\},\{c,d,e\},\{f\}\}$

de Venn.

Para efectos de una implementación y manipulación computacional es conveniente representar hipergrafos mediante matrices [Pol08]:

$$H' = \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

En el que cada fila corresponde a una hiperarista $X_i \in H'$ definida sobre el conjunto base $A = \{a_1, ..., a_{|A|}\}$, tal que $\forall_i = \{1, ..., |\, H' \,|\}$ y $\forall_j = \{1, ..., |\, A \,|\}$:

$$x_{ij} = \begin{cases} 0 & \text{si } a_j \notin X_i \\ 1 & \text{si } a_j \in X_i \end{cases} \tag{3.1}$$

3.2 Operadores sobre Hipergrafos

Para manipular los hipergrafos, se han definido algunos operadores que ayudan a la obtención de resultados y en la simplificación de definiciones y notaciones [Pol08]. Para todos todos estos

operadores existen algoritmos que permiten mecanizarlos, aunque no todos polinomialmente. Los principales operadores sobre hipergrafos son:

3.2.1 El producto cartesiano: $\vee$

Dado dos hipergrados, $H = \{E_1, E_2, ..., E_m\}$ y $H' = \{F_1, F_2, ..., F_{m'}\}$ el producto cartesiano se define como:

$$H \vee H' = \{E_i \cup F_j | 1 \leq i \leq m, 1 \leq j \leq m'\} \tag{3.2}$$

Produce la unión de todos los posibles pares de hiperaristas, una hiperarista del primer hipergrafo y una del segundo.

3.2.2 El minimal de H : $\mu(H)$

El operador μ aplicado sobre un hipergrafo H genera otro hipergrafo con todas las hiperaristas minimales de H. Se define como:

$$\mu(H) = \{X \in E; \neg \exists Y \in E, Y \subsetneqq Y\} \tag{3.3}$$

Ejemplo. Sea $H' = \{\{c\}, \{b, c\}, \{a, b\}\}$, representado en una matriz como:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Entonces

$$\mu(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Notar que $\mu(H) \subseteq H$, y que $A \in \mu(H)$ sí y sólo si $H = \{\{A\}\}$.

El cálculo de $\mu(H)$ es polinomial, ya que se puede realizar comparando todos con todos, y de esta forma obtener las hiperaristas minimales.

3.2.3 El clutter de $H : \nu(H)$

El operador ν aplicado sobre un hipergrafo H genera el hipergrafo que es respondido por las hiperaristas de H; se define como:

$$\nu(H) = \{X | X \subseteq A; \exists Y \in E, Y \subsetneqq X\} \tag{3.4}$$

Ejemplo. Sea $H' = \{\{c\},\{b,c\},\{a,b\}\}$, representado en una matriz como:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Entonces

$$\nu(H') = \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

Notar que $H \subseteq \nu(H) \subseteq P(A)$, y que siempre $A \in \nu(H)$.

El cálculo de $\nu(H)$ no es polinomial, pues en el peor caso, se genera un número de hiperaristas que crece de forma exponencial.

3.2.4 El complemento del clutter de H : $\nu'(H)$

El operador ν' aplicado sobre un hipergrafo H genera el complemento del hipergrafo que es respondido por las hiperaristas de H; se define como:

$$\nu(H) = \{X | X \subseteq A; \exists Y \in E, Y \supset X\} \tag{3.5}$$

Ejemplo. Sea $H' = \{\{c\},\{b,c\},\{a,b\}\}$, representado en una matriz como:

$$H' = \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Entonces

$$\nu'(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

3.2.5 El blocker de H : $\tau(H)$

El operador τ aplicado sobre un hipergrafo H genera el hipergrafo transversal a las hiperaristas de H; se define como:

$$\tau(H) = \{X \subseteq A; \forall Y \in E, X \cap Y \neq \emptyset\} \tag{3.6}$$

Ejemplo. Sea $H' = \{\{c\},\{b,c\},\{a,b\}\}$, representado en una matriz como:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Entonces

$$\tau(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

Notar que $\tau(H) \subseteq P(A)$, y que siempre y cuando H sea propio, $A \in \tau(H)$, pues A es el máximo de $(P(A), \subseteq)$.

El cálculo de $\tau(H)$ no es polinomial, pues el número de comparaciones que el operador debe realizar para obtener todos los $Z \subseteq P(A)$ que pertenecen a $\tau(H)$ crece exponencialmente.

3.2.6 Los slices de H : $\lambda(H)$

El operador λ aplicado sobre un hipergrafo H genera el hipergrafo de transversales minimales de H. Se define como:

$$\lambda(H) = \mu(\tau(H)) \tag{3.7}$$

Ejemplo. Sea $H' = \{\{c\},\{b,c\},\{a,b\}\}$, representado en una matriz como:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Entonces

$$\lambda(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$

Notar que $\lambda(H) \subseteq \tau(H) \subseteq P(A)$. Se obtienen de esta forma las hiperaristas mínimas que tienen interseción no vacía con cada $X \in H$, entendiendo por mínimas a que $\forall(Z \in \lambda(H)), \nexists Y \subset Z$ tal que $Y \in \tau(H)$.

3.3 Definiciones

Definición. Un par de hipergrafos $G := (H,K)$ sobre A es:

- vacío, si $H = \varnothing$ o $K = \varnothing$.
- coherente, si $\tau(H) \supseteq \nu(K)$.
- completo, si $\tau(H) \subseteq \nu(K)$.
- dual, si $\tau(H) = \nu(K)$, es decir, si es coherente y completo.

Definición. Un *Hitting Set* H' de H, también llamado transversal de H, es un subconjunto de A tal que H' intersecta a las hiperaristas de H. Un hitting set que no incluye a otro hitting set es llamado minimal. Por ejemplo, $\{1,3,4\}$ es un hitting set de H, y $\{1,3\}$ es un minimal hitting set de H. El dual de un hipergrafo H es el conjunto de todos los hitting set minimales de H. Nótese que $\tau(H)$ es el conjunto de transversales de H.

3.4 El problema de dualidad de hipergrafos

El problema de dualidad de hipergrafos tiene muchas aplicaciones directas en lógica, teoría de juegos, indexación de base de datos, diagnóstico basado en modelos, optimización, inteligencia artificial, machine learning, entre muchas otras. Dualización de hipergrafos además tiene muchos problemas equivalentes, a continuación se describen los principales [MU11]:

1. Enumeración del conjunto de cobertura minimal

   Para un subconjunto F definido sobre un conjunto E, un conjunto de cobertura S es un subconjunto de F tal que la unión de los miembros de S es igual a E. Por ejemplo, $E = \bigcup_{X \in S} X$. Un conjunto de cobertura es minimal si este no incluye a otros conjuntos de cobertura. Consideremos que F es un conjunto de vértices, y $F(v)$ es una hiperarista donde $F(v)$ es el conjunto de F $\in F$ que incluye v. Entonces, para el conjunto de hiperaristas F=$\{F(v)|v \in E\}$, un *hitting set* de F es un conjunto de cobertura de F, y vice versa. Así entonces, la enumeración de conjuntos de cobertura minimal es equivalente a la dualización.
2. Enumeración del conjunto de descobertura minimal

Para un subconjunto F definido sobre un conjunto E, un conjunto de descobertura S es un subconjunto de E tal que S no incluye a algún miembro de F. Por ejemplo, $F = \{E \setminus X | X \in F\}$. S no está incluido en $X \in F$ si y sólo si S y $E \setminus X$ tiene una intersección no vacía. Un conjunto de descobertura de F es un *hitting set* de F, y vice versa, por lo que la enumeración del conjunto de descobertura minimal es equivalente a la enumeración de *minimal hitting sets*.

3. Enumeración de circuitos para sistemas independientes.

   Un subconjunto F definido sobre E es llamado un sistema independiente si para cada miembro X de F, algún de estos subconjuntos es también miembro de F. Un subconjunto de E es llamado independiente si este es un miembro de F, y por otro lado dependiente. Un circuito es un conjunto dependiente minimal, por ejemplo, un conjunto dependiente que no contiene a otro conjunto dependiente. Cuando un sistema independiente es dado por el conjunto de un conjunto dependiente maximal de F, entonces la enumeración de circuitos de F es equivalente a la enumeración de conjuntos de descobertura de F.

4. Transformación de Forma Normal Disyuntiva a Forma Normal Conjuntiva.

   Forma Normal Disyuntiva (FND) es una fórmula donde las cláusulas están compuestas por literales conectados por "or" y las cláusulas están conectadas por "and". Forma Normal Conjuntiva (FNC) es una fórmula donde las cláusulas están compuestas por literales conectados por "and" y las cláusulas están conectadas por "or". Una fórmula puede ser representada como una FND y una FNC. Sea D una fórmula en FND compuestas de variables $x_1, ..., x_n$ y cláusulas $C_1, ..., C_m$. Una FND/FNC es llamada monótona si no contiene cláusulas con literales con "not". Entonces, S es un *hitting set* de cláusulas de D si y sólo si la asignación obtenida mediante el establecimiento de literales en S es verdarero dado una asignación verdadera de D. Sea H una fórmula minimal en FNC equivalente a D, H tiene que incluir algún *minimal hitting set* de D como cláusula, donde alguna cláusula de H debe contener al menos un literal de alguna cláusula de D. Así, una FNC minimal

equivalente a D debe incluir todos los *minimal hitting set* de D. Por la misma razón, computar la FND minimal desde una FNC es equivalente a la dualización.

Ejemplo. Las fórmulas booleanas $\phi = p_1 \vee (p_2 \wedge p_3) = (p_1 \vee p_2) \wedge (p_1 \vee p_3)$ y $\psi = p_1 \wedge (p_2 \vee p_3)$ son duales, porque intercambiando los $\vee$ y $\wedge$ de una fórmula, se obtiene la otra. Estas fórmulas vistas como hipergrafos serían, respectivamente, $H := \{\{p_1, p_2\}, \{p_1, p_3\}\}$ y $K := \{\{p_1\}, \{p_2, p_3\}\}$, porque $\lambda(H) = \mu(\tau(H)) = \mu(\{\{p_1\}, \{p_1, p_2\}, \{p_1, p_3\}, \{p_2, p_3\}, \{p_1, p_2, p_3\}\}) = \{\{p_1\}, \{p_2, p_3\}\} = K$.

En el presente trabajo, y de acuerdo al nuevo método propuesto, se define el siguiente problema de dualización: dados H, K hipergrafos minimales, decidir si $K = \lambda(H)$.

Ejemplo. El dual $K = (A, E')$ de H es el hipergrafo tal que E' es la corrección del minimal hitting set de E. Al tener $A = \{1,2,3,4\}$, $E = \{\{1,2\}, \{1,3\}, \{2,3,4\}\}$, el conjunto $\{1,3,4\}$ es un hitting set pero no minimal, y $\{2,3\}$ si es un minimal hitting set. El dual de $H = (A, E)$ es dado por el conjunto $E' = \{\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}\}$. El problema de dualización entonces, es poder computar el dual de un hipergrafo $H = (A, E)$.

Para este problema Fredman y Khachiyan propusieron en 1996 un algoritmo sub-exponencial en [FK96], lo que permite conjeturar que no pertenece a la clase *NP*-completo. Pero tampoco se sabe si pertenece a la clase *P*, o al menos *NP* [EMG08].

En los últimos años se han propuestos nuevos algoritmos que también han resultado eficientes, como es el caso del propuesto por Kavvadias y Stavropoulos en 1999 y Murakami y Uno el 2011. Todos estos algoritmos se analizan en la siguiente sección.

3.5 Algoritmos para el cálculo de transversales minimales

Numerosos estudios se han realizado en el problema de dualización, logrando con esto la propuesta de variados algoritmos que buscan resolver el problema en un tiempo óptimo. Los algoritmos propuesto a lo largo del tiempo se pueden clasificar en dos tipos de acuerdo a su

estructura: variantes al algoritmo de Berge [Ber89] y algoritmos del tipo hill-climbing.

3.5.1 Algoritmo de Berge

Fue propuesto por Claude Berge en [Ber89] y corresponde al algoritmo elemental para el cálculo de transversales minimales en un hipergrafo.

Dado un hipergrafo $H = \{E_1, ..., E_m\}$ en A. Sea $H_i = \{E_1, ..., E_i\}$, con $i = 1, ..., m$, el hipergrafo parcial de H en A. Entonces :

$$\lambda(H_i) = \mu(\tau(H_{i-1}) \vee \tau(E_i))$$
$$\lambda(H_i) = \mu(\tau(H_{i-1}) \vee \{\{v\}, v \in E_i\})$$

La forma en que realiza el cálculo es computando iterativamente las transversales minimales del hipergrafo parcial H_{i-1} y entonces calcula el producto cartesiano de $Tr(H_{i-1})$ por la $i-$ésima hiperarista E_i de H, para luego remover todos los elementos que no son minimales. El Algoritmo 3.1 muestra los pasos que realiza de forma iterativa, computando así todas las hiperaristas H_i.

Ejemplo. Sea $H = \{\{4,5\}, \{1,2,3\}, \{1,2,5\}\}$, aplicamos el algoritmo de Berge para encontrar las transversales minimales:

1. En la primera iteración se tiene $\{4,5\} \vee \{1,2,3\}$, que produce el conjunto de transversales $\{\{1,4\}, \{2,4\}, \{3, 4\}, \{1, 5\}, \{2, 5\}, \{3, 5\}, \{1, 2\ 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 5\}, \{1, 3, 5\}, \{2, 3, 5\}\}$.
2. Al aplicar el operador μ al conjunto anterior resultan las transversales minimales $\{\{1,4\}, \{2,4\}, \{3,4\}, \{1,5\}, \{2,5\}, \{3,5\}\}$.
3. En la siguiente iteración el producto cartesiano del conjunto anterior con la última hiperarista: $\{\{1,4\}, \{2,4\}, \{3,4\}, \{1,5\}, \{2,5\}, \{3,5\}\} \vee \{1,2,5\}$ produce el conjunto de transversales: $\{\{1, 4\}, \{2, 4\}, \{1, 5\}, \{2, 5\}, \{3, 5\}, \{1, 2, 4\}, \{1, 4, 5\}, \{2,4,5\}, \{1,3,4\}, \{2, 3,4\}, \{3, 4, 5\}, \{1, 2, 5\}, \{1, 3, 5\}, \{2,3,5\}\}$.

4. Finalmente, a aplicar el operador μ al conjunto anterior, se obtiene el conjunto final de transversales minimales: $\{\{1,4\},\{2,4\},\{1,5\},\{2,5\},\{3,5\},\{1,3,4\},\{2,3,4\},\{3,4,5\}\}$.

La principal desventaja de este algoritmo, es que produce demasiadas transversales intermedias, producto de la operatoria todos con todos que realiza.

input : H

1 **for** $i = 2,...,m$ **do**

3 | Obtener $\lambda(H_{i-1})$

4 | Computar $\lambda(H_i) = \mu(\tau(H_{i-1}) \vee \{\{v\}, v \in E_i\})$

5 **end**

6 return $\tau(H_m)$

Algoritmo 3.1: El Algoritmo de Berge

3.5.2 Algoritmo de Fredman y Khachiyan

Expuesto por Fredman y Khachiyan en [FK96, KBEG06], desde el punto de vista de la complejidad teórica, constituye la mejor solución hasta ahora conocida para resolver el Problema de Dualidad.

Es un algoritmo sub-exponencial, pero a su vez super-polinomial, es decir su complejidad es mayor que cualquier polinomio pero sigue siendo significativamente menor que la exponencial. Se basa principalmente en la construcción de un árbol binario, cuyos nodos representan estructuras de hipergrafos coherentes. El nodo raíz corresponde a la estructura original y cada par de nodos intermedios son estructuras más pequeñas (también coherentes) formadas a partir de una descomposición de la estructura de su respectivo nodo padre. Si en algún determinado momento se verifica que todas las hojas del árbol actual son completas, entonces la estructura original también es completa. Si en algún momento se verifica que al menos una hoja del árbol actual es incompleta, entonces la estructura original también es incompleta.

Para descomponer cada estructura, se debe elegir un $a \in A$, que cumpla con alguna característica particular que la haga más provechosa que las demás. Por ejemplo, se puede elegir una que esté presente en una hiperarista X de cardinalidad pequeña. Una vez realizada la descomposición de un nodo, se debe aplicar el operador μ a las nuevas estructuras. Las condiciones de parada del algoritmo, vale decir, de la construcción del árbol binario (que puede hacerse recursivamente recorriéndolo en profundidad), pueden ser variadas. Por ejemplo, la condición de semi-completitud puede ser suficiente para verificar incompletitud, y cuando la cardinalidad de alguna hiperarista es igual a 1 ó 0, resulta sencillo (polinomial) verificar la completitud.

Se puede mostrar que este algoritmo tiene complejidad $O(n^{logn})$ en el peor caso. Es el mejor algoritmo conocido en términos de la complejidad en el peor caso.

3.5.3 Algoritmo de Kavvadias y Stavropoulos

Propuesto por Kavvadias y Stravropoulos en el año 2005 [KS05]. Consiste en una modificación al algoritmo de Berge basada en la generación de nodos generalizados y búsqueda primero en profundidad (depth-first search).

Definición. Dado H en A. El conjunto $X \subseteq A$ es un *Nodo Generalizado* de H si todos los elementos en X pertenecen a las mismas hiperarista de H. Es decir:

1. $A = X_1 \cup X_2 \cup ...X_k$
2. $X_i \cap X_j = \oslash$
3. $\forall i = 1,..,k, \forall a,b \in A,$
   $a \in X_i \wedge b \in X_i \Rightarrow \forall Y \in H, a \in Y \Leftrightarrow b \in Y$

Considerando un hipergrafo $H = \{E_1,...,E_m\}$ definido en un conjunto A de n nodos. Dado el conjunto de nodos generalizados $X_1, X_2, \ldots, X_k$ del hipergrafo parcial $H_i = \{E_1, \ldots, E_i\}$ de H, $k_i \geq 1$, se calculan estos nodos generalizados y se computa $\lambda(H_i)$ donde en cada paso se adiciona

la siguiente hiperarista E_{i+1} para definir el hipergrafo $H_{i+1} = H \cup \{E_{i+1}\}$. El hecho de agregar esta nueva hiperarista provocará nuevas determinaciones sobre los nodos generalizados que se han calculado. Existen tres posibles casos para cada nodo generalizado X de H_i:

- (α) $X \cap E_{i+1} = \varnothing$. En este caso. X es también un nodo generalizado de H_{i+1}.
- (β) $X \subset E_{i+1}$. En este caso. X es también un nodo generalizado de H_{i+1}.
- (γ) $X \cap E_{i+1} \neq \varnothing$ y $X \nsubseteq E_{i+1}$. En este caso. X es dividido en $X_1 = X \setminus (X \cap E_{i+1})$ y $X_2 = X \cap E_{i+1}$. Ambos, X_1 y X_2 son nodos generalizados de H_{i+1}.

El algoritmo ahora determina: si (α) o (β) es el caso para todos los nodos generalizados de H_{i+1}, entonces todas las transversales minimales y E_{i+1} permanecen como estaban. Si se cumple (γ), se asume que un nodo generalizado fue dividido en X_1 y X_2. De esta forma se sigue iterando hasta lograr computar todos los hipergrafos parciales con $\lambda(H_i)$. El Algoritmo 3.2 muestra formalmente cada uno de los pasos de este algoritmo.

En este enfoque se introduce el concepto de nodos generalizados en los pasos intermedios, lo que reduce notoriamente la cantidad de transversales intermedias que se calculan con el enfoque tradicional.

Ejemplo. Considere que las dos primeras hiperaristas de un hipergrafo tienen 100 nodos cada una: $E_1 = \{v_1, ..., v_{100}\}$ y $E_2 = \{v_{51}, ..., v_{150}\}$. El hipergrafo parcial $\{E_1, E_2\}$ tiene 2550 transversales, en las que 2500 poseen dos nodos y 50 un sólo nodo. Utilizando el enfoque de nodo generalizado, tendremos sólo 2 transversales que considerar, las cuales serán los conjuntos $\{v_{51}, ...v_{100}\}$ y el conjunto $\{\{v_1, ..., v_{50}\}, \{v_{101}, ..., v_{150}\}\}$.

Ejemplo. Considere el hipergrafo con 5 nodos y 4 hiperaristas $H = \{\{1,2,3\}, \{3,4,5\}, \{1,5\}, \{2,5\}\}$. El árbol de transversales correspondiente a la adición de hiperaristas acorde al orden descrito se muestra en la Figura 3.2. Los nodos generalizados se señalan con círculos. Entonces, en el árbol formado por el algoritmo se puede observar que a partir de la primera hiperarista y la segunda $\{1,2,3\}$ y $\{3,4,5\}$ se obtienen las transversales minimales parciales $\{\{1,2\}, \{4,5\}, \{3\}\}$ y así sucesivamente con las demás hiperaristas.

input : H

for $k = 0, \ldots, m-1$ **do**

  Agregar E_{k+1}

  Actualizar el conjunto de nodos generalizados

  Expresar $\tau(H_k)$ y E_{k+1} como conjuntos de nodos generalizados del nivel $k+1$

  Computar $\lambda(H_{k+1}) = \mu(\tau(H_k) \vee \{\{v_x\} : v_x \in E_{i+1}\})$

end

Retornar $\lambda(H_m)$

Algoritmo 3.2: Algoritmo KS (Modificación al Algoritmo de Berge)

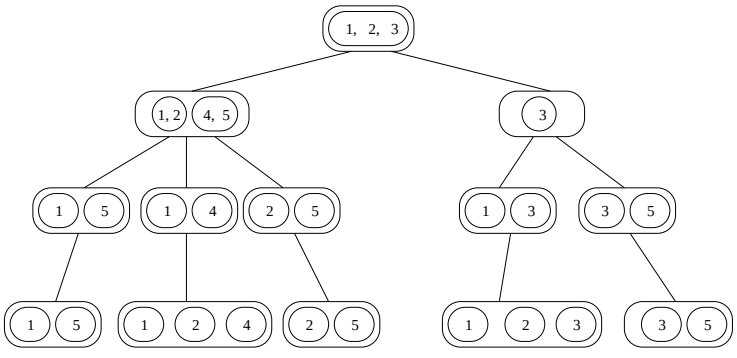

Figura 3.2: Árbol de transversales del hipergrafo $H' = \{\{1,2,3\},\{3,4,5\},\{1,5\},\{2,5\}\}$.

3.5.4 Algoritmo de Murakami y Uno

Corresponde a dos algoritmos propuestos por Murakami y Uno en [MU11]. Consisten en modificaciones a enfoques existentes agregando el concepto de hiperaristas críticas y nuevas técnicas de pruning para optimizar las búsquedas.

Definición. Hiperarista crítica se define como: sea $S \subseteq A$, $uncov(S)$ denota el conjunto de hiperaristas que no se intersectan con S, es decir $uncov(S) = \{X | X \in H, X \cap S = \varnothing\}$. S entonces es una transversal de H si y sólo si $uncov(S) = \varnothing$. Para un vértice $v \in S$, una hiperarista $X \in H$ es llamada crítica para v si $S \cap X = \{v\}$. Se define a todas las hiperaristas críticas para un v como $crit(v,S)$, es decir $crit(v,S) = \{X | X \in H, S \cap X = \{v\}\}$.

El primer algoritmo está basado en búsqueda inversa, y puede ser considerado como un versión modificada del algoritmos KS propuesto por Kavvadias y Stavropoulos, explicado anteriormente. La estrategia de búsqueda es esencialmente equivalente al algoritmo KS con la salvedad que evita todas las iteraciones redundantes en las que no se añade ningún vértice al subconjunto de vértices actual.

La búsqueda comienza con el conjunto vacío. Cuando se visita un subconjunto de vértices S, encuentra todos los menores de S iterativamente y genera una llamada recursiva para cada uno. De esta manera, se puede realizar una búsqueda sólo para encontrar menores del subconjunto de vértices actual. El Algoritmo 3.3 muestra este procedimiento de búsqueda.

Para actualizar crit[] y *uncov* se requiere tiempo $O(|H(A)|)$, así una iteración de este algoritmo toma tiempo $O(||H||)$, donde $||H||$ es la suma de los tamaños de las hiperaristas de H. En total, el algoritmo RS toma tiempo $O(||H|| \times ||S||)$.

Importante también son las técnicas de pruning que se realizan en este algoritmo. Específicamente están enfocadas a encontrar todas las violaciones de vértices antes del paso 10 y poder obtener todos los *hitting set* minimales $S \cup v$ hallados en el proceso. De esta manera, el loop desde el paso 10 al paso 18 sólo considerará aquellos vértices que no violan la condición y de esta forma evitar realizar llamadas recursivas.

```
   input : S
1  global variable: critic[], uncov
3  if uncov = ∅ then
5  |   output S
6  |   return
7  end
9  i := min{ j | F_j ∈ uncov}
10 foreach v ∈ F_i do
12 |   call Update_crit_uncov(v, crit[], uncov)
13 |   if min{t | F_t ∈ crit[f]} < i for each f ∈ S then
14 |   |   call RS(S')
15 |   end
17 |   recuperar los cambios de crit[] y uncov en 12
18 end
```

Algoritmo 3.3: Algoritmo RS

El segundo algoritmo, consiste en un algoritmo con búsqueda primero en profundidad, su procedimiento se detalla en el Algoritmo 3.4. La estrategia de búsqueda es simple, comienza con el conjunto S vacío y luego se agregan vértices uno por uno, verificando la condición de que algún vértice en S puede generar una hiperarista crítica. Se verifica además si S es una transversal y si esta transversal es minimal. Para evitar la duplicación en cada iteración, sólo se agregan vértices más grandes que el máximo vértice en S.

En este método se utiliza una lista de vértices llamada *CAND* que representa los vértices que han sido agregados en cada iteración, comienza con $S = \varnothing$ y agrega vértices recursivamente descartando aquellos que violen la condición de minimalidad.

La correctirud del algoritmo está dada por: Dado un hipergrafo $H = (A,E)$, sea $Z(H)$ el conjunto de conjuntos de vértices de S tal que algún vértice en S tal que es una hiperarista crítica. Se observa que al eliminar un vértice desde S, no pierde su calidad de hiperarista crítica. Por ejemplo $crit(u,S)$ está incluido en $crit(u,S-v)$ para algún vértice v. Entonces, para algún X de $Z(H)$, algún subconjunto de X está también en $Z(H)$ por monotonía.

Ejemplo. Para un hipergrafo $H(A,E)$, $A = \{1,2,3,4\}$, $H = \{\{1,2\},\{1,3\},\{2,3,4\}\}$ y $Z(H) = \{\{1\},\{2\},\{3\},\{4\},\{1,2\},\{1,3\},\{1,4\},\{2,3\}\}$. Una transversal minimal es un

elemento máximo (borde positivo) en $Z(H)$. Entonces, el algoritmo genera algún X de $Z(H)$ como resultado de enumerar todos los elementos de $Z(H)$.

```
   input : S
1  global variable: critic[], uncov, CAND
3  if uncov = ∅ then
5  |  output S
6  |  return
7  end
9  elegir una hiperarista F from uncov
10 C := CAND ∩ F
11 CAND := CAND \ C
12 foreach v ∈ C do
14 |  call Update_crit_uncov(v, crit[], uncov)
15 |  if crit(f, S') ≠ ∅ for each f ∈ S then
17 |  |  call DFS(S ∪ v)
18 |  |  CAND := CAND ∪ v
19 |  end
21 |  recuperar los cambios de crit[] y uncov en 14
22 end
```

Algoritmo 3.4: Algoritmo DFS

Similar al Algoritmo 3.3 RS, el algoritmo DFS requiere tiempo $O(||H||)$ en cada iteración.

Este algoritmo se utilizará para el cálculo de transversales minimales de un hipergrafo, correspondiente al último paso del método para la extracción de DFs que se detalla en el próximo capítulo.

Capítulo 4

Extracción de dependencias funcionales basada en dualidad de hipergrafos

En este capítulo se expone el enfoque propuesto para la obtención de DFs desde relaciones de bases de datos utilizando refutaciones entre atributos y teoría de hipergrafos, con el uso de algoritmos capaces de determinar las transversales minimales entre las hiperaristas de un hipergrafo. La validez del método se basa en lo expuesto en [MR86] y explicitado en la sección 2.1.2, esta validez indica que dado un conjunto F de DFs que se satisfacen en la relación r, cualquier DF inferida desde F usando los axiomas de Armstrong, *Reflexividad, Aumentación y Transitividad* es satisfecha también por r. Su completitud indica que los tres axiomas de Armstrong pueden ser aplicados repetidamente para inferir todas las DFs lógicamente implicadas por un conjunto F de DFs [Mai83].

4.1 Descripción del Método

A continuación, se describe de forma general el enfoque propuesto para la extracción de DFs desde una instancia r de una relación, basado en refutaciones de DFs e hipergrafos: para un

atributo $D \in R$,

1. Obtener el conjunto de refutaciones de DFs; una refutación viene dada por un par de tuplas que coinciden en los antecedentes y discrepan en el consecuente. Por lo tanto un conjunto de refutaciones es un conjunto de pares de tuplas $(t,u) \in r$ tales que $(\forall V \in X)t[V] = u[V] \wedge t[D] \neq u[D]$. Una refutación se denotará con el símbolo $\nrightarrow$ y serán de la forma $X \nrightarrow D$, con $X \subset R$, siendo X el antecedente y D el consecuente.
2. Del conjunto obtenido anteriormente sólo considerar las refutaciones maximales, es decir, se obviarán aquellas refutaciones que sean subconjunto de otra (para el mismo consecuente). Por ejemplo, si tenemos las refutaciones $A,B,C \nrightarrow D$ y $A,B \nrightarrow D$, sólamente consideramos $A,B,C \nrightarrow D$. De esta forma todas las refutaciones maximales producirán un hipergrafo con hiperaristas maximales. Notar que si $H = \{ X_1, X_2, \ldots, X_k \}$ es el conjunto de refutaciones maximales, entonces todos los subconjuntos de los X_i están refutados. Por ejemplo si $H = \{\{A,B,C\}\}$, es decir si tenemos $\{A,B,C\} \nrightarrow D$ entonces también tenemos $\{A,B\} \nrightarrow D$, $\{B,C\} \nrightarrow D$, etc. Por lo tanto el operador para obtener todas las refutaciones posibles es ν' (obtener todos los subconjuntos).
3. Consideremos entonces: $H' = \{A \setminus X_1, A \setminus X_2, \ldots, A \setminus X_k\}$. Si tomamos cualquier $Z \in \nu(H')$, entonces $A \setminus Z$ es una DF refutada, por lo tanto el complemento, $\wp(A) \setminus \nu(H')$, entrega precisamente todos los X tales que $(A \setminus X) \rightarrow D$ es una DF.
4. Se obtendrán las DFs en forma minimal. Es decir, si tenemos por ejemplo $A \rightarrow D$ y $A,B \rightarrow D$, entonces sólo consideramos $A \rightarrow D$. En forma general si tenemos $X \rightarrow D$, todos los $Y \supseteq X$ también determinan a D, por lo tanto solamente conservamos X.
5. Finalmente, tomando $\wp(A) \setminus \nu(H')$ y conservando las DFs minimales obtenemos un hipergrafo H''. Es decir, los antecedentes de las DFs minimales son $H'' = \mu(\{A \setminus X; X \in \wp(A) \setminus \nu(H')\})$.

Para $H'' = \lambda(H')$ tenemos que $\{A \setminus X; X \in \wp(A) \setminus \nu(H')\} = \tau(H')$. En efecto, sea $Y \in \wp(A) \setminus \nu(H')$. Es decir, Y no es superconjunto de ningún $X \in H'$. Esto significa que $\forall X \in$

ID Tupla	1	2	3	4	5	6	7	8	9
1	a_1	b_3	c_2	d_1	e_4	f_3	g_5	h_5	i_1
2	a_1	b_3	c_3	d_3	e_1	f_5	g_3	h_3	i_2
3	a_2	b_3	c_5	d_1	e_5	f_3	g_2	h_4	i_4
4	a_3	b_3	c_2	d_3	e_3	f_4	g_1	h_1	i_1
5	a_4	b_2	c_2	d_8	e_2	f_1	g_4	h_4	i_1
6	a_5	b_4	c_4	d_1	e_3	f_3	g_1	h_1	i_3
7	a_1	b_1	c_3	d_7	e_5	f_2	g_4	h_1	i_2

Figura 4.1: Instancia del conjunto de atributos $R = \{1,2,3,4,5,6,7,8,9\}$

$H' \exists a \in X a \notin Y$, es decir $a \in A \backslash Y$. Por lo tanto $A \backslash Y$ es una transversal de H'.

De esta forma se evitará el hecho de recorrer todo el retículo booleano con todas las combinaciones posibles del conjunto de atributos (mostrado en la Figura 2.1), que es de tamaño exponencial.

El funcionamiento de este método se realizará sobre instancias en las que los valores de los atributos se encontrarán previamente codificados, de tal forma de no trabajar con los valores reales sino con códigos que utilicen menos espacio en memoria (por ejemplo, números en vez de texto).

Ejemplo. Dado el conjunto de atributos $R = \{1,2,3,4,5,6,7,8,9\}$ y la instancia r que se muestra en la Figura 4.1 se realiza la búsqueda de refutaciones para el atributo 6.

Observamos que las tuplas 1 y 2 producen la refutación $\{1,2\} \nrightarrow 9$, las tuplas 1 y 3 producen la refutación $\{2,4,6\} \nrightarrow 9$, las tuplas 1 y 6 producen la refutación $\{4,6\} \nrightarrow 9$, etcétera, hasta las tuplas 6 y 7, que producen $\{8\} \nrightarrow 9$. Conservando solamente las refutaciones maximales obtenemos el hipergrafo $H = \{\{1, 2\}, \{2, 4, 6\}, \{5, 7, 8\}\}$ en el primer paso del método. Luego, obtenemos el hipergrafo de los complementos de las hiperaristas de H, que es $H' = \{\{3, 4, 5, 6, 7, 8\}, \{1, 3, 5, 7, 8\}, \{1, 2, 3, 4, 6\}\}$. Finalmente el operador λ produce las dependencias funcionales: $H'' = \lambda(H') = \{\{1,4\},\{1,5\},\{1,6\},\{1,7\},\{1,8\},\{2,5\},\{2,7\},\{2,8\},\{3\},\{4,5\},\{4,7\},\{4,8\},\{5,6\},\{6,7\},\{6,8\}\}$. Es decir, hemos descubierto las dependencias funcionales $\{1,4\} \rightarrow 9, \{1,5\} \rightarrow 9, \{1,6\} \rightarrow 9, \{1,7\} \rightarrow 9, \{1,8\} \rightarrow 9, \{2,5\} \rightarrow 9, \{2,7\} \rightarrow$

$9, \{2,8\} \rightarrow 9, \{3\} \rightarrow 9, \{4,5\} \rightarrow 9, \{4,7\} \rightarrow 9, \{4,8\} \rightarrow 9, \{5,6\} \rightarrow 9, \{6,7\} \rightarrow 9$ y $\{6,8\} \rightarrow 9$.

Nótese que las DFs obtenidas a partir de una instancia particular de una relación pueden no resultar siempre válidas en la realidad (es decir que pudieran ser coincidencias de los datos). En una situación real estas deberán ser confirmadas por expertos en el ámbito del negocio. Nuestra conjetura es que la cantidad de DFs producto de coincidencias puede ser alta en general, dado que existen muchas condiciones que pueden ser producto de las circunstancias y no de una regla general. Estas DFs circunstanciales no son resultados erróneos, dado que su identificación depende de la interpretación de los datos desde el punto de vista del negocio, por ejemplo una misma instancia de una base de datos en distintos negocios se puede interpretar de distinta forma.

4.2 Implementación

4.2.1 Estructura de datos utilizadas

Para la manipulación computacional de hipergrafos se utilizó matrices de campos binarios, en el que cada fila corresponde a una hiperarista $X_i \in H'$ definida sobre el conjunto base $A = \{a_1, ..., a_{|A|}\}$, tal que $\forall_i = \{1, ..., |\, H' \,|\}$ y $\forall_j = \{1, ..., |\, A \,|\}$. La siguiente matriz muestra el ejemplo de la representación de un hipergrafo H':

$$H' = \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

De esta forma, las operaciones sobre los hipergrafos se puede realizar cómodamente, permitiendo realizar comparaciones entre hiperaristas de maneran simple.

Las principales estructuras de datos utilizadas en la implementación de estos algoritmos corresponden a listas enlazadas, árboles binarios (con su variante balanceada AVL) y árboles B. Además, para la codificación de datos del conjunto de entrada se utilizaron tablas Hash.

4.2.2 Pasos del método

Como se señaló anteriormente, las DFs se obtienen de la aplicación de algoritmos que calculan las transversales a hipergrafos. A continuación se detalla como se realiza el armado de estos hipergrafos desde el conjunto de datos.

1. *Codificación de los atributos:* El funcionamiento de este método se realiza sobre instancias en las que los valores de los atributos se encuentran previamente codificados, de tal forma de no trabajar con los valores reales sino con códigos que utilicen menos espacio en memoria (por ejemplo, números en vez de texto). De esta forma, como primera instancia de implementación, se llevó a cabo un pre-procesamiento de la relación a analizar. Para este fin se utilizaron tablas hash cuyas funciones hash generaban códigos de tamaño mínimo, con el propósito de asignar a distintos valores en los atributos una codificación acorde. La asignación de esta codificación permite reducir considerablemente el uso de la instancia en memoria y además optimizar las consultas de comparación cuando se trata de cadenas de texto.
2. *Búsqueda de refutaciones:* Se realiza una comparación cuadrática sobre las tuplas en búsqueda de refutaciones sobre los atributos. Es importante destacar que estas refutaciones se obtienen en su forma maximal, para mantener esta maximalidad se utilizaron listas comparables creadas para este fin.
3. *Generación de hipergrafos con hiperaristas maximales:* Con el resultado de la etapa anterior se procede a armar el hipergrafo utilizando la estructura matriz especificada anteriormente. Este hipergrafo tiene la característica de estar formado por hiperaristas maximales, que corresponden a las refutaciones maximales.

4. *Cálculo de hipergrafos complemento:* A los hipergrafos obtenidos en la etapa anterior se les calcula su hipergrafo complemento. Este paso es sencillo, ya que sólo requiere intercambiar los 0's por 1's y vice-versa en la matriz que representa el hipergrafo.
5. *Cálculo de las transversales minimales:* Finalmente, para resolver el problema de dualización de hipergrafos se llevó a cabo la implementación de tres algoritmos que son capaces de realizar el cálculo de λ sobre un hipergrafo, vale decir, obtener todas las transversales minimales de dicho hipergrafo. Los algoritmos implementados son:
   - Algoritmo de Berge (ver Algoritmo 3.1),
   - Algoritmo de Kavvadias y Stavropoulos (ver Algoritmo 3.2) y
   - Algoritmo de Murakami y Uno (ver Algoritmos 3.3 y 3.4)

   Cabe destacar que se realizó la implementación de los dos primeros algoritmos y el tercero, de Murakami y Uno, fue provisto por los autores para luego modificarlo y adaptarlo a nuestra herramienta.

El primer paso del método, referente a la codificación, se realiza en un pre-procesamiento considerenado que requiere un tiempo importante. Para los pasos posteriores, búsqueda de refutaciones y cálculo de λ, la complejidad teórica del método corresponde a la suma de operaciones cuadrática ($O(n^2)$) sobre $|r|$ y subexponencial sobre $|R|$ ($O(n^{logn})$) respectivamente.

A continuación se presenta el la implementación del método propuesto en sus diferentes versiones de acuerdo al algoritmos de dualización de hipergrafos utilizado: El Algoritmo 4.1 corresponde al método utilizando el algoritmo de Berge, el Algoritmo 4.2 utilizando el de Kavvadias y Stavropoulos y el Algoritmo 4.3 utilizando el de Murakami y Uno.

```
input : r
refs := call REFS-MAX(r)
foreach H ∈ refs_i do
    foreach X ∈ H_i do
        X := X^C
    end
end
foreach H ∈ refs_i do
    DFs := DFs ∪ BERGE(H)

end
```

Algoritmo 4.1: Algoritmo HIP-BERGE

```
input : r
refs := call REFS-MAX(r)
foreach H ∈ refs_i do
    foreach X ∈ H_i do
        X := X^C
    end
end
foreach H ∈ refs_i do
    DFs := DFs ∪ KS(H)

end
```

Algoritmo 4.2: Algoritmo HIP-KS

```
input : r
refs := call REFS-MAX(r)
foreach H ∈ refs_i do
    foreach X ∈ H_i do
        X := X^C
    end
end
foreach H ∈ refs_i do
    DFs := DFs ∪ RS(H)

end
```

Algoritmo 4.3: Algoritmo HIP-RS

4.3 Pruebas y Resultados Experimentales

4.3.1 Datos de Prueba

Se utilizaron distintas instancias de relaciones de bases de dato reales, las cuales se obtubieron principalmente desde el repositorio *UCI repository of machine learning databases* [BM98].

La Tabla 4.1 muestra las instancias reales utilizadas en las pruebas experimentales. Cabe destacar que las pruebas se realizaron con instancias de sistemas heredados (las tres primeras) e instancias de sistemas de información no heredados (las tres últimas).

4.3.2 Ambiente de Prueba

Las pruebas se realizaron en un clúster (utilizando un sólo procesador) con procesador de 3.2 GHz y 8 GB de memoria RAM. Los distintos algoritmos fueron implementados en el lenguaje de programación C. También se implementaron algunas variantes de algoritmos en Java.

4.3.3 Resultados

Se realizaron pruebas experimentales con las instancias de relaciones reales mostradas en la sección anterior. De acuerdo a la característica de estas instancias se formaron tres categorías y en base a ellas se centró el análisis: de 100 a 500 tuplas, 500 a 2000 tuplas y de 3000 a 5000 tuplas. Para cada una de estas categorías se realizaron pruebas considerando como número de atributos: hasta 20 atributos y hasta 50 atributos.

La ejecución de las pruebas se realizó con los siguientes 4 algoritmos (detallado en las secciones anteriores): TANE, HIP-BERGE, HIP-KS y HIP-RS. TANE corresponde al algoritmo más popular de extracción de DFs, los otros son las diferentes variantes de la propuesta de esta tesis.

Se midió el tiempo de demora de los cuatro algoritmos y la cantidad de DFs que obtuvieron

Tabla 4.1: Instancias utilizadas. Obtenidas desde UCI Repository of machine learning databases (http://archive.ics.uci.edu/ml)

Nombre	Número de tuplas	Número de atributos	Año
Particle	3302	50	1987
Bridge	155	19	1988
Wine	178	13	1991
Automobile	205	20	1998
Commerce	1300	43	2008
Parkinsons Telemonitoring	5175	26	2009

desde la instancia, considerando que cada algoritmo utilizado obtuvo las mismas DFs. La tabla 4.2 muestra el detalle de los resultados por cada instancia y cada algoritmo.

La Figura 4.2 muestra los resultados promedio de la experimentación con los cuatro algoritmos e instancias que iban de las 100 a 500 tuplas. Se puede observar que el comportamiento de los algoritmos propuestos y su diferencia con TANE es poca, e incluso este último tiene mejor rendimiento al tratarse de pocas tuplas y pocos atributos.

La Figura 4.3 muestra los resultados promedio de la experimientación con los cuatro algoritmos e instancias que iban de las 500 a 2000 tuplas. Se puede observar que en esta experimentación se amplió el número de tuplas y atributos, lo que provoca que TANE baje su rendimiento condicionado especialmente por el alto número de atributos, llegando este último a no entregar resultados y a bloquearse en las pruebas con alrededor de 30 atributos. Por otro lado los algoritmos basados en hipergrafos, pese a que demoran un tiempo considerable, logran entregar los resultados.

Finalmente la Figura 4.4 muestra los resultados promedio de la experimientación con los cuatro algoritmos e instancias que iban de las 2000 a 5000 tuplas. Se puede observar que en esta experimentación, y similar al caso anterior, TANE se bloquea y no entrega resultados en las pruebas con alrededor de 30 atributos. Los algoritmos basados en hipergrafos aumentan su demora y tiempo de respuesta, pero logran entregar el resultado.

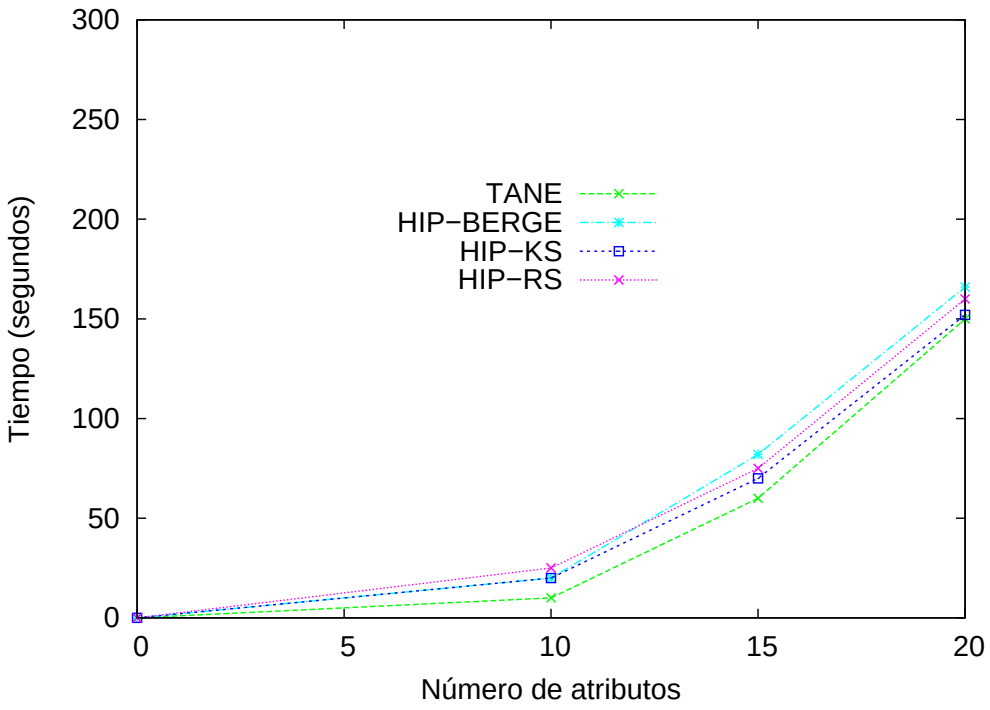

Figura 4.2: Búsqueda de DFs con los distintos algoritmos en instancias con hasta 200 tuplas y 20 atributos

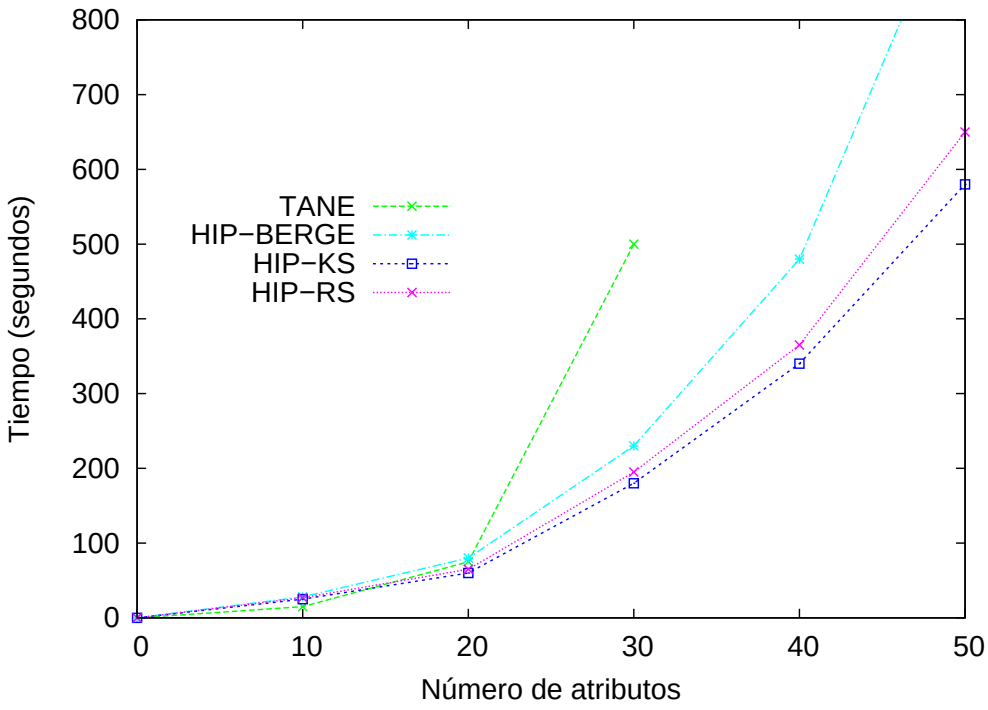

Figura 4.3: Búsqueda de DFs con los distintos algoritmos en instancias con hasta 2000 tuplas y 50 atributos

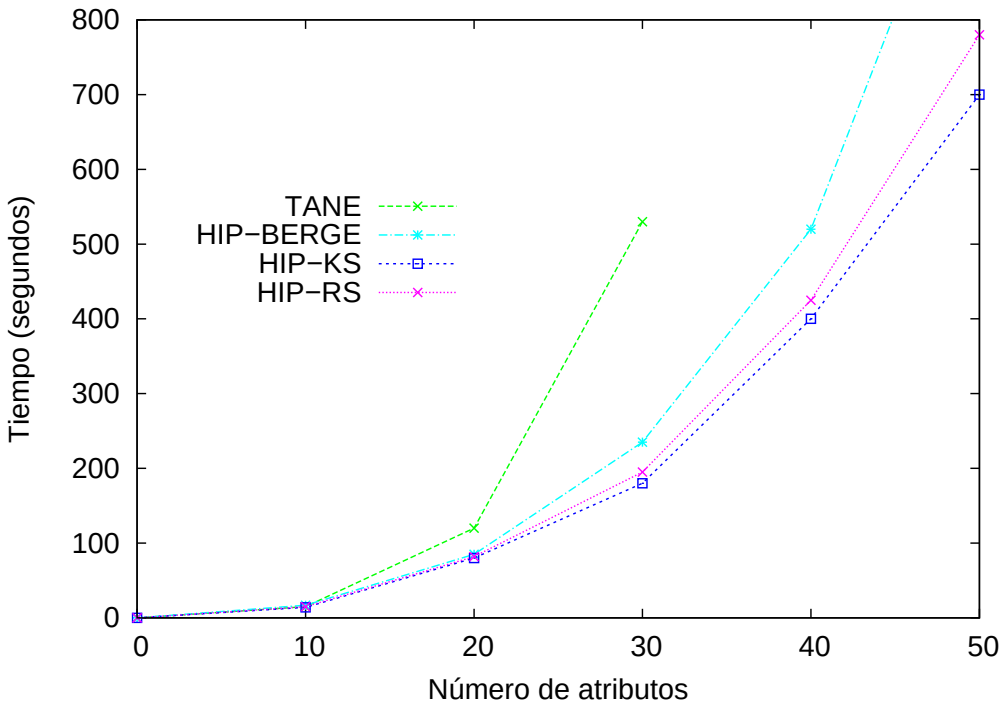

Figura 4.4: Búsqueda de DFs con los distintos algoritmos en instancias con hasta 5000 tuplas y 20 atributos

Tabla 4.2: Comportamiento de algoritmos de extracción de DFs

instancia	$\|r\|$	$\|R\|$	DF	TANE	HIP-BERGE	HIP-KS	HIP-RS
Particle	3302	50	4350	-	1100	585	670
Commerce	1300	43	1804	-	720	412	493
Parkinsons Telemonitoring	5175	26	2312	855	570	481	496
Automobile	205	20	494	65	74	64	70
Bridge	155	19	1250	40	53	45	47
Wine	178	13	61	1	1	1	1

Capítulo 5

Conclusiones

En este último capitulo se presentan las conclusiones finales y los trabajos futuros de esta tesis de investigación.

5.1 Conclusiones

La obtención de DFs desde un conjunto de datos es una importante etapa para las metodologías propuestas para obtener el modelo relacional desde sistemas heredados. La necesidad de contar con alguna herramienta automática para este fin es alta, considerando que las labores migración de sistemas heredados han ido en aumento.

En este trabajo de investigación se revisaron las principales herramientas de obtención de DFs, destacando la propuesta por Huthala et al. en 1999 llamada TANE, y que se presenta como la herramienta más conocida en este ámbito. Sin embargo, se demostró que la eficiencia de esta heramienta está directamente ligada al número de atributos que tiene la relación a analizar, lo que evidentemente dificulta la obtención de DFs sobre aquellas relaciones con sobre 30 atributos.

El estudio de dualidad de hipergrafos permitió desarrollar un nuevo método que permite obtener todas las DFs desde una relación, consistente en aplicar los operadores que calculan

las transversales minimales a hipergrafos formados por hiperaristas de refutaciones maximales, refutaciones que resultan de un análisis previo de la instancia de la relación.

A su vez, el resultado del estudio del estado del arte arrojó que para el cálculo del operador λ, es decir el cálculo de transversales minimales, existían algoritmos que destacaban por su operación en tiempo sub-exponencial. Por lo tanto, es factible realizar la obtención de las DFs de forma más eficiente a través de dualidad de hipergrafos. Los algoritmos considerados para este fin fueron el de Berge, Kavvadias y Stavropoulos y el de Murakami y Uno.

Las pruebas realizadas, con la implementación de los algoritmos para el cálculo de transversales minimales de Berge, Kavvadias-Stavropoulos y Murakami y Uno mostraron que este nuevo método responde bien cuando se trabaja con relaciones que tienen gran cantidad de atributos, mostrando diferencias de rendimiento con TANE sobre los 20 atributos, donde este último incluso no muestra resultados en un tiempo razonable cerca de los 30 atributos. Por otro lado, la debilidad del método propuesto, es que su rendimiento comienza a disminuir a medida que se aumenta el número de tuplas de la relación (situación mostrada en los gráficos de resultados), factor que está ligado al cálculo inicial de refutaciones que se debe realizar antes de las operaciones con hipergrafos.

Los resultados mostrados en las diferentes experimentaciones de esta tesis muestran que la transformación del problema de obtener las DFs desde una instancia de una relación se pudo llevar de forma eficiente al problema de dualidad de hipergrafos. La teoría de hipergrafos se presenta entonces como una potente herramienta que puede ser utilizada en muchas áreas de la ciencia. Los algoritmos que se han propuesto en el último tiempo presentan complejidades sub-exponenciales y sus implementaciones, pese a ser complejas, aseguran un tiempo de respuesta adecuado la mayoría de las veces.

Respecto a los algoritmos para el cálculo de transversales minimales, el que obtuvo mejor rendimiento fue el propuesto por Kavvadias y Stravropoulos (algoritmo KS). Sin embargo, la diferencia con el propuesto por Murakami y Uno (en este tipo de problemas) es baja, destacando

que este último y de acuerdo a lo expuesto por sus autores está enfocado a obtener un mejor rendimiento cuando los hipergrafos poseen un alto número de hiperaristas.

5.2 Trabajos futuros

Un posible buen panorama es la experimentación y comparación de la actual herramienta con variantes de TANE que han surgido en el último tiempo. La mayor parte de estas nuevas variantes se enfocan a mejorar las técnicas de pruning en el algoritmo.

También puede resultar favorable incorporación a la herramienta construida de nuevos algoritmos para el cálculo de transversales minimales, tal es el caso por ejemplo, del algoritmo de Fredman y Khachiyan. La idea es poder mejorar la eficiencia del algoritmos incluyendo algunos que pudiesen ser más eficientes que los estudiados en esta tesis.

Los sistemas de gestión de bases de datos actuales presentan gran potencialidad operacionalmente, por lo que puediese ser favorable la incorporación a esta nuestra herramienta conexión directa a un motor de base de datos, de modo de no hacer la operación directamente sobre archivos planos (que implica muchas veces su carga completa en memoria), sino que poder utilizar el potencial de consultas que posee dicho motor.

Para el caso no favorable del método, es decir cuando el número de tuplas es elevado, se podría utilizar mejores técnicas de codificación y compresión. También pudiese ser factible adicionar mecanismos para entregar DFs con algún grado de error, pero reduciendo la cantidad de tuplas que analizar.

Referencias

[And94] M. Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. *Entity-Relationship ApproachER'94 Business Modelling and Re-Engineering*, pages 403–419, 1994.

[Arm74] W.W. Armstrong. Dependency structures of data base relationships. *Information processing*, 74:580–583, 1974.

[Bai04] J. Baixeries. A formal concept analysis framework to mine functional dependencies. In *Workshop on mathematical mehtods for learning, Como, Italy*, 2004.

[BEGK00] E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. *Parallel Process Letter*, 10(4):253–266, 2000.

[Ber89] C. Berge. Hypergraphs. *volume 45 of North-Holland Mathematical Library*, 1989.

[BM98] C.L. Blake and C.J. Merz. Uci repository of machine learning databases, 1998, 1998.

[CDGL01] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification constraints and functional dependencies in description logics. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 155–160, 2001.

[EMG08] T. Eiter, K. Makino, and G. Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.

[ENC+02] R. Elmasri, S.B. Navathe, V.C. Castillo, B.G. Espiga, and G.Z. Pérez. *Fundamentos de sistemas de bases de datos*. Addison-Wesley, 2002.

[FFGZ03] S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Repairs and consistent answers for xml data with functional dependencies. *Database and XML Technologies*, pages 238–253, 2003.

[FK96] M.L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21(3):618–628, 1996.

[FPSSU96] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in knowledge discovery and data mining*. 1996.

[FS99] P.A. Flach and I. Savnik. Database dependency discovery: a machine learning approach. *AI communications*, 12(3):139–160, 1999.

[FSG11] J. Fuentes, P. Sáez, and G. Gutiérrez. Propuesta de un método para la obtención de dependencias funcionales basado en dualidad de hipergrafos. *XVI Jornadas del Software y las Bases de Datos, JISBD , La Coruña, España*, 2011.

[HKPT98] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 392–401. IEEE, 1998.

[HKPT99] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100, 1999.

[HRCH07] J. Henrard, D. Roland, A. Cleve, and J.L. Hainaut. An industrial experience report on legacy data-intensive system migration. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 473–476. IEEE, 2007.

[KBEG06] L. Khachiyan, E. Boros, K. Elbassioni, and V. Gurvich. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete applied mathematics*, 154(16):2350–2372, 2006.

[KS05] D.J. Kavvadias and E.C. Stavropoulos. An efficient algorithm for the transversal hypergraph generation. *Journal of Graph Algorithms and Applications*, 9(2):239–264, 2005.

[Lin08] C.Y. Lin. Migrating to relational systems: Problems, methods, and strategies. *Contemporary Management Research*, 4(4):369–380, 2008.

[LPL00] S. Lopes, J.M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *Advances in Database Technology — EDBT 2000*, volume 1777, pages 350–364. Springer Berlin Heidelberg, 2000.

[LPL02] S. Lopes, J.M. Petit, and L. Lakhal. Functional and approximate dependency mining: database and fca points of view. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2):93–114, 2002.

[Mai83] D. Maier. *The theory of relational databases*, volume 13. Computer science press, 1983.

[MI98] K. Makino and T. Ibaraki. A fast and simple algorithm for identifying 2-monotonic positive boolean functions. *Journal of Algorithms*, 26(2):291–305, 1998.

[MR86] H. Mannila and K. Raiha. Design by example: An application of armstrong relations. *Journal of computer and system sciences*, 33(2):126–141, 1986.

[MU11] K. Murakami and T. Uno. Efficient algorithms for dualizing large-scale hypergraphs. *Arxiv preprint arXiv:1102.3813*, 2011.

[NC01a] N. Novelli and R. Cicchetti. Fun: an efficient algorithm for mining functional and embedded dependencies. In *Database Theory — ICDT 2001*, volume 1973, pages 189–203. Springer Berlin Heidelberg, 2001.

[NC01b] N. Novelli and R. Cicchetti. Functional and embedded dependency inference: a data mining point of view. *Information Systems*, 26(7):477–506, 2001.

[Pol08] A. Polyméris. Stability of two player game structures. *Discrete Applied Mathematics*, 156(14):2636–2646, 2008.

[RG00] R. Ramakrishnan and J. Gehrke. *Database management systems*. Osborne/McGraw-Hill, 2000.

[RG02] R. Ramakrishnan and J. Gehrke. *Database management systems*. McGraw-Hill, Inc., 2002.

[Sne95] H.M. Sneed. Planning the reengineering of legacy systems. *Software, IEEE*, 12(1):24–34, 1995.

[TZ04] H.B.K. Tan and Y. Zhao. Automated elicitation of functional dependencies from source codes of database transactions. *Information and Software Technology*, 46(2):109–117, 2004.

[VCG10] F. Villagrán, A. Caro, and G. Gutiérrez. Definición de un marco de trabajo para la obtención de un modelo de base de datos relacional desde sistemas

heredados. Antofagasta, Chile, 2010. Workshop de Tesistas, Jornadas Chilenas de Computación.

[WGR01] C. Wyss, C. Giannella, and E. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *Data Warehousing and Knowledge Discovery*, volume 2114, pages 101–110. Springer Berlin Heidelberg, 2001.

[YH08] H. Yao and H.J. Hamilton. Mining functional dependencies from data. *Data Mining and Knowledge Discovery*, 16(2):197–219, 2008.

[YHB02] H. Yao, H.J. Hamilton, and C.J. Butz. Fd_mine: discovering functional dependencies in a database using equivalences. In *ICDM*, pages 729–732. IEEE Computer Society, 2002.

Anexos

Anexo A

Instrucciones de uso

Compilación del programa.

Para realizar la compilación de la herramienta debe considerar las siguientes restricciones:

- Sistema Operativo Linux
- Compilador GNU de C *gcc*

El dentro de la carpeta del proyecto ubicar el archivo HyperDF.c e ingresar a través del terminal el siguiente comando:

```
$gcc -o HyperDF HyperDF.c
```

Lo cual generará el archivo ejecutable HyperDF.

Ejecución del programa.

Para la ejecución de la herramienta debe seguir los siguientes pasos:

1. Ubicar en la carpeta del proyecto el conjunto de datos a analizar, el cual debe ser un archivo de texto plano y en donde cada línea corresponde a una tupla y los atributos se encuentran separados por coma. Por ejemplo:

   ```
   842302,M,Alfred,10.38,122.8,F-12
   842322,F,Scott,12.1,72.2;E-14
   ```

2. Ingresar el comando:

```
$./HyperDF -a TipoAlgoritmo -d ConjuntoDeDatos
```

Donde:

- *TipoAlgoritmo* puede ser: BERGE, KAVVADIAS o MURAKAMI (respetando las mayúsculas)
- *ConjuntoDeDatos* es el nombre del archivo a analizar (considere también su extensión)

Por ejemplo:

```
$./HyperDF -a KAVVADIAS -d dbcommerce.dat
```

Lo que producirá como salida:

```
Analisis: Refutaciones maximales encontradas
Ejecutando algoritmo de KAVVADIAS
UN TOTAL DE 112 DFs encontradas
El tiempo de demora es: 3483 miliseg
DFs encontradas:
A->B,C
A,D->F
B->E
...
...
```

Anexo B

Implementación (principales métodos)

```
/*Metodo que codifica la relacion*/
 Static Int** codificaRelacion(char** relacion) {
      LIST v;= creaList();
      HASHMAP atr1=reaHash();
        int** codificada = (int **) malloc(sizeof(int*tam_relacion));
        int j = 0;
        while (j < atributos) {
            int codigo = 0;
            for (int i = 0; i < tam_relacion; i++) {
                if (!containsKey(atr1, relacion[i][j])) {
                    put(atr1, relacion[i][j], ++codigo);
                }
                codificada[i][j] = get(atr1, relacion[i][j]);
            }
            addElement(v, atr1);
            j++;
```

```
    }
    //liberar memoria hashmp
    v.removeAllElements();
    return codificada;
}


/*metodo que agrega refutaciones*/
    Static Void addRefutacion(REFUTACION *ref, ARRAYLIST *refConsecuente) {
    if (refConsecuente->size == 0) {
         add(refConsecuente,ref);
         return;
     }
    int i = 0;
     while (i < refConsecuente.size()) {
            if (isSubSet(ref, refConsecuente.get(i)))
             return;
        else {
            if (isSubSet(get(refConsecuente, i),refutacion)) {
                   remove(refConsecuente,i);
                    add(refConsecuente, refutacion);
                    return;
            } else {
                ++i;
            }
        }
```

```
        }
        add(refConsecuente, refutacion);
    }

/*Algoritmo de Kavvadias y Stavropoulos*/
/* Verificacion de simplicidad de clausulas*/
Static Void verifica_simplicidad(setnode *instancia){
   setnode *ip = instancia;
   setnode *jp;
   boolean error = false;
   while (ip != NULL) {
     jp = ip->next;
     while (jp != NULL) {
       if (P_setequal(jp->vari, ip->vari)) {
printf("Alg: clausula duplicada encontrada\n");
error = true;
       } else {
if (P_subset(jp->vari, ip->vari) || P_subset(ip->vari, jp->vari)) {
   error = true;
   printf("Alg: clausula subconjunto encontrada\n");
}
      }
      jp = jp->next;
    }
    ip = ip->next;
  }
  if (!error) {
```

```
    if (!suppressoutput)
      printf("Alg: instancia ok (simplicidad verificada)\n");
  }
}

/*generacion de transversales*/
Local boolean genera_sig_transversal(LINK){
  variable SET, SET1, TEMP;
  LINK->tp = NULL;
  LINK->endtp = NULL;
  LINK->maxko2 = (LINK->limit + 1) >> 1;
  switch (LINK->action) {

  case 1:
    if (LINK->k > LINK->limit)
      LINK->found_tran = false;
    else {
      LINK->found_tran = true;
      if (LINK->k == LINK->limit) {
LINK->local_action = 1;
add_monogamic(&LINK->cl, LINK);
      } else {
LINK->local_action = 2;
  }
    }
    break;
```

```
  case 2:
    if (LINK->k > LINK->limit)
      LINK->found_tran = false;
    else {
      LINK->found_tran = true;
      LINK->local_action = 2;
    }
    break;

  case 5:
    if (LINK->k >= LINK->limit) {
      LINK->local_action = 4;
      LINK->found_tran = false;
      while (!LINK->found_tran && LINK->trail != NULL) {
if (*P_setint(SET, LINK->trail->vari, LINK->cl->vari) != 0 &&
    *P_setint(SET1, LINK->trail->vari, LINK->maintoken) == 0) {
   LINK->found_tran = true;
   mynew(&LINK->tp);   /* la nueva transversal comienza aqui*/
   LINK->tp->next = NULL;
   LINK->tp->node = LINK->trail;
   assign_new_tvalue(&LINK->tp->token, LINK->cl->vari);
   add_monogamic(&LINK->cl, LINK);
   LINK->endtp = LINK->tp;
      /* Aqui termina la nueva transversal */
}
```

```
LINK->trail = LINK->trail->next;
    }
    }  /* si k>=limit */
    else {
       if (LINK->k == ((LINK->prvpwr >> 1) | LINK->maxko2) ||
   LINK->k == LINK->limit >> 1){
LINK->found_tran = true;
LINK->local_action = 3;
add_monogamic(&LINK->cl, LINK);
LINK->prvpwr = LINK->k;
      } else {
LINK->found_tran = true;
LINK->local_action = 2;
      }
    }
    break;
  }

  if (LINK->found_tran) {
    /*  crear copia de la lista */
    LINK->tempaf = LINK->af;
    P_expset(LINK->newmaintoken, 0L);
    while (LINK->tempaf != NULL) {
      mynew(&LINK->temptp);
      LINK->temptp->node = LINK->tempaf->node;
      LINK->temptp->next = LINK->tp;
```

```
      LINK->tp = LINK->temptp;
      if (LINK->local_action == 4)
assign_new_tvalue(&LINK->temptp->token, omega);
      else {
if (LINK->local_action == 1 &&
    LINK->piece->node->vari == LINK->tp->node->vari) {
   P_setint(TEMP, LINK->tempaf->token->tokenset, LINK->cl->vari);
   assign_new_tvalue(&LINK->temptp->token, TEMP);
} else
   eq_tokens(&LINK->temptp->token, &LINK->tempaf->token);
       }
       if (LINK->local_action != 2 && LINK->local_action != 4)
P_setunion(LINK->newmaintoken, LINK->newmaintoken,
   LINK->temptp->token->tokenset);
       if (LINK->endtp == NULL)
LINK->endtp = LINK->tp;
       LINK->tempaf = LINK->tempaf->next;
     }

    LINK->temk = LINK->k;
    LINK->tempsf0 = LINK->sf0;
    LINK->tempsf1 = LINK->sf1;
    while (LINK->tempsf0 != NULL) {
      mynew(&LINK->temptp);
      if ((LINK->temk & 1) == 0 && LINK->k < LINK->limit) {
LINK->temptp->node = LINK->tempsf0->node;
```

```
if (LINK->local_action == 3 && LINK->k == LINK->prvpwr) {
  P_setint(TEMP, LINK->tempsf0->token->tokenset, LINK->cl->vari);
  assign_new_tvalue(&LINK->temptp->token, TEMP);
} else
  eq_tokens(&LINK->temptp->token, &LINK->tempsf0->token);
if (LINK->local_action != 2)
  P_setunion(LINK->newmaintoken, LINK->newmaintoken,
     LINK->temptp->token->tokenset);
      } else {
LINK->temptp->node = LINK->tempsf1->node;
if (LINK->local_action == 4)
  assign_new_tvalue(&LINK->temptp->token, omega);
else
  eq_tokens(&LINK->temptp->token, &LINK->tempsf1->token);
if (LINK->local_action != 2 && LINK->local_action != 4)
  P_setunion(LINK->newmaintoken, LINK->newmaintoken,
     LINK->temptp->token->tokenset);
      }
      LINK->temptp->next = LINK->tp;
      LINK->tp = LINK->temptp;
      if (LINK->endtp == NULL)
LINK->endtp = LINK->tp;
      LINK->tempsf0 = LINK->tempsf0->next;
      LINK->tempsf1 = LINK->tempsf1->next;
      LINK->temk >>= 1;
    }
```

```
}

if (LINK->local_action == 2)
  P_setcpy(LINK->newmaintoken, LINK->maintoken);

if (!(LINK->found_tran && LINK->local_action == 4)) {
return LINK->found_tran;
}
LINK->tempi = monogamic->next;
while (LINK->tempi != NULL) {
  if (*P_setint(SET, LINK->tempi->vari, LINK->endtp->node->vari) != 0) {
    remove_monogamic(&LINK->tempi, LINK);
    continue;
  }
  LINK->temptp = LINK->tp;
  while (LINK->temptp != NULL) {
    if (*P_setint(SET1, LINK->tempi->vari, LINK->temptp->node->vari) != 0)
      P_setint(LINK->temptp->token->tokenset,
     LINK->temptp->token->tokenset, LINK->tempi->vari);
    LINK->temptp = LINK->temptp->next;
  }
  LINK->tempi = LINK->tempi->next;
}
LINK->temptp = LINK->tp;
while (LINK->temptp != NULL) {
  P_setunion(LINK->newmaintoken, LINK->newmaintoken,
```

```
       LINK->temptp->token->tokenset);
    LINK->temptp = LINK->temptp->next;
  }
  return LINK->found_tran;
}



/* Metodo SHD utilizando la propuesta de Murakami y Uno */
void SHDC (PROBLEM *PP, QUEUE_INT prv, VEC_ID tt){
  QUEUE *CAND = &PP->itemcand;
  QUEUE_INT u=0, *x, flag, f=PP->problem&SHD_DFS, item;
  QUEUE_ID jt, js=PP->itemjump.s, cs=CAND->s;
  PP->II.iters++;
  CAND->s = CAND->t; PP->itemjump.s = PP->itemjump.t;
  if ( (PP->problem&(SHD_DFS+SHD_PRUNE)) == SHD_PRUNE )
   SHDC_crit_check (PP, tt, prv);
  ARY_REALLOCZ (*CAND, (PP->FF.clms-PP->FF.v[tt].t) + CAND->t, EXIT);
PP->II.solutions += PP->FF.clms;
  x = PP->FF.v[tt].v;
  FLOOP (item, 0, PP->FF.clms){
    if ( item == *x ){ x++; continue; }
    if ( PP->itemchr[item] == 0 ){
      ARY_INS (*CAND, item);
      if ( f ) PP->itemchr[item] = 4;
    } else if ( PP->itemchr[item] == 1 ) PP->itemchr[item] = 0;
    else if ( (PP->problem&(SHD_DFS+SHD_PRUNE))
```

```
     != SHD_PRUNE ) PP->II.iters3++;
  }

  if ( f ) tt = PP->FF.t-1;
  for ( jt=CAND->t ; jt > CAND->s ; ){
        /* actualizacion conjunto critico */
    item = CAND->v[--jt];
PP->II.solutions2 += PP->OQ[item].t;  // counter
    if ((flag=SHDC_update (PP,item,&u,&PP->OQ[item].t,prv))
     >=PP->FF.t && (PP->problem&SHD_PRUNE)){
      PP->itemchr[item] = 4; ARY_INS (PP->itemjump, item);
    } else if ( flag > tt ){ PP->itemchr[item] = 0;
    } else {
      ARY_INS (PP->II.itemset, item); PP->itemchr[item] = 4;
      if ( u < PP->FF.t ) SHDC (PP, item, u); // recursion
      else{ /* llamada al metodo para mostrar la solucion*/
        ITEMSET_output_itemset (&PP->II, NULL, 0);
      }
      PP->itemchr[item] = 0; PP->II.itemset.t--;
    }
    SHDC_recov (PP, item, prv);
  }

  MQUE_SLOOP (PP->itemjump, x) PP->itemchr[*x] = 0;
  CAND->t = CAND->s; CAND->s = cs;
  PP->itemjump.t = PP->itemjump.s; PP->itemjump.s = js;
```

```
}

/* Metodo correspondiente a la version DFS */
int SHDC_update (PROBLEM *PP, QUEUE_INT e,
QUEUE_INT *u, QUEUE_INT *h, QUEUE_INT prv){
  QUEUE_INT *t, z, b=PP->II.itemset.t-1;

  if ( (PP->problem&(SHD_DFS+SHD_PRUNE)) != SHD_PRUNE )
   PP->II.iters2++;
  *u = PP->FF.t;
  MQUE_FLOOP (PP->II.itemset, t) PP->itemary[*t] = PP->FF.t;
  MQUE_FLOOP (PP->OQ[e], t){
    if ( PP->vecflag[*t] < b ) continue;
    if ( PP->vecflag[*t] == b ){
      if ( PP->vecmark[*t] == PP->FF.clms ) PP->vecmark[*t] = prv;
      ENMIN (PP->itemary[PP->vecmark[*t]], *t);
    } else ENMIN (*u, *t);
    PP->vecflag[*t]++;
  }
  z = 0; MQUE_FLOOP (PP->II.itemset, t) ENMAX (z, PP->itemary[*t]);
  return (z);
}

/* actualizacion del conjunto critico ( version DFS) */
void SHDC_recov (PROBLEM *PP, QUEUE_INT e, QUEUE_INT prv){
  QUEUE_INT *t, b=PP->II.itemset.t;
```

```
  MQUE_FLOOP (PP->OQ[e], t){
    if ( PP->vecflag[*t] < b ) continue;
    PP->vecflag[*t]--;
    if ( PP->vecmark[*t] == prv ) PP->vecmark[*t] = PP->FF.clms;
  }
}
```