RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Generación de conjuntos de ítems y reglas de asociación.

José E. Medina P., José Hdez. Palancar, Raudel Hdez., Airel Pérez S., Abdel Hechavarría D., Ricardo Glez. Gazapo

RT_003 **Mayo 2007**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
Ciudad de La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

RNPS No. 2143
ISSN 2072-6260

SERIE GRIS

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

REPORTE TÉCNICO
Minería de Datos

Generación de conjuntos de ítems y reglas de asociación

José E. Medina P., José Hdez. Palancar, Raudel Hdez., Airel Pérez S., Abdel Hechavarría D., Ricardo Glez. Gazapo

RT 003 **Mayo 2007**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
Ciudad de La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

Generación de conjuntos de ítems y reglas de asociación

José E. Medina Pagola, José Hernández Palancar, Raudel Hernández León, Airel Pérez Suárez, Abdel Hechavarría Díaz, Ricardo González Gazapo

Dpto. Minería de Datos, Centro de Aplicaciones de Tecnología de Avanzada (CENATAV)
{ jmedina, jpalancar, rhernandez, asuarez, ahechavarria, rgazapo } @cenatav.co.cu

RT_003 CENATAV
Fecha del camera ready: mayo de 2007

Resumen. En este trabajo se realiza un estudio del estado del arte de los métodos de generación de *itemsets* frecuentes y el descubrimiento de reglas de asociación, aplicables a bases de datos textuales como, por ejemplo, las bases noticiosas. Considerando que la generación de *itemsets* frecuentes es el paso más costoso de estos algoritmos, en este trabajo se han abordado algunas de las técnicas más relevantes, incluyendo algunos métodos paralelos. Además, se ha incluido una propuesta realizada por algunos de los miembros del colectivo de autores, denominada CBMine, el cual ha mostrado resultados significativos. Sobre este método, se ha propuesto también una versión paralela denominada ParCBMine.

Palabras claves: Conjunto de ítems frecuentes, reglas de asociación, minería de datos

Abstract. In this paper we make a study of the state of the art of itemset generation methods and the discovery of association rules, applicable to textual databases as, for instance, news sources. Considering the frequent itemset generation is the most expensive step of these algorithms, in this work we analyze some relevant techniques, including several parallel methods. Besides, it has been included a proposal developed by several of the authors of this work, called CBMine, which has shown significant results. About this method, it has been also proposed a parallel version, called ParCBMine.

Keywords: Frequent itemsets, association rules, data mining

Contenido

Introducción

El descubrimiento de asociaciones es una de las técnicas de más reciente desarrollo dentro de la Minería de Datos. Los primeros trabajos se observan casi a inicios de la década pasada, a partir de un trabajo de Rakesh Agrawal, Tomasz Imielinski y Arun Swami de 1993 [Agrawal, 1993]. En el caso de asociaciones en textos, el primer trabajo conocido es el presentado por Ronen Feldman y Haym Hirsh en 1995 [Feldman, 1995]. En los últimos años se ha dado una mayor atención a la generación automática de reglas de asociación; sin embargo, en datos no estructurados, como son las noticias, artículos científicos y cualquier otro tipo de documento, no son muchos los trabajos encontrados. También se han observado en años recientes propuestas e implementaciones de algoritmos paralelos sobre estas técnicas, lo cual se corresponde con la complejidad computacional y el gran volumen de datos a procesar [Joshi, 2000].

El descubrimiento de Reglas de Asociación ha resultado un instrumento de minería útil en muy diversas aplicaciones [Fayyad, 1996], [Cheung, 1996a], [Chen, 1996], particularmente en las ventas minoristas y el mercadeo directo, alcanzando cada vez mayor importancia en aplicaciones sobre colecciones de documentos [Feldman, 1998], [Holt, 2001].

Muy diversos han sido los algoritmos propuestos, siendo Apriori el de mayor impacto [Agrawal, 1994], ya que este estableció una concepción general: la generación y prueba de candidatos para reducir la explosión combinatoria, la cual ha servido como base para el desarrollo de nuevos métodos. No obstante, esta temática aún presenta muchas preguntas abiertas, motivado sobre todo por los grandes volúmenes de información y características de los soportes de datos existentes.

Este trabajo ha sido estructurado en ocho secciones, incluyendo la introducción y conclusión. En la sección 2 se presentan las definiciones formales básicas. En la sección 3 se ofrecen algunas características y clasificaciones de los métodos de generación de *itemsets*. En la sección 4 se describen los principales métodos secuenciales de generación de itemsets; en esta sección resalta el algoritmo CBMine propuesto por varios de los autores de este reporte técnico. En la sección 5 se realiza una evaluación de los métodos paralelos de generación de *itemsets*, proponiéndose el algoritmo ParCBMine, una versión paralela del método anterior. En la sección 6 se ofrecen algunas características y clasificaciones de las reglas de asociación y sus medidas de interés. En la sección 7 se describen los principales métodos de generación de las reglas de asociación.

1. Definiciones formales básicas

Los algoritmos de reglas de asociación están orientados al descubrimiento de dependencias o asociaciones entre conjuntos de atributos, variables o (como suele denominarse en este campo) ítems. Los primeros tipos de aplicaciones en el que se comprobó la utilidad de este tipo de regla fueron los sistemas que procesan los datos de ventas obtenidos mediante lectores de código de barra, conocidos como Datos de la Canasta (Basket Data) [Agrawal 1993]. Un registro o tupla en estos datos está compuesto por la fecha de la transacción y los ítems comprados en ella. Un ejemplo de regla que pudiera descubrirse en tales tipos de datos es la siguiente:

El 98% de los clientes que compran neumáticos y accesorios de autos también adquieren sistemas de alarmas.

Realmente, estas técnicas pueden ser aplicadas a cualquier tipo de dato, existiendo diversos trabajos orientados a los diferentes tipos, e incluso a la mezcla de ellos. En el caso de aplicarse a noticias, o datos textuales en general, un registro o tupla sería cada documento a procesar, mientras que los ítems serían los términos significativos presentes en los documentos. En este caso, la base de datos (o de noticias) puede considerarse con similar formato a los de la forma clásica; o sea, como una base de transacciones compuestas de conjuntos de ítems. A continuación se expondrán algunas definiciones aplicables a estos tipos de datos.

Definición. *Conjunto de ítems o itemset* – Un conjunto de ítems, o *itemset*, es aquel cuyos elementos son atributos, variables o campos de cualquier base de datos.

Aunque no es un requisito esencial, los ítems que se considerarán en lo sucesivo son aquellos que toman valores bivalentes. Un ejemplo del empleo de esta notación es su aplicación a bases de datos transaccionales. Si una base de datos D está compuesta de un conjunto de ítems $I = \{i_1, i_2, \ldots i_n\}$, entonces cada transacción de D puede identificarse por el itemset T, $T \subseteq I$, considerando en T solamente los ítems presentes en la transacción.

El tamaño de un itemset, o conjunto de ítems, está dado por la cantidad de ítems contenidos en el itemset. Con la expresión $|T|$ se indicará en lo sucesivo al tamaño o cardinalidad del itemset T.

Definición. *k-itemset* – Un *itemset* X es un *k-itemset* si $|X| = k$; o sea, si X es un conjunto de k ítems.

Definición. *Regla de Asociación* – Sea $I = \{i_1, i_2, \ldots, i_n\}$ el conjunto de ítems de una base de datos. Una regla de asociación (RA) es una implicación de la forma $X \Rightarrow Y$, donde $X \subset I$, $Y \subset I$, y $X \cap Y = \varnothing$.

Usualmente, se dice que una transacción T satisface una regla de asociación $X \Rightarrow Y$, si $X \cup Y \subseteq T$.

Definición. *Cubrimiento (transaccional) de un itemset* – El cubrimiento (transaccional) de un *itemset* X en una base de datos D está formado por el siguiente conjunto:

$$D(X) = \{T \in D \mid X \subseteq T\}. \quad (1)$$

Definición. *Soporte de un itemset o conjunto de ítems* – El soporte (*support*) de un *itemset* o conjunto de ítems X en una base de datos D está dado por la siguiente expresión:

$$Sup(X) = \frac{|D(X)|}{|D|}. \quad (2)$$

En la definición anterior $|D|$ y $|D(X)|$ representan los tamaños o cardinalidades de esos conjuntos.

Definición. *Conjunto de ítems (itemset) frecuente o extenso* – Un conjunto de ítems (*itemset*) frecuente (*frequent itemset*) o extenso (*large itemset*) es aquel que ocurre con un soporte no menor que un mínimo dado (*minsup*).

Definición. *Soporte de una Regla de Asociación* – El soporte (*support*) de una regla de asociación $X \Rightarrow Y$ en una base de datos D está dado por la siguiente expresión:

$$Sup(X \Rightarrow Y) = Sup(X \cup Y) = \frac{|D(X \cup Y)|}{|D|}. \quad (3)$$

En la definición anterior se emplean similares notaciones que las utilizadas en la del soporte de un itemset. Usualmente las RA pueden encontrarse en una base de datos con mayor o menor soporte, el problema de la minería de RA es encontrar todas las reglas con un soporte mayor o igual que un mínimo especificado por el usuario. Por ejemplo, si la regla dada al inicio satisface un *minsup* = 0.45, entonces debe cumplirse que la cantidad de transacciones que verifican simultáneamente la compra de neumáticos, accesorios y sistemas de alarma es superior o igual al 45% del total de transacciones.

Además del descubrimiento de las RA con un soporte mínimo, en las diferentes aplicaciones observadas también suelen exigirse otras medidas de calidad. De todas ellas, la medida con mayor aceptación es la de confianza de la regla. En un epígrafe posterior se analizarán otras medidas de calidad de las RA.

Definición. *Confianza de una regla de asociación* – La confianza (*confidence*) de una regla de asociación $X \Rightarrow Y$ está dada por la siguiente expresión:

$$Conf(X \Rightarrow Y) = \frac{Sup(X \Rightarrow Y)}{Sup(X)} = \frac{|D(X \cup Y)|}{|D(X)|}. \quad (4)$$

Al igual que ocurre con el soporte, las RA pueden encontrarse con mayor o menor confianza, presentándose como problema el descubrimiento de reglas con un mínimo especificado por el usuario (*minconf*). En el caso del ejemplo del inicio, esa regla se satisface con una confianza de 0.98.

El procedimiento general para el descubrimiento de las RA ha sido planteado en dos pasos, siendo estos los siguientes [Agrawal, 1994]:

- La identificación de todos los conjuntos de ítems frecuentes, dado un *minsup*, con sus respectivos soportes.
- La generación de las RA que satisfagan las medidas de calidad indicadas como, por ejemplo, la confianza mínima (*minconf*).

Dado que el costo computacional del primer paso es mucho mayor que el del segundo, en este trabajo se aborda con mayor detenimiento el primero. En las dos siguientes secciones se analiza en detalles la identificación de los itemsets.

2. Características de los métodos de generación de itemsets

Los métodos de generación de los itemsets se diferencian por las estrategias de recorrido en los espacios de estado y las formas de representación de esos conjuntos de ítems. A continuación se analizarán estos dos aspectos.

2.1. Recorridos en el espacio de itemsets

Generar todos los *itemsets* frecuentes es una tarea ardua y de alto costo. En general, la evaluación de los subconjuntos de un conjunto de ítems que caracterizan a una base de datos se corresponde con la búsqueda en el espacio de estado asociado con el retículo (*Lattice*) que estos forman [Zaki, 1997a], [Hipp, 2000].

Para comprender mejor lo anterior, supóngase un conjunto universo de cuatro ítems: $I = \{i_1, i_2, i_3, i_4\}$. El retículo (omitiendo el conjunto vacío) que forman los subconjuntos de ese conjunto de ítems puede observarse en la fig. 1.

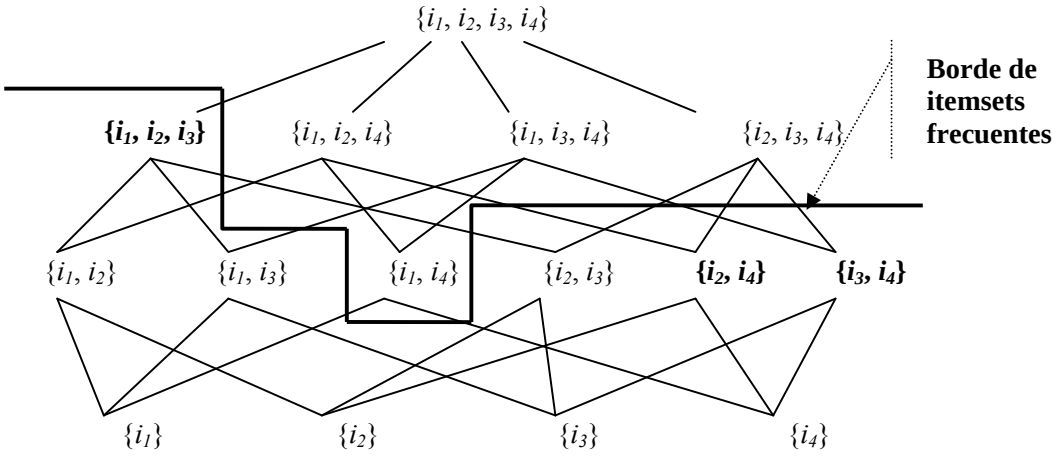


Fig. 1. Retículo de subconjuntos de $\{i_1, i_2, i_3, i_4\}$

No es difícil constatar que los soportes de los itemsets de un retículo disminuyen al recorrerlos desde los 1-itemsets hasta el itemset universo, verificándose la clausura descendente de sus soportes.

Propiedad. *Clausura descendente del soporte de itemset* (*downward closure of itemset support*) – El soporte de cualquier subconjunto de un *itemset* es mayor o igual que el soporte de ese *itemset*. Además, todo subconjunto de un itemset frecuente es frecuente, mientras que cualquier supraconjunto de un *itemset* no frecuente tampoco es frecuente.

Considerando la propiedad de la clausura descendente de los soportes, cualquiera sea el *minsup*, todo retículo posee un borde o frontera que separa los itemsets frecuentes de los no frecuentes (ver Fig. 1 y Fig. 2). Además, ese borde se asocia con los denominados itemsets maximales. Estos conceptos son ampliamente utilizados en las heurísticas de los algoritmos de búsqueda de los itemsets frecuentes. En la Fig. 1 (y Fig. 2) los itemsets maximales se han indicado en negrita.

Definición. *Conjunto de ítems (itemset) maximal* – Un conjunto de ítems (*itemset*) es maximal si no es un subconjunto propio de ningún otro *itemset* frecuente.

Otro tipo de itemset utilizado por algunos algoritmos es el conocido como conjunto de ítems cerrado (*closed itemset*) el cual depende fuertemente de las características de las transacciones de la base de datos.

Definición. *Conjunto de ítems (itemset) cerrado* – Un conjunto de ítems (*itemset*) X se considera cerrado en una base de datos D si no existe otro conjunto diferente Y tal que $X \subset Y$, siendo $D(X) = D(Y)$, verificándose además la expresión: $X = \bigcap D(X)$.

En general, un *itemset* cerrado es el de mayor cardinalidad en cualquier secuencia de igual soporte en el retículo. No es difícil comprender que un *itemset* maximal es al mismo tiempo cerrado; o sea, que el conjunto de los *itemsets* maximales es un subconjunto de los *itemsets* cerrados, cualquiera sea la base de datos.

Todos los algoritmos de generación de *itemset* establecen un método de recorrido en el retículo o espacio de itemsets. Sin embargo, como puede observarse, este representa un grafo y no un árbol, por lo que se precisa de un orden para su recorrido. Si se considera que existe un orden lexicográfico entre los ítems, y los *itemsets* se indican atendiendo a ese orden, entonces los *itemsets* pueden agruparse en clases de equivalencia según se muestra a continuación.

Definición. *Clase de Equivalencia de un itemset* – La clase de equivalencia de un *itemset* X, indicado por $E(X)$, está dado por el siguiente conjunto de *itemsets*:

$$E(X) = \left\{Y \mid Prefix_{k-1}(Y) = X \wedge |Y| = k\right\}, \quad \textbf{(5)}$$

siendo $Prefix_k(c)$ el prefijo de tamaño k de c, formado por sus k primeros ítems, según orden lexicográfico.

Otra notación conveniente para la lógica de algunos algoritmos, dada por otros autores [Zaki, 1997a], nombra a la clase E(X) como [X] y define como elementos del conjunto asociado a los ítems sufijos de los itemsets que componen la clase.

Considerando estas clases de equivalencia, el retículo de la Fig. 1 puede estructurarse en un árbol (o floresta) de búsqueda agrupando todos los itemsets de la misma clase de equivalencia e incluyéndolos como descendientes de los itemsets formados por sus prefijos comunes, según se muestra en la Fig. 2.

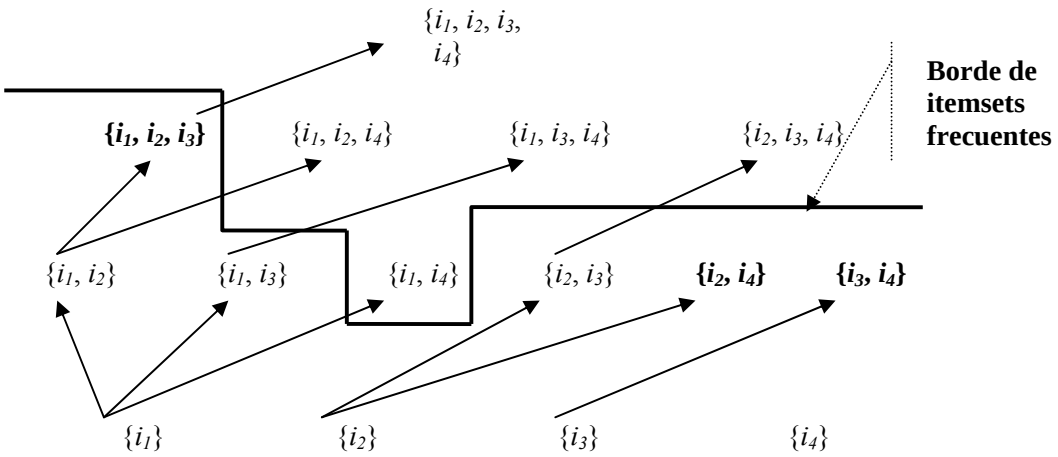


Fig. 2. Árbol de búsqueda de $\{i_1, i_2, i_3, i_4\}$

Considerando todo lo anterior, los métodos de generación de los itemsets frecuentes definen diferentes recorridos. Estos métodos pueden clasificarse atendiendo a la dirección y amplitud de los recorridos. Si se considera la dirección del recorrido, estos pueden clasificarse como:

- Métodos descendentes: Si el recorrido se realiza desde las raíces (1-*itemsets*) hasta los itemsets fronteras o maximales.
- Métodos ascendentes: Si el recorrido se realiza en el sentido opuesto, desde supraconjuntos de los itemsets fronteras (o maximales) hasta los itemsets fronteras (o maximales).

Si se considera la amplitud de los recorridos, estos pueden clasificarse como:

- Métodos a lo ancho: Si los itemsets de igual tamaño se evalúan antes de evaluar otro de diferente tamaño.
- Métodos en profundidad: Si se evalúan los *itemsets* por cada rama del árbol.

2.2. Representación de los itemsets

La forma de representar los *itemsets* es muy importante en el cálculo de los conjuntos frecuentes. Conceptualmente, las bases de datos son matrices bidimensionales, donde las filas representan las transacciones y las columnas los ítems existentes. Estas matrices pueden implementarse de cuatro formas diferentes, las cuáles se muestran en la fig. 3 [Shenoy, 2000].

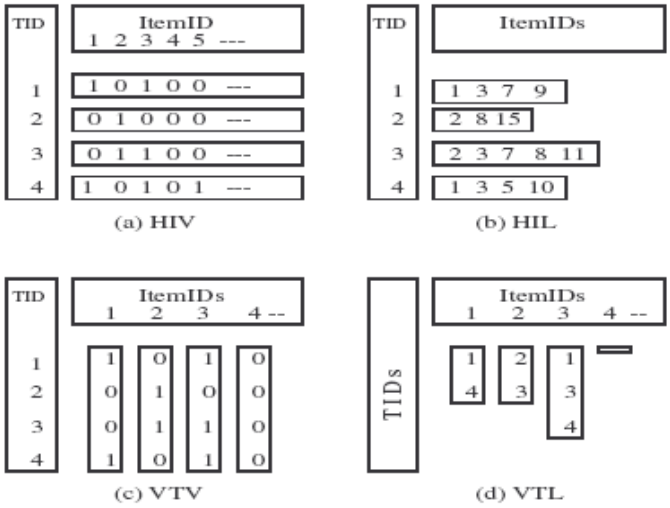


Fig. 3. Ejemplos de representaciones

Las cuatro formas representadas en la Fig. 3 son las siguientes:

- **Vector horizontal de ítems (HIV)** – Las transacciones se organizan como un conjunto de filas, donde cada fila almacena el identificador de la transacción (TID) y un vector de bits, para representar por cada ítem su presencia (con uno) o ausencia (con cero) en la transacción (Fig. 3a).
- **Lista horizontal de ítems (HIL)** – Esta representación es similar a HIV exceptuando que cada fila almacena una lista ordenada de identificadores de ítems (IID), representando sólo los ítems que están presentes en la transacción (Fig.3b).
- **Vector vertical de TID (VTV)** – Las transacciones se representan como un conjunto de columnas, asociadas con los ítems, donde por cada columna se almacena su IID y un vector de bits para representar la presencia ó ausencia del ítem en cada transacción (Fig. 3c). Se puede notar que VTV ocupa exactamente la misma cantidad de memoria que la representación HIV respecto a los vectores de bits, aunque por lo general se precisa menos memoria para almacenar los identificadores asociados.
- **Lista vertical de TID (VTL)** – Esta representación es similar a VTV exceptuando que cada columna almacena una lista ordenada de TID (tidList), representando sólo las transacciones en las cuáles el ítem está presente (Fig. 3d).

Virtualmente, la mayoría de los algoritmos existentes han optado por las representaciones basadas en listas, tanto horizontales como verticales, debido a que este formato ocupa menos memoria que las basadas en vectores de bits; particularmente en las bases de datos esparzas, ya que no tienen el sobrecosto de la representación explícita de las ausencias. De estas dos representaciones, y a pesar de ser la menos empleada, la forma VTL facilita el cálculo del soporte ya que este se logra determinando los tamaños de las tidList. Además, para el procesamiento en este formato de un *itemset Z* a partir de dos *itemsets* cualesquiera *X*, *Y*, tal que $X \cup Y = Z$, es suficiente obtener la lista intersección de los tidList de *X* y *Y*. Este principio ha

sido empleado por diversos autores con diferentes variantes [Zaki, 1997a], [Burdick, 2001], [Shenoy, 2000], [Gardarin, 1998], [Borgelt, 2003].

No obstante lo antes dicho, algunos autores han apostado por las representaciones binarias [Burdick, 2001], [Gardarin, 1998], [Lin, 2000], [Shenoy, 2000]. En particular, la representación vertical (VTV), por similares razones que la VTL, permite procesar los *itemsets* intersecando dos cualquiera que lo generen. En este caso, la intersección se logra mediante simples operaciones lógicas AND sobre los *bitmaps* o vectores binarios asociados, representados como listas de números enteros cuyos tamaños dependen de la palabra del procesador.

Tomar una decisión entre los formatos VTV y VTL no es una tarea trivial. Burdick, Calimlim y Gehrke analizaron estas dos formas de representación vertical al proponer el algoritmo MAFIA [Burdick, 2001]. En ese trabajo se precisa que la representación VTL es más costosa en términos de espacio si el soporte del ítem (o *itemset*) es mayor que 1/32, o aproximadamente el 3%. En la representación VTL, para expresar la presencia de un ítem en la tidList se precisa de toda una palabra (32 bits en la mayoría de las arquitecturas), mientras que en la VTV basta con un simple bit. Sin embargo, aún cuando la base de datos no sea esparza y los soportes promedios entre los ítems sea mayor que 1/32, la representación VTV no necesariamente es la mejor opción, ya que al descender en los árboles de búsqueda los soportes decrecen significativamente. Ante esta característica del proceso, Borgelt propuso (aunque basado sólo en una experimentación) utilizar tidList cuando el soporte sea menor que 1/8; o sea, 12.5% [Borgelt, 2003].

Un tipo alternativo a la representación VTV es la definición de diferentes métodos de compresión de los bitmaps asociados. Los propios autores de MAFIA observaron la conveniencia de métodos de compresión para contrarrestar las ineficiencias que se producen por el aumento de los bits ceros en bases de datos esparzas o en la medida que se desciende en los árboles de búsqueda ante soportes bajos. Ante tal situación, Burdick *et al.* propusieron eliminar los *bits* ceros de los *itemsets* tipo *Head* (ver epígrafe sobre MAFIA), aunque tal solución encarece la detección de los TID de las transacciones asociadas con los *itemsets* frecuentes descubiertos.

Una versión de este tipo es la propuesta de Shenoy *et al.* [Shenoy, 2000]. En este trabajo se propuso un algoritmo con recorrido a lo ancho, llamado VIPER, el cual utiliza representaciones de tidList y *bitmaps* verticales. Aunque VIPER utiliza una representación binaria comprimida en disco, sustituyendo varios enteros nulos consecutivos por un par de la forma <Posición, Cantidad de ceros>, cuando estos vectores son procesados en memoria se convierten “al vuelo” en tidList, no considerando las ventajas que ofrecen las operaciones booleanas.

Otra variante tipo VTV es la propuesta de Gardarin *et al.* [Gardarin 1998]. En ella se utiliza una estructura denominada *Hierarchical Bitmap*, la cual mantiene los *bitmaps* completos junto a otro bitmap comprimido al que se le asocia cada bit con cada entero del primer *bitmap*, asignándole cero si el entero es nulo y uno en cualquier otro caso. Cuando la operación AND entre los *bitmaps* comprimidos produce un bit uno, entonces se procesan los enteros asociados completándose la operación AND.

Otra forma de comprimir los *bitmaps* de la representación VTV es la propuesta realizada en el algoritmo CBMine, la cual se presenta en un epígrafe aparte.

3. Métodos secuenciales de generación de itemsets

Como ha podido observarse, la cantidad de estados asociados con el retículo de un conjunto de ítems puede ser grande. Particularmente, el costo computacional del recorrido de tal retículo en su forma exhaustiva es exponencial. Por esta razón, la mayoría de los métodos existentes centran su atención en estrategias que reduzcan el tiempo requerido para la generación de los *itemsets* representativos, proponiendo heurísticas para generar los itemsets potencialmente frecuentes, cerrados, fronteras o maximales.

Los métodos que se presentan en este epígrafe se han agrupado atendiendo, primeramente, a la dirección del procesamiento de los árboles de búsqueda y, como segundo criterio, a la amplitud del recorrido. Debe notarse que la mayoría de los algoritmos que se presentan clasifican como métodos descendentes. Por otra parte, según algunos autores – y también a nuestro parecer – los métodos descendentes secuenciales más referenciados son Apriori, FP-Growth y Eclat, y variaciones de estos [Bodon, 2005].

A continuación se expondrán algunos de los principales métodos y estrategias existentes con un esquema secuencial de ejecución. Debe señalarse que en el último sub-epígrafe se presenta el algoritmo CBMine, desarrollado como resultado de este proyecto.

3.1. Métodos descendentes con recorridos a lo ancho

A continuación se presentan algunos de los métodos descendentes con recorrido a lo ancho. El método paradigmático de esta clasificación es el algoritmo Apriori. Posteriormente, se presentan otro métodos descendentes con recorridos a lo ancho, todos del tipo Apriori, como son: Partition, tipo Multipas (M-Apriori, M-DHP), y tipo Hashing (DHP, IHP, MIHP). Además de estos algoritmos, como fue señalado anteriormente, en un epígrafe aparte se expondrá otro método descendente con recorrido a lo ancho – también del tipo Apriori – desarrollado por varios de los autores de este reporte, conocido como CBMine.

3.1.1. Método Apriori

Varios han sido los algoritmos propuestos para generar *itemsets* frecuentes. De todos ellos, el método Apriori ha sido uno de los de mayor impacto, quizás el más referenciado de todos [Agrawal, 1994]. La estrategia que este algoritmo sigue es del tipo descendente a lo ancho, siendo su concepción general la base para muchos de los algoritmos desarrollados posteriormente.

De forma general, la mayoría de los algoritmos tipo Apriori primero construyen un conjunto de *itemsets* candidatos basados en alguna heurística y, posteriormente, determinan el subconjunto que realmente contiene los *itemsets* frecuentes. Este proceso puede realizarse de forma repetitiva conociendo que los *itemsets* frecuentes obtenidos en una iteración servirán de base para la generación del conjunto candidato en la siguiente iteración.

En particular, en el método Apriori, y variaciones de este como el AprioriTID y el Apriori Hybrid [Agrawal, 1994], en la k-ésima iteración se generan todos los itemsets frecuentes de tamaño k. En la siguiente iteración, para construir el conjunto de los (k+1)-*itemsets* candidatos, se expanden determinados k-*itemsets* frecuentes en un (k+1)-*itemset*, considerando ciertas reglas y condiciones conocidas por "heurística Apriori". Este proceso se repite hasta un cierto k, o hasta

que no se puedan generar más *itemsets* frecuentes por encontrarse su borde en el retículo por todas las ramas del árbol.

En el algoritmo Apriori se generan dos tipos de conjuntos: C_k y L_k. El conjunto L_k contiene los *itemsets* frecuentes de tamaño k (k-itemsets). Mientras tanto, C_k contiene el conjunto de los k-*itemsets* candidatos, representando un supraconjunto de L_k. Este proceso continúa hasta que se genere un conjunto L_k nulo, o se alcance el valor de k deseado. El conjunto L_k es obtenido recorriendo la base de datos y determinando el soporte de cada k-*itemset* candidato en C_k. El conjunto C_k se genera a partir de L_{k-1} con el siguiente procedimiento:

$$C_k = \{c \mid \text{Join}(c, L_{k-1}) \wedge \text{Prune}(c, L_{k-1})\}, \quad (6)$$

donde:

$$\text{Join}(\{i_1,\ldots,i_k\}, L_{k-1}) \equiv \langle \{i_1,\ldots,i_{k-2},i_{k-1}\} \in L_{k-1} \wedge \{i_1,\ldots,i_{k-2},i_k\} \in L_{k-1} \rangle, \quad (7)$$

$$\text{Prune}(c, L_{k-1}) \equiv \langle \forall s [s \subset c \wedge |s| = k-1 \rightarrow s \in L_{k-1}] \rangle. \quad (8)$$

Puede observarse que en la heurística Apriori se aplica directamente la propiedad de la clausura descendente, tanto en el proceso Join (7) como en el Prune (8).

En la siguiente figura se describe el algoritmo Apriori.

```
Algoritmo: Apriori
 1)  L1 = {large 1-itemsets};
 2)  for ( k = 2; Lk-1 ≠ ∅; k++ ) do begin
 3)    Ck = apriori_gen(Lk-1); // New candidates
 4)    forall transactions t ∈ D do begin
 5)      Ct = subset(Ck, t);      // Candidates contained in t
 6)      forall candidates c ∈ Ct do
 7)        c.Support ++ ;
 8)    end
 9)    Lk = {c ∈ Ck | c.Support ≥ minsup};
10)  end
11)  Answer = ⋃k Lk;
```

Fig. 4. Pseudocódigo del algoritmo Apriori

El procedimiento "apriori_gen(L_k-1)", indicado en el paso 3 de la figura anterior, se corresponde con lo definido en (6).

3.1.2. Métodos basados en particionamientos de las transacciones

La ventaja de mantener la base de datos en memoria y evitar las operaciones de E/S en disco, motivaron los trabajos de Savasere, Omiecinski y Navathe [Savasere, 1995]. Con tales objetivos, propusieron un algoritmo, al que denominaron *Partition*, el cual divide las transacciones de la base de datos en tantas partes disjuntas (o sea, en particiones no necesariamente de iguales tamaños) como fueran necesarias para que todas las transacciones en cada partición pudieran almacenarse en la memoria operativa.

En contraste con el algoritmo Apriori clásico, este recorre la base de datos sólo dos veces. En la primera fase, cada partición es minada independientemente para encontrar todos los *itemsets* frecuentes localmente. Al final de esta fase se toman los *itemsets* localmente frecuentes como *itemsets* candidatos globalmente. En la segunda fase, se determinan los soportes de los *itemsets* candidatos y se identifican los *itemsets* frecuentes. En la generación del conjunto de *itemsets* candidatos se considera la siguiente propiedad.

Propiedad. Particionamiento del soporte de un *itemset* o conjunto de ítems – Si un *itemset* no es localmente frecuente en ninguna parte, o subconjunto, de una partición de una base de datos, entonces no será frecuente globalmente.

De esta propiedad se infiere que los *itemsets* que no sean localmente frecuentes en ninguna parte entonces no deben considerarse como candidatos globales. Esta propiedad garantiza que en el conjunto de *itemsets* candidatos que se obtiene en la primera fase pudieran existir falsos positivos, pero nunca existirían falsos negativos. En la Fig. 5 se muestra el algoritmo considerando una partición P de m subconjuntos p_1 a p_m.

```
Algoritmo: Partition
 1)  for ( i = 1; i≤m ; i++ ) do begin // Phase I
 2)    "Load partition p_i ∈ P";
 3)    GenItemsets(p_i, L_i);
 4)  end
 5)  C = ⋃_{i=1..m} L_i ;
 6)  for ( i = 1; i≤m ; i++ ) do begin // Phase II
 7)    "Load partition p_i ∈ P";
 8)    AccumulateSupport(p_i, C);
 9)  end
10)  L = {c ∈ C | c.Support ≥ minsup};
11)  Answer = L
```

Fig. 5. Pseudocódigo del algoritmo Partition

Como se explicó en un inicio, este algoritmo, en la primera fase, carga en memoria cada partición y obtiene sus conjuntos frecuentes mediante el procedimiento GenItemsets, y une estos conjuntos formando el conjunto de candidatos. En la segunda fase, vuelve a cargar cada

partición determinando los soportes de cada candidato mediante el procedimiento AccumulateSupport. Aunque no es una exigencia a la lógica central del algoritmo, considerando que los tamaños de las partes son relativamente pequeños, los autores propusieron como estrategia de representación de las transacciones en cada parte el esquema de listas verticales VTL.

En la fig. 6 se muestra el procedimiento GenItemsets y en la fig. 7 el procedimiento AcumulateSupport.

```
Procedimiento: GenItemsets
Entrada: p_i – One of the m partitions.
Salida:  L_i – Set of large itemsets of partition p_i.
1)  L_i^1 = {large 1-itemsets on partition p_i along with their
    tidList}
2)  for ( k = 2; L_{k-1} ≠ ∅; k++ ) do
3)    forall itemset a = {a_1, …, a_{k-1} } ∈ L_i^{k-1} do
4)      forall itemset b = {b_1, …, b_{k-1} } ∈ L_i^{k-1} do
5)        if (Prefix_{k-2}(a) = Prefix_{k-2}(b)) and (a_{k-1} < b_{k-1}) then begin
6)          c = { a_1, …, a_{k-1}, b_{k-1} };
7)          if Prune(c, L_i^{k-1}) then begin
8)            c.tidList = a.tidList ∩ b.tidList
9)            if |c.tidList| / |p_i| ≥ minsup then
10)             L_i^k = L_i^k ∪ {c} ;
11)         end
12)       end
13) L_i = ⋃_k L_i^k
```

Fig. 6. Pseudocódigo del procedimiento GenItemsets

Como puede observarse, el procedimiento GenItemsets implementa un método tipo Apriori. Los pasos 5 y 6 se corresponden con el proceso Join (7). El procedimiento Prune del paso 7 se corresponde con el proceso de igual nombre (8), definidos ambos en Apriori. Por otra parte, los pasos 8 y 9 (así como los 4 y 5 del procedimiento AccumulateSupport) resuelven los cálculos de los soportes a partir de los tidList exigidos por el esquema VTL.

```
Procedimiento: AccumulateSupport
Entrada: p_i – Partition i.
Entrada/Salida: C – Candidates with accumulated support
                  from partition p_i.
1)  forall 1-itemset do
2)    "Generate its tidList on partition p_i";
3)  forall itemset c = { c_1, …, c_k } ∈ C do begin
4)    TempList = ⋂_{j=1..k} c_j.tidList ;
5)    c.Support = c.Support + |TempList| ;
6)  end
```

Fig. 7. Pseudocódigo del procedimiento AcumulateSupport

Aunque no es objeto de análisis en este trabajo, la lógica de este algoritmo y la propiedad en la que se basa han sido aplicadas en diversos algoritmos paralelos.

3.1.3. Métodos basados en particionamientos de los ítems

Siguiendo una lógica de particionamiento, pero en este caso del conjunto de ítems de la base de datos, se han propuesto los algoritmos tipo Multipass M-Apriori y M-DHP [Holt 2001].

Las razones que motivaron a los autores de este trabajo en particionar el conjunto de ítems en lugar de la base de datos se tienen en su aplicación a bases de datos textuales. En este tipo de base, la cantidad de ítems puede ser muy alta, y en ocasiones superior, respecto a la cantidad de transacciones o documentos.

La idea básica de los algoritmos M-Apriori y M-DHP es particionar los conjuntos de 1-*itemsets* frecuentes en *p* partes y procesar cada una separadamente. El procedimiento propuesto por Holt y Shung es el siguiente:

1. Contar las ocurrencias de cada ítem en la base de datos para encontrar los 1-*itemsets* frecuentes.
2. Particionar el conjunto de los 1-*itemsets* frecuentes en *p* partes: $\{P_1, P_2, \ldots, P_p\}$.
3. Aplicar Apriori o DHP (descrito en un epígrafe posterior) para encontrar todos los *itemsets* frecuentes hasta la parte procesada, desde la última (P_p) hasta la primera (P_1), determinando sus soportes sobre la base de datos. La lógica de este paso es la siguiente: cuando la parte P_p se ha procesado, se han encontrado todos los itemsets frecuentes cuyos ítems están en P_p. Cuando se procesa la próxima parte P_{p-1}, se pueden encontrar todos los *itemsets* frecuentes cuyos ítems están en P_{p-1} y P_p, con sus prefijos en P_{p-1}. Más específicamente, se determinan los *itemsets* frecuentes considerando en sus extensiones sólo los *itemsets* de P_p que son frecuentes. Este procedimiento continúa hasta que se procese P_1.

Se asume, sin pérdida de generalidad, que los ítems están ordenados en orden lexicográfico, de forma tal que en las *p* partes: $\forall\, i < j, \forall\, a \in P_i, \forall\, b \in P_j \;\; a < b$. Este orden permite conocer si los *sub-itemsets* son frecuentes por haberse ya procesado partes léxicamente superiores, facilitando la aplicación de la propiedad de clausura descendente.

3.1.4. Métodos de hashing

En general, hasta este instante se han presentado varios algoritmos descendentes con recorrido a lo ancho para calcular los *itemsets* frecuentes. Estos algoritmos generan un conjunto de *itemsets* candidatos, basados en alguna heurística, y a partir de éste determinan el subconjunto que

contiene a los *itemsets* frecuentes. Este proceso puede ser implementado de forma iterativa debido a que los *itemsets* frecuentes encontrados en una iteración pueden utilizarse en la generación de los candidatos de la iteración siguiente.

Como ha podido observarse, la heurística para decidir si un *itemset* es candidato es crucial para el funcionamiento de estos algoritmos. De esta forma, para lograr una mayor eficiencia se debería contar con heurísticas que generen solamente candidatos con una alta probabilidad de ser frecuentes; puesto que para determinar esto se precisa determinar sus soportes, operación esta de gran consumo de tiempo.

Debido a lo anterior, han surgido una serie de algoritmos con heurísticas que mejoran el cálculo de los conjuntos de itemset candidatos. Ejemplos de estos algoritmos son aquellos que utilizan tablas de *hash*, como son el DHP, IHP y MIHP, los cuales se exponen a continuación.

3.1.4.1. Algoritmo DHP

Como resultado directo de la propiedad de clausura descendente, en el proceso de generación de itemsets frecuentes, el tamaño de los conjuntos de *itemsets* frecuentes – y por lo tanto de sus conjuntos candidatos – en las iteraciones iniciales es mucho mayor que los de las iteraciones avanzadas. Esta observación muestra la conveniencia de disminuir al máximo los conjuntos de candidatos en las fases iniciales, especialmente la de los 2-*itemsets*.

Otro factor que atenta contra el rendimiento del proceso de generación de *itemsets* frecuentes es la cantidad de datos que es necesario procesar para determinar el soporte de los *itemsets* candidatos. Nótese que a medida que el proceso de generación de itemsets frecuentes avanza no sólo se reducen los conjuntos de *itemsets* frecuentes, sino también la cantidad de transacciones que contienen a los *itemsets* de cada iteración. Luego, la eliminación tanto de aquellas transacciones que no serán capaces de contener a *itemsets* mayores, así como de los ítems que contienen las transacciones y que no formarán parte de ningún *itemset* frecuente de tamaño superior, debe incrementar el rendimiento de un algoritmo de cálculo de *itemsets* frecuentes.

El algoritmo DHP (*Direct Hashing and Pruning,* por sus siglas en Inglés) emplea una técnica de *hashing* para eliminar *itemsets* innecesarios en la generación del conjunto de candidatos, considerando a su vez heurísticas para la reducción efectiva del tamaño de las transacciones y la disminución del número de transacciones a procesar [Park, 1997]; [Holt, 1999].

Cuando se explora la base de datos para calcular el soporte de cada *k*-*itemset* candidato, se generan informaciones sobre cada (*k*+1)-*itemset* elegible como candidato en adelanto de forma tal que todos los posibles (*k*+1)-*itemsets* de cada transacción, tras cierto filtraje, son acumulados en una tabla de *hash*.

En cada celda de la tabla de hash se cuentan los (*k*+1)-*itemsets* con iguales valores de *hash*. Es importante mencionar que, debido a las colisiones, *itemsets* diferentes pueden contarse en una misma celda de la tabla de *hash*. Por lo tanto, el valor de cualquier celda será la suma de los soportes de todos los *itemsets* con iguales valores de *hash*. Lógicamente, dicho valor de celda representa una cota superior de los soportes de los *itemsets* asociados. De esta forma, cada (*k*+1)-*itemset* posible será considerado como candidato en la siguiente pasada si su valor en la tabla de hash es mayor que el soporte mínimo

Para reducir la dimensión de la base de datos, este algoritmo propone una heurística que se basa en la propiedad de la clausura descendente del soporte de los *itemsets*. Esta propiedad

sugiere que una transacción puede contener un candidato (k+1)-*itemset* sólo si contiene k+1 *itemsets* candidatos de tamaño k (k-*itemset*) obtenidos en la iteración anterior. Luego, si una transacción es evaluada para contar las ocurrencias de los k-*itemsets* candidatos, entonces puede determinarse si esa transacción puede ser eliminada (podada) de la base de datos en la pasada siguiente. Por otra parte, si una transacción contuviera un (k+1)-*itemset* frecuente, entonces cualquier ítem contenido en ese (k+1)-*itemset* debería aparecer en al menos k de los k-*itemsets* candidatos contenidos en esa transacción. Por lo tanto, si un ítem de una transacción no satisface tal condición, este puede eliminarse (recortarse) de esa transacción para la siguiente pasada.

La heurística descrita anteriormente se resume en las propiedades siguientes.

Propiedad. *Recorte de transacción (transaction trimming)* – Un ítem de una transacción t puede ser eliminado de t si:

- Este no aparece en al menos k de los k-itemsets candidatos contenidos en esa transacción.
- Este no aparece en al menos un (k+1)-itemset elegible de esa transacción, considerando como elegible aquel con todos sus k+1 itemsets de tamaño k contenidos en C_k.(conjunto de candidatos de k-itemsets).

Propiedad. *Poda de transacción (transaction pruning)* – Si una transacción t puede contener a un (k+1)-itemset, entonces esta debe contener a (k+1) itemsets de tamaño k. En caso contrario, esta puede ser eliminada.

En este algoritmo se emplean las siguientes notaciones: C_k es el conjunto de candidatos de k-*itemsets*, H_k es la tabla de *hash* para contar las ocurrencias de los (k+1)-*itemsets* en las transacciones.

```
Algoritmo: DHP
 1)  // Part 1
 2)  forall transaction t_i ∈ D do begin
 3)    "Insert and count 1-itemset occurrences in a hash tree";
 4)    forall 2-itemset x of t_i do
 5)      H_2[h_2(x)]++ ;
 6)  end
 7)  L_1 = { c | c.Support ≥ minsup ∧
                "c exist in a leaf node of the hash tree" };
 8)  // Part 2
 9)  k = 2 ;
10)  D_k = D ; // Database for large k-itemset
11)           // Make a hash table
12)  while |{ x | H_k[h_k(x)] ≥ minsup }| ≥ LARGE do begin
13)    GenCandidate(L_k-1, H_k, C_k);
14)    D_k+1 = ∅;
15)    forall transaction t ∈ D_k do begin
16)      CountSupport(t, C_k, k, t');      // t' ⊆ t
17)      if |t'| > k then begin
18)        MakeHashT(t', H_k, k, H_k+1, t'');
19)        if |t''| > k then D_k+1 = D_k+1 ∪ {t''};
20)      end
21)    end
22)    L_k = { c ∈ C_k | c.Support ≥ minsup };
23)    k++;
24)  end
25)  // Part 3
26)  GenCandidate(L_k-1, H_k, C_k);
27)  while |C_k| > 0 do begin
28)    D_k+1 = ∅;
29)    forall transaction t ∈ D_k do begin
30)      CountSupport(t, C_k, k, t');  // t'⊆t
31)      if |t'| > k then D_k+1 = D_k+1 ∪{t'};
32)    end
33)    L_k = { c ∈ C_k | c.Support ≥ minsup };
34)    if |D_k+1| = 0 then break;
35)    C_k+1 = apriori_gen(L_k); // refer to Apriori
36)    k++;
37)  end
38)  Answer = ⋃_k L_k ;
```

Fig. 8. Pseudocódigo del algoritmo DHP

A continuación se exponen los pseudocódigos de los métodos en los que se apoya el DHP para su funcionamiento.

```
Procedimiento: GenCandidate
Entrada: L_{k-1} - Conjunto de (k-1)-itemsets frecuentes
                 H_k - Tabla de hash de los k-itemsets posibles candidatos
Salida:      C_k - Conjunto de k-itemsets candidatos
1)  C_k = ∅;
2)  foreach itemset c = {c_p[1], c_p[2], …, c_p[k-2], c_p[k-1], c_q[k-1]}
            and c_p, c_q ∈ L_{k-1} and |c_p ∩ c_q| = k-2 do
3)    if H_k[h_k(c)] ≥ minsup then
4)      C_k = C_k ∪ {c};  //  Insert c into a hash tree
```

Fig. 9. Pseudocódigo del procedimiento GenCandidate

```
Procedimiento: CountSupport
Entrada: t - Transacción que esta siendo analizada
                C_k - Conjunto de k-itemsets candidatos
                k - Tamaño de los itemsets candidatos
Salida: t' - Transacción que se analiza modificada
1)  // Count support and make transaction trimming
2)  forall itemset c ∈ C_k and c = {t_{i_1}, t_{i_2}, ⋯, t_{i_k}} ⊂ t do begin
3)    c.Support ++;
4)    for ( j = 1; j ≤ k; j++ ) do a[i_j]++;
5)  end
6)  for ( i = 0; i < |t|; i++ ) do
7)    if a[i] ≥ k then t' = t' ∪ {t_i};
```

Fig. 10. Pseudocódigo del procedimiento CountSupport

```
Procedimiento: MakeHashT
Entrada: t' - Transacción que será evaluada
               H_k - Tabla de hash de los k-itemsets
               k - Tamaño de los itemsets de la tabla H_k
Salida: t'' - Transacción t' modificada
          H_{k+1} - Tabla de hash de los (k+1)-itemsets
1)  // Make hash table and a further transaction trimming
2)  forall (k+1)-itemset x = {t'_{i_1}, t'_{i_2}, ⋯, t'_{i_{k+1}}} ⊂ t' do
3)    if "∀ k-itemset y ⊂ x , H_k[h_k(y)] ≥ minsup" then begin
4)      H_{k+1}[h_{k+1}(x)]++ ;
5)      for ( j = 1; j ≤ k+1; j++ ) do a[i_j]++ ;
6)    end
7)  for ( i = 0; i < |t|; i++ ) do
8)    if a[i] > 0 then t'' = t'' ∪ {t'_i};
```

Fig. 11. Peusocógido del procedimiento MakeHashT

Los autores de este trabajo plantean que el algoritmo DHP se ve afectado positivamente con el incremento de los tamaños de las tablas de *hash*, aunque esto se logra a costo de la memoria [Park, 1997].

3.1.4.2 Algoritmo IHP

El algoritmo IHP (*Inverted Hashing and Pruning,* por sus siglas en Inglés) [Holt, 2002a] es análogo a DHP ya que ambos utilizan tablas de *hash* para podar algunos de los *itemsets* candidatos. En DHP, durante la *k*-ésima pasada sobre la base de datos, todo (*k*+1)-itemset contenido en cada transacción es acumulado en una tabla de hash permitiendo decidir, en caso que el valor de ese (*k*+1)-*itemset* sea menor que el soporte mínimo, que dicho *itemset* no sea considerado como candidato en la siguiente pasada.

En el algoritmo IHP, en lugar de acumular los *itemsets* en tablas de *hash* lo que se acumulan son los identificadores de las transacciones (TID), denominándose por ello estas tablas como TID *Hash Tables* (THT). Para lograr un efecto similar; o sea, para decidir si se puede alcanzar el soporte mínimo para un *itemset* se cuenta con una THT por cada ítem, verificándose que en todas las THT de ítem se utilice la misma función de *hash* de TID. Nótese que, debido a las colisiones, una celda de THT acumulará varios TID, asociándose las celdas con una partición del conjunto de transacciones. En la primera exploración de la base de datos se acumula el TID de cada transacción en cada THT de los ítems contenidos en ella, eliminándose al final de la primera pasada aquellos ítems que no resultaron frecuentes. Para saber si un ítem es frecuente basta con sumar el valor acumulado en cada celda de la THT de ese ítem y comprobar que no sea menor que el soporte mínimo establecido.

En general, tras cada pasada *k*-ésima, $k \geq 1$, se pueden eliminar los THT de ítems que no se encuentren en ningún *k-itemset* frecuente, utilizándose los restantes THT para determinar los (*k*+1)-*itemsets* candidatos.

Para decidir si un *itemset* debe considerarse como candidato se presenta el siguiente lema.

Lema. Sean i_0.THT[j], i_1.THT[j], …, i_n.THT[j] los soportes en un subconjunto de transacciones para cada ítem posible de una base de datos y sea c un *itemset*, entonces la expresión $\min_{i \in c}(i.\text{THT}[j])$ representa una cota superior del soporte de ese *itemset* en el subconjunto de transacciones.

Observe que este lema es una consecuencia directa de la clausura descendente. En este, el subconjunto de transacciones se corresponde con los TID con iguales valores de *hash* (*j*). A partir de este lema, entonces puede enunciarse la siguiente propiedad.

Propiedad. Sean i_0.THT, i_1.THT, …, i_n.THT tablas de *hash* conteniendo por cada entrada los soportes de cada ítem posible en cada parte o subconjunto de una partición de la base de datos y sea c un *itemset*, entonces la siguiente expresión representa su soporte máximo posible (SMP); o sea, una cota superior del soporte de ese *itemset* en la base de datos.

$$SMP(c) = \sum_j \min_{i \in c}\left(i.\text{THT}[j]\right) \qquad (9)$$

Si el soporte máximo posible (SMP) de c es menor que el soporte mínimo (*minsup*), entonces ese *itemset* no debe incluirse en el conjunto de candidatos.

Para obtener mejores resultados se propone combinar el algoritmo IHP con los criterios de recorte y poda (*transaction trimming and pruning*) definidos en DHP.

3.1.4.3. Algoritmo MIHP

El algoritmo MIHP (*Multipass with Inverted Hashing and Pruning,* por sus siglas en Inglés) [Holt, 2002b] es una combinación de los algoritmos M-Apriori (*Multipass* Apriori) e IHP (*Inverted Hashing and Pruning*) propuestos por Holt *et al.* en [Holt, 2001] y [Holt, 2002a] respectivamente. La concepción general del algoritmo es aprovechar las características de M-Apriori en cuanto a reducir el espacio de memoria requerido mediante el particionamiento de los 1-*itemsets* frecuentes y el procesamiento de cada partición separadamente y, al mismo tiempo, utilizar el algoritmo IHP para en cada pasada eliminar de forma más eficiente algunos de los *itemsets* candidatos generados.

3.2. Métodos descendentes con recorridos en profundidad

Como se menciona en el epígrafe 3.1, en la búsqueda de los conjuntos de ítems frecuentes se pueden emplear dos formas de recorrido del árbol de acuerdo a su amplitud: a lo ancho y en profundidad. La estrategia seguida por el método Apriori, y en general por los métodos del tipo Apriori, garantiza que todos los *itemsets* frecuentes sean visitados, al mismo tiempo que reduce el número de *itemsets* infrecuentes visitados. Cuando se usa la estrategia en profundidad los conjuntos candidatos constan solamente de conjuntos de ítems de los nodos anteriormente visitados. No obstante, esta estrategia presenta el inconveniente de reiterados recorridos de la base de datos, lo cual es extremadamente costoso; a pesar de ello, muchos de estos métodos logran resultados significativos.

Los métodos descendentes con recorridos en profundidad pueden agruparse a partir de sus dos algoritmos básicos: los basados en árboles, cuyo paradigma es FP-Growth, y los basados en agrupamientos de maximales potenciales, cuyo paradigma es Eclat. A continuación se exponen esos algoritmos y algunas variaciones propuestas.

3.2.1. Métodos basados en árboles

Los algoritmos en profundidad que utilizan árboles e intersección de conjuntos para el proceso de minado de la base de datos tienen la ventaja de que realizan una representación compacta de la base de datos, permitiendo el cálculo del soporte de manera eficiente y requiriendo solamente dos pasadas por la base de datos.

El problema de generar candidatos puede ser eliminado utilizando el método de crecimiento de ítems frecuentes [Agrawal, 1994], [Agrawal, 1999], [Bayardo, 1998], que adopta el paradigma de divide y vencerás, realizando el minado de los *itemsets* frecuentes sin generación de candidatos. Para la mayoría de las base de datos, este método resulta más eficiente que los expuestos en epígrafes anteriores.

El crecimiento de ítems frecuentes ha sido desarrollado y analizado de diferentes formas, cada una con sus particularidades como se muestra a continuación:

- En vez de generar un gran número de candidatos, se conserva (de forma compacta) el grupo esencial de elementos originales de la base de datos. Debido a esto, el análisis se centra en el cálculo del soporte de *itemsets* y no en generar candidatos.
- En vez de recorrer toda la base de datos para verificar si un *itemset* es frecuente, se particionan los *itemsets* para que sean examinados sólo cuando sea necesario. Aquí se pone de manifiesto la metodología de divide y vencerás, reduciendo sustancialmente el espacio de búsqueda y elevando la eficiencia del algoritmo.

Entre los algoritmos desarrollados aplicando este método, y que serán analizados a continuación, se tienen los siguientes: FP-Growth, Patricia Trie-Mine, CTITL-Mine, CTPRO-Mine.

3.2.1.1. Algoritmo FP-Growth

FP-Growth, paradigma de este grupo, es un algoritmo que se basa en el crecimiento de ítems o patrones frecuentes, utilizando una estructura compacta y eficiente de datos denominada FP-Tree [Han, 2000].

El FP-Tree es una estructura que almacena información importante acerca de los conjuntos frecuentes que se encuentran en la base de datos. Sólo los ítems frecuentes tendrán asociados nodos en el árbol; a su vez, los nodos en el árbol son reordenados de forma tal que los de mayor soporte tengan más probabilidad de compartir nodos que otros con menos soporte. Esta estructura (compacta) de datos es diseñada basada en las siguientes consideraciones:

- Como los itemsets frecuentes son los que resultan relevantes en el proceso de búsqueda de ítems frecuentes, es necesario realizar una iteración sobre la base de datos para determinar cuáles son estos ítems.
- Si se almacena en alguna estructura compacta el conjunto de ítems frecuentes de cada transacción, entonces se evita tener que hacer iteraciones reiteradas e innecesarias sobre la base de datos.
- Si múltiples transacciones tienen el mismo conjunto de ítems frecuentes, entonces estas pueden unirse en una sola que almacene el número de veces que ocurre esto. Esta operación se puede realizar de forma sencilla si los ítems en cada transacción estuvieran ordenados de acuerdo a un orden prefijado.
- Si dos transacciones comparten un prefijo común, de acuerdo a un cierto orden prefijado, entonces las partes comunes pueden unirse en una sola estructura de prefijos que almacene también el número de veces que ha ocurrido esto. Si los ítems de cada transacción son ordenados en orden descendente de acuerdo a su soporte, entonces existirá mayor probabilidad de que se compartan más prefijos y con ello se reduzca el gasto de memoria de almacenaje y a su vez se eliminen iteraciones innecesarias sobre la base de datos.

Con estas observaciones se puede construir el FP-Tree a través de los siguientes pasos:

1. Se realiza una iteración sobre la base de datos para determinar los ítems frecuentes. Luego, estos son ordenados descendentemente de acuerdo a su soporte e insertados en una lista *L*.
2. Se crea la raíz del FP-Tree y se inicializa como vacía. Posteriormente, para cada transacción *t* de la base de datos se realizan las siguientes acciones:
   - Se seleccionan de *t* los ítems que resultaron frecuentes y se insertan en la lista *P* ordenadamente de acuerdo al orden que ocupan en la lista *L*.
   - Esta lista de ítems frecuentes se le pasa como parámetro a un procedimiento que se encarga de adicionar una transacción al FP-Tree siguiendo las consideraciones planteadas anteriormente.

Para facilitar el recorrido por el árbol, se construye una tabla denominada *Header Table* que almacena cada ítem frecuente, ordenado descendentemente de acuerdo al valor de su soporte. Cada celda de dicha tabla almacenara los siguientes atributos:

- Ítem: Representa un ítem frecuente.
- Cantidad: Representa el número de transacciones que contienen al ítem frecuente.
- Enlace: Un apuntador a la primera aparición de ese ítem en el árbol.

A su vez, cada nodo del árbol es una estructura formada por los siguientes atributos:

- Ítem: Representa un ítem frecuente de la transacción almacenada en dicho nodo.
- Cantidad: Representa al número de transacciones que tienen en común el prefijo que se forma al concatenar todos los ítems que se encuentran en el camino desde la raíz del árbol hasta el nodo.
- Enlace: Un apuntador que almacena la dirección del próximo nodo en el árbol que almacena al mismo ítem, o el enlace nulo en caso contrario.
- Hijos: Un conjunto de apuntadores que enlazan al nodo con sus descendientes.

Como se observa en la descripción de las estructuras anteriores, cada ítem frecuente se encuentra enlazado a partir de la tabla *Header Table* con todas sus apariciones en el FP-Tree.

En la Tabla 1 y Fig. 12 se muestra un ejemplo de base de datos y estructura FP-Tree asociada, considerando que *minsup* = 3.

Tabla 1. Base de datos transaccional

TID	Ítems de la Transacción	Ítems Frecuentes
100	f; a; c; d; g; i; m; p	f; c; a; m; p
200	a; b; c; f; l; m; o	f; c; a; b; m
300	b; f; h; j; o	f; b
400	b; c; k; s; p	c; b; p
500	a; f; c; e; l; p; m; n	f; c; a; m; p

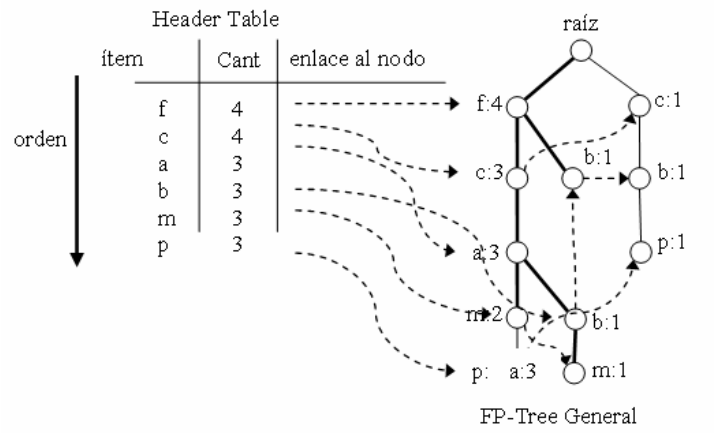


Fig. 12. FP-Tree construido para la base de datos de la Tabla 1

Nótese que el árbol del FP-Tree posee las siguientes características:

- La cantidad de nodos hojas coincide con la cantidad de conjuntos de ítems únicos diferentes entre todas las transacciones de la base de datos, estando la cantidad de los nodos intermedios en correspondencia con la cantidad de prefijos coincidentes entre todas las transacciones.
- La altura del árbol depende de la cardinalidad del conjunto de ítems únicos mayor entre todas las transacciones.

– Para cada ítem frecuente a_i, todos los posibles *itemsets* frecuentes en los que puede estar contenido ese ítem se pueden encontrar navegando a través de la lista enlazada que contiene a los nodos que almacenan al ítem frecuente a_i y que se puede acceder a partir de la correspondiente celda del *Header Table*.

Definición. *Patrón Base condicional de un ítem* – El Patrón Base condicional de un ítem *p* cualquiera está formado por los caminos en el FP-Tree que incluyen a los prefijos del nodo que almacena al ítem frecuente *p*.

Definición. *FP-Tree condicional de un ítem* – El FP-Tree condicional de un ítem *p* es el que se forma a partir de su patrón base condicional.

En la siguiente figura se muestra un ejemplo de un patrón base condicional y un FP-Tree condicional para el ítem "*m*" dado el FP-Tree.

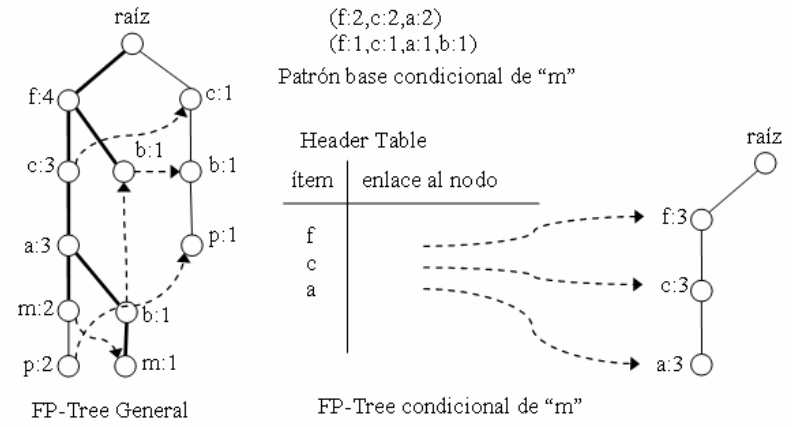


Fig. 13. Ejemplo de patrón base condicional y FP-Tree condicional para el ítem "*m*"

Basándose en las propiedades abordadas anteriormente se puede enunciar el algoritmo FP-Growth, el cual consta de los siguientes pasos:

1. Construir la base condicional del patrón frecuente para cada patrón en el *Header Table.*
2. Construir el FP-Tree condicional de cada base condicional del patrón frecuente analizado.
3. Realizar el minado recursivo del FP-Tree condicional y repetir el proceso sobre los patrones frecuente obtenidos. Si el FP-Tree condicional contiene un solo camino, entonces todas las combinaciones de patrones que se pueden formar con los ítems que representa cada nodo son frecuentes.

La idea general seguida para efectuar el proceso de minado es la siguiente:

1. Para cada patrón frecuente, se construye su base condicional, y luego su FP-Tree condicional.
2. Repetir el proceso en cada FP-Tree condicional creado hasta que el FP-Tree resultante sea nulo, o contenga sólo un camino.

En la siguiente figura se evidencia este proceso.

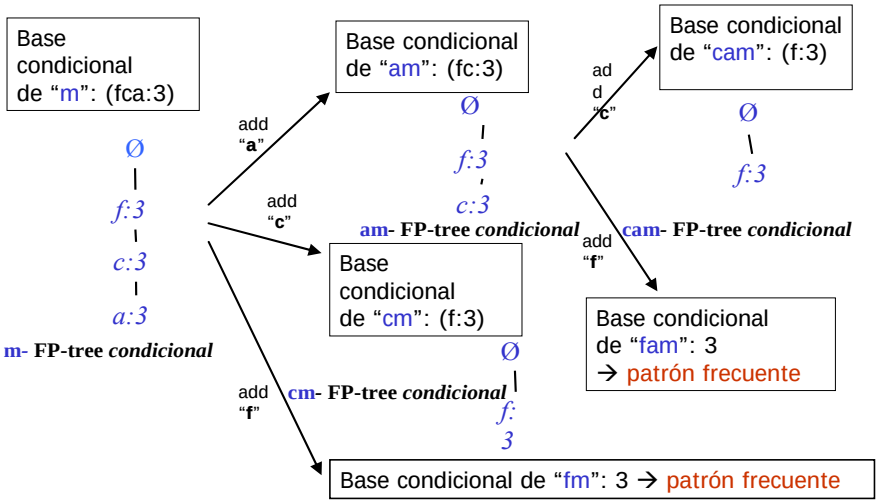


Fig. 14. Ejemplo de minado recursivo del FP-Growth

El proceso de crecimiento de los patrones frecuentes se basa en la siguiente propiedad.

Propiedad. Sea α un patrón frecuente en una base de datos, B la base condicional del patrón frecuente α, y β un patrón frecuente en B. El conjunto $\alpha \cup \beta$ es un patrón frecuente en la base de datos si y sólo si β es frecuente en la base de datos.

Como principales ventajas de este algoritmo se pueden mencionar las siguientes:

- Un FP-Tree es más pequeño que el tamaño de la base de datos.
- La base condicional del patrón frecuente es más pequeña que el FP-Tree original.
- El FP-Tree condicional es más pequeño que la base condicional del patrón frecuente.

Posteriormente al FP-Growth, se desarrollaron algoritmos basados en la misma heurística pero representando los datos de diferente manera, adecuándose a las características propias de las base de datos y/o comprimiendo aún más la información existente en memoria. Estos algoritmos se presentan a continuación.

3.2.1.2 Algoritmo Patricia Trie-Mine

Este algoritmo compacta los nodos del FP-Growth que tienen igual valor del contador en un solo nodo. La estructura para almacenar la información relevante para el proceso de minado se denomina *Patricia Trie-Mine* y permite compactar aún más la información representada [Pietracaprina 2003].

El Patricia Trie consta de similares características que el árbol del FP-Growth. Su principal diferencia, que a la vez es la ventaja del *Patricia Trie-Mine*, es que aquellos nodos que compartan igual información se funden en uno solo, incrementándose únicamente un contador.

Para la representación de una TID en un *Patricia Trie-Mine* con el objetivo de su minado en profundidad, aquellos ítems frecuentes de una transacción que compartan información de soporte son representados en un solo nodo disminuyendo el espacio ocupado por la representación.

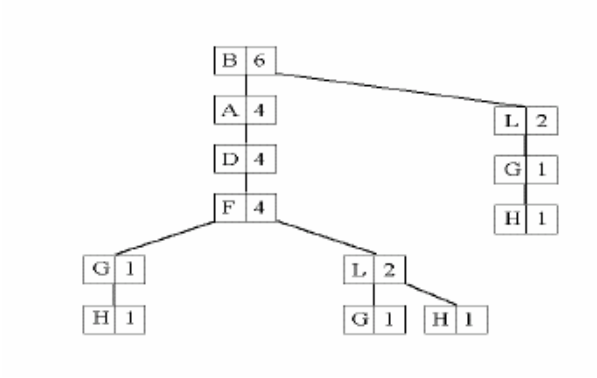


Fig. 15. FP-Tree estándar para un TID

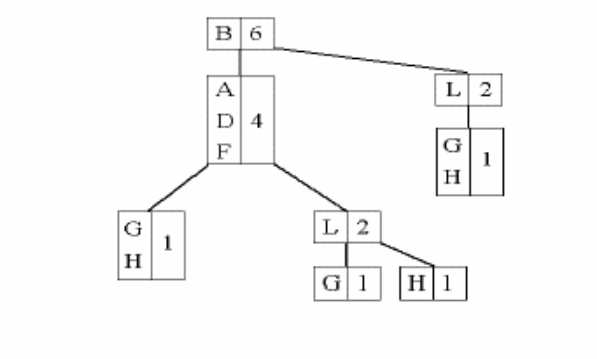


Fig. 16. Patricia Trie para similar TID

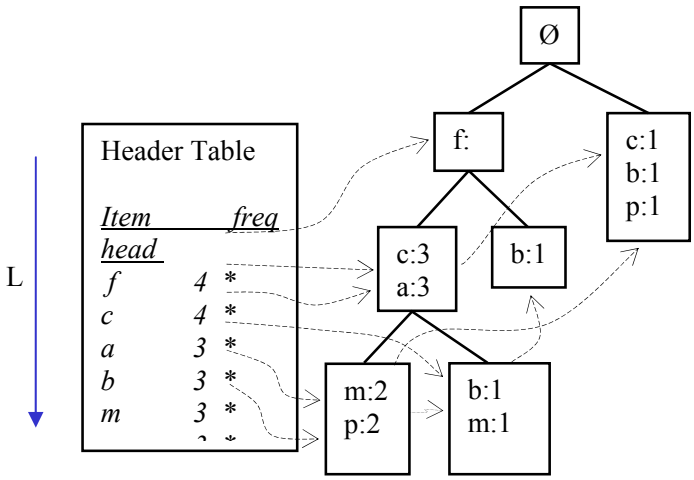


Fig. 17. Ejemplo de minado recursivo del Patricia Trie-Mine

3.2.1.3. Algoritmo CTITL-Mine

Este algoritmo utiliza algunas de las ideas más útiles de otros algoritmos de búsqueda de itemsets frecuentes como son: los árboles de prefijos (FP-Tree), la técnica del crecimiento de patrones (utilizada en el FP- Growth) y la intersección de listas de TID. Como resultado, se obtiene un algoritmo que utiliza una nueva estructura de datos denominada CT-Tree (*Compress Transaction Tree*) y que junto con otro estructura llamada ITL (*Item Trans Link*, por sus siglas en inglés) realiza la búsqueda de forma más eficiente de los itemsets frecuentes [Gopalan, 2003].

La estructura CT-Tree permite reducir el número de nodos en el *transaction tree* o árbol de transacciones, permitiendo de esta forma un mejor agrupamiento de las transacciones que comparte un conjunto de ítems. Un árbol de prefijos completo puede contener muchos sub-árboles idénticos; luego, si se quiere construir un árbol que represente la información en forma más comprimida se debe almacenar de forma unida la información contenida en aquellos sub-árboles idénticos.

Siguiendo la idea anteriormente descrita se podría entonces comprimir el árbol de prefijos de la figura siguiente, el cual contiene 16 nodos, en un árbol más pequeño que contenga solamente 8 nodos almacenando para ello alguna información extra en los nodos del árbol resultante. En la siguiente figura se ha considerado que la base de datos presenta como ítems únicos el conjunto: {1, 2, 3, 4}.

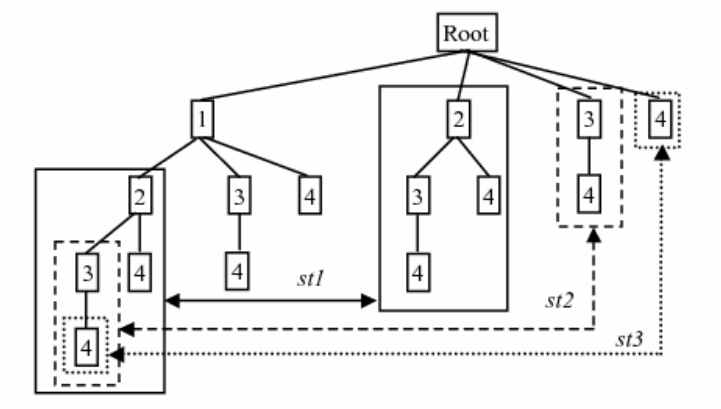


Fig. 18. Sub-árboles idénticos en un árbol de prefijos

El árbol comprimido que represente al árbol de la figura anterior necesita guardar información acerca de los nodos que se han eliminado del árbol de prefijos completos. Para mantener un acumulado del número de transacciones que contienen a cada subconjunto de ítems se adiciona en cada nodo, y especialmente para cada subconjunto que contenga al ítem almacenado en este nodo, un contador. Cada celda de este arreglo contiene dos campos:

1. El nivel en que se encuentra el primer ítem del subconjunto representado
2. El número de transacciones que contiene al subconjunto

A continuación se muestra un ejemplo de cómo sería el CT-Tree o árbol de transacciones comprimido que represente al árbol de prefijos de la figura anterior.

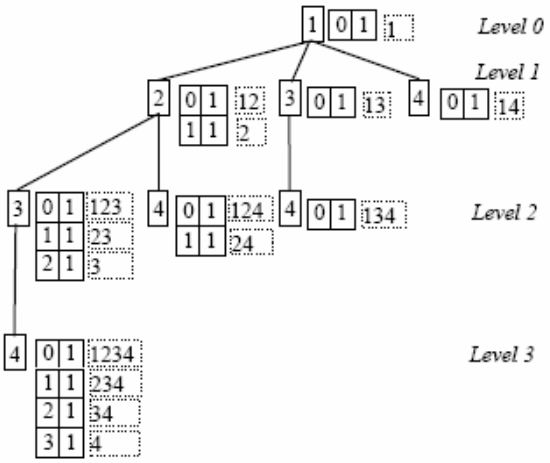


Fig. 19. Ejemplo de CT-Tree o árbol de transacciones

Como se observa en la figura anterior, el nodo 3 en la rama más a la izquierda contiene 3 entradas: (0,1), (1,1) y (2,1). La primera indica que existe una sola transacción en la base de datos que contiene al *itemset* que comienza por el ítem que está en el nivel 0 en el camino desde la raíz a ese nodo; esta entrada representa al *itemset* '123'. La entrada (1,1) significa que existe una sola transacción que contiene al *itemset* que comienza en el nivel 1 del camino que parte

desde la raíz hasta el nodo que se analiza. El mismo análisis se le puede realizar a la última entrada. Note que en este ejemplo se ha asumido que el contador para cada entrada en cada nodo es uno. En la figura se ha adicionado en rectángulos con líneas discontinuas los *itemset* que son representados por cada entrada para facilitar la comprensión de la figura.

La otra estructura de datos en la que se basa el funcionamiento del algoritmo, como se había mencionado anteriormente, es el ITL el cual posee características de los esquemas de representación de datos vertical y horizontal. Esta estructura esta compuesta por dos campos: una tabla de ítems (*ItemTable*) y la lista de transacciones enlazadas a esta (*TransLink*). El *ItemTable* almacena todos los ítems frecuentes de la base de datos, cada uno con su soporte y un enlace a la primera aparición de este en *TransLink*.

La lista *TransLink* representa a cada transacción que contiene a ítems que se encuentran en el *ItemTable*. Los ítems en las transacciones son reordenados y cada ítem en la transacción es enlazado con su próxima aparición en la siguiente transacción. En otras palabras, existe un enlace que conecta a todos las apariciones de un ítem en la base de datos, o si se tiene en cuenta la representación binaria de la base de datos, un enlace que conecta a todos los 1's que expresan la presencia del ítem en la transacción. De esta forma, esta estructura permite un rápido conteo del soporte de los ítems. A continuación se muestra una base de datos y su representación utilizando esta estructura.

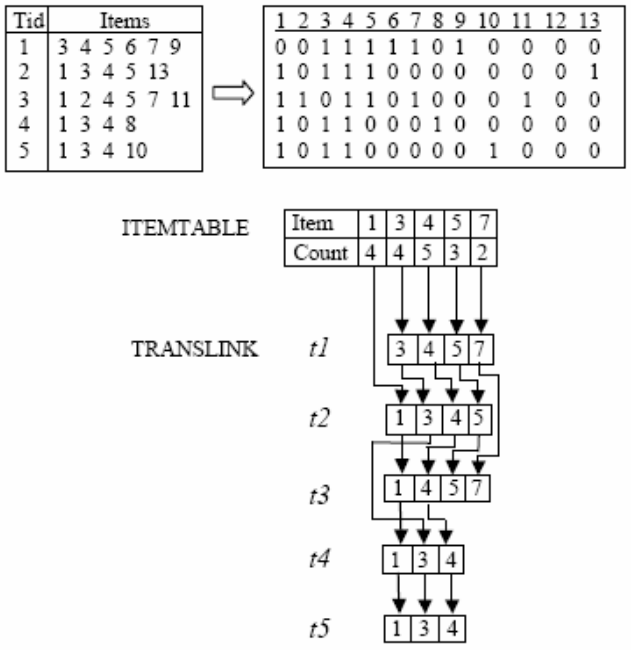


Fig. 20. Ejemplo del uso de la estructura ITL para representación de una base de datos.

Otras de las propiedades de esta estructura se muestran a continuación:

1. Los identificadores de los ítems son discretizados a un rango de valores enteros en la ITL.
2. Los identificadores de las transacciones son ignorados pues los ítems de cada transacción son enlazados.

Como se puede apreciar, esta estructura es muy apropiada para aquellos algoritmos que necesiten de esquemas de representación horizontal o vertical, e incluso una combinación de estos.

Para incrementar la eficiencia, este algoritmo modifica la estructura ITL de forma tal que no almacene las transacciones de la base de datos real, sino aquellas que representa el CT-Tree. Además, para cada fila de TransLink se almacena información concerniente a la cantidad de transacciones que contienen diferentes subconjuntos de ítems. A continuación se muestra un ejemplo de la modificación realizada sobre el ITL.

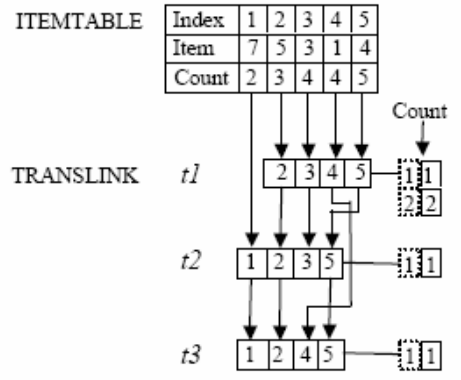


Fig. 21. Ejemplo de ITL modificada

La transacción *t1* en el campo TransLink es el resultado de la discretización del ítem 5 en el valor 2 del rango de valores enteros del ITL. Luego, *t1* representa a la transacción '2345' (que representa a los ítems del *itemset* '5314'), la cual ocurre una sola vez en la base de datos; además, también representa a la transacción '345' (originalmente '314') que ocurre 2 veces.

El proceso de minado es similar al del FP-Growth y sigue los siguientes pasos:

1. Identificar el conjunto de ítems frecuentes e inicializar el *ItemTable*. Para poder utilizar el patrón de compresión en pasos posteriores, los ítems son insertados en el *ItemTable* en orden ascendente de acuerdo al valor de su soporte y se les asigna un nuevo identificador de acuerdo al lugar que ocupan en la tabla.
2. Construir el CT-Tree utilizando como datos la salida del primer paso del algoritmo, por lo que sólo los ítems frecuentes son leídos de la base de datos.
3. Construir el ITL. En este paso se recorre el CT-Tree para construir el *ItemTable* y el *TransLink*.
4. Se buscan los *k-itemsets* frecuentes, $k \geq 2$, utilizando una función recursiva similar a la del FP-Growth.

3.2.1.4 Algoritmo CTPRO-Mine

Este algoritmo utiliza la estructura de datos denominada CT-Tree (*Compress Transaction Tree,* poe sus siglas en inglés) para el minado de los ítems frecuentes, pero representa los datos mediante un FP-Tree (ver fig. 12). Este algoritmo es muy similar al anterior, considerando las estructuras anotadas [Gopalan, 2004].

3.2.2 Métodos basados en agrupamientos de itemsets maximales potenciales

Los métodos basados en estos agrupamientos consideran, directa o indirectamente, subconjuntos de ítems asociados con itemsets potencialmente maximales. Estos grupos inducen sub-retículos y definen sub-árboles de búsqueda. Entre lo algoritmos más significativos de este grupo se tienen los siguientes: Eclat, MaxClique y MAFIA.

3.2.2.1 Algoritmo Eclat

Eclat (*Equivalence Class Transformation,* por sus siglas en inglés) es el método paradigmático de este grupo de algoritmos. Este fue propuesto por Zaki, Parthasarathy, Ogihara y Li considerando el concepto de clase de equivalencia [Zaki, 1997a], [Zaki, 1997b], [Zaki, 1997c].

En el epígrafe 3.1 se planteó el concepto de clase de equivalencia (5). Según esta definición, los *itemsets* de tamaño k pueden particionarse en clases, agrupándolos por iguales prefijos de tamaño (k-1). El algoritmo Eclat desciende recursivamente por cada clase de equivalencia encontrada. A continuación se presenta este algoritmo.

```
Algoritmo: Eclat
1)  L1 = { large 1-itemsets };
2)  F = L1;
3)  Bottom-up(L1, F);
4)  Answer = F;
```

Fig. 22. Pseudocódigo del algoritmo Ecla

```
Procedimiento: Bottom-up
Entrada: Fk – Conjunto de k-itemsets de una
              clase de equivalencia
Entrada/Salida: F – Frequent l-itemsets, l > k
1)   forall itemset Ii ∈ Fk do begin
2)     Fk+1 = ∅;   // Fk+1 will be E(Ii)
3)     forall itemset Ij ∈ Fk and i < j do begin
4)       N = Extend(Ii, Ij);
5)       if N.Support ≥ minsup then
6)         Fk+1 = Fk+1 ∪ {N};
7)     end
8)     if Fk+1 ≠ ∅ then begin
9)       Bottom-up(Fk+1, F);
10)      F = F ∪ Fk+1;
11)    end
12)  end
```

Fig. 23. Procedimiento Bottom-up

El procedimiento *Bottom-up* es recursivo, teniendo como entrada cada clase de equivalencia que se va generando y como salida los *itemsets* frecuentes encontrados.

El descenso se realiza con un procedimiento (Extend) que extiende y calcula el soporte simultáneamente. Este procedimiento Extend(X, Y) opera sobre los *itemsets* X e Y, extendiéndolos mediante la operación: $X \cup Y$. Por otra parte, considerando que la representación

elegida es vertical del tipo VTL, los soportes pueden determinarse con la intersección de las tidList asociadas con X e Y.

Aunque el algoritmo Eclat ha sido ampliamente referenciado, este método tiene como deficiencia la cantidad considerable de *itemsets* procesados que, por no generarse completamente a lo ancho, no pueden podarse por no tenerse todos los frecuentes del mismo nivel. No obstante, se han propuesto implementaciones eficientes de este método. Particularmente, debido a su estrategia descendente y a las características del árbol de búsqueda, Eclat puede encontrar *itemsets* maximales previos que serán supraconjuntos de otros desde donde se descendería posteriormente, evitándose recorridos innecesarios. Basado en esta característica, se han realizado implementaciones que logran el descubrimiento de *itemsets* maximales en menos tiempo que el método Apriori clásico [Borgelt, 2003].

3.2.2.2 Algoritmo MaxClique

El algoritmo *MaxClique* fue propuesto por Zaki *et al.,* en el mismo trabajo antes referenciado [Zaki, 1997a]. El propósito de este método es la búsqueda eficiente de los *itemsets* maximales. En ese mismo trabajo se propusieron otros métodos basados en agrupamientos de maximales potenciales. Ejemplo de estos es el método MaxEclat, una versión menos eficiente pero similar a MaxClique que toma como grupos a las clases de equivalencia. También propusieron el algoritmo Clique, el cual se basa en los conceptos utilizados en MaxClique pero realizando recorridos descendentes, a modo de Eclat, mediante el procedimiento *Botton-up* anteriormente presentado para la búsqueda de todos los *itemsets* frecuentes.

El algoritmo MaxClique, al igual que Clique, realiza los recorridos considerando los subconjuntos de itemsets formados por los agrupamientos de "Cliques de Hipergrafos Uniformes Maximales". Para comprender mejor este algoritmo es necesario introducir una serie de definiciones que sustentan el concepto antes mencionado.

Definición. *Clique de hipergrafo uniforme maximal* – Sea $I = \{i_1, \ldots, i_n\}$ el conjunto de vértices de un hipergrafo, siendo sus aristas (o hiperaristas) el conjunto $H = \{E_1, \ldots, E_n\}$, tal que $Ei \subseteq I$, $E_i \neq \varnothing$ y $\cup E_i = I$. Considerando el concepto de hipergrafo, pueden enunciarse las siguientes definiciones:

- *Hipergrafo simple* – Un hipergrafo simple es aquel que verifica: $E_i \subset E_j \Rightarrow i = j$.
- *Rango de un hipergrafo* – El rango de un hipergrafo está dado por la máxima cardinalidad de sus hiperaristas; o sea, $r(H) = \max_j |E_j|$.
- *Hipergrafo uniforme* – Un hipergrafo es uniforme si todas sus aristas tienen la misma cardinalidad.
- *Hipergrafo r-uniforme* – Un hipergrafo r-uniforme es un hipergrafo uniforme simple de rango r.
- *Sub-hipergrafo inducido por X* – Un sub-hipergrafo inducido por X es un sub-hipergrafo que verifica que $X \subset I$ y $H_X = \{ E_j \cap X \neq \varnothing \mid 1 \leq j \leq n \}$.
- *Hipergrafo completo r-uniforme* – Un hipergrafo completo r-uniforme es un hipergrafo cuyas hiperaristas son todos los subconjuntos de I de tamaño r.
- *Clique de hipergrafo r-uniforme* – Un clique de hipergrafo r-uniforme es un sub-hipergrafo completo r-uniforme. Si el sub-hipergrafo contiene m vértices, entonces el clique se denotará como K^r_m.
- *Clique de hipergrafo maximal* – Un clique de hipergrafo es maximal si no está contenido en ningún otro clique.

Puede observarse que un hipergrafo de rango 2 se corresponde con el conocido concepto de clique maximal[1] de un grafo.

Los conceptos anteriores se introducen teniendo en cuenta lo siguiente. Dado el conjunto L_k de *k-itemsets* frecuentes, para cualquier *m-itemset* frecuente, siendo $m > k$, se cumple que todos sus subconjuntos de tamaño k (o sea, los *k-itemsets* contenidos) tienen que ser frecuentes. Expresando lo anterior en términos de hipergrafos, si cada ítem es un vértice del hipergrafo y cada *k-itemset* frecuente una hiperarista, entonces un *m-itemset* frecuente forma un clique de hipergrafo *k*-uniforme. Es más, el conjunto de cliques de hipergrafos maximales representa una aproximación de un conjunto de *itemsets* maximales potenciales; o sea, todos los *itemsets* maximales están contenidos en el conjunto de vértices de los cliques maximales. Lo anterior se puede precisar mejor con la siguiente propiedad.

Propiedad. Sea H_{L_k} un hipergrafo *k*-uniforme con los vértices I e hiperaristas L_k. Sea C el conjunto de cliques de hipergrafos uniformes maximales en ese hipergrafo; o sea, $C = \{ K^k_m \mid m > k \}$, y sea M el conjunto de los conjuntos de vértices de los cliques en C. Entonces, para todo itemset maximal f, $\exists\, t \in M$, tal que $f \subseteq t$.

Considerando el concepto de clique de hipergrafo uniforme maximal, se propone el algoritmo MaxClique como sigue.

Algoritmo: MaxClique

```
1)  C = { K²_m | each element is a maximal clique };
2)  M = { { X ∈ H(K²_m) } | K²_m ∈ C };
3)  foreach set M_2 ∈ M do
4)    Hybrid(M_2, F);
5)  Answer = "Maximal itemsets in F";
```

Fig. 24. Pseudocódigo del algoritmo MaxClique

[1] Un clique maximal en un grafo representa un subconjunto de vértices de máxima cardinalidad totalmente conectado; o sea, con aristas entre todos los pares de vértices del subconjunto.

Procedimiento: Hybrid
Entrada: M_2 – Conjunto de 2-itemsets de un clique uniforme maximal
Salida: F – Itemsets maximales potenciales

```
 1)  // Top-down phase
 2)  M2 = "M2 sorted in descending order by support";
 3)  N = I1;   // First 2-itemset of sorted M2
 4)  forall itemset Ii ∈ M2 and i > 1 do begin
 5)    F = { N };
 6)    N = Extend(N, Ii);
 7)    if N.Support ≥ minsup then
 8)      S1 = S1 ∪ {Ii};
 9)    else break;
10)  end
11)  S2 = M2 \ S1;
12)  // Bottom-up phase
13)  forall itemset Ii ∈ S2 do begin
14)    M3 = { Yj = Extend(Ii, Xj) | Xj ∈ S1 ∧ Yj.Support ≥ minsup };
15)    S1 = S1 ∪ {Ii};
16)    if M3 ≠ ∅ then Bottom-up(M3, F);
17)  end
```

Fig. 25. Pseudocódigo del procedimiento Hybrid

Como puede observarse, el algoritmo MaxClique itera sobre cada conjunto de 2-*itemsets* pertenecientes a cada clique uniforme maximal, con el propósito de encontrar los posibles itemsets maximales asociados. La elección del conjunto L_2 es una decisión práctica, aunque sus autores proponen analizar otros L_k, con $k > 2$, considerando que mientras k sea mayor el conjunto de *itemsets* maximales potenciales es más preciso. El primer paso del algoritmo es determinar los cliques uniformes maximales. Este proceso es realmente costoso y encarece sensiblemente al algoritmo. Los autores consideraron un método y propusieron analizar otras variantes para optimizar este paso.

Las iteraciones del algoritmo MaxClique las realiza mediante el procedimiento *Hybrid*, el cual sigue la siguiente lógica. Primeramente, se desciende en profundidad por el camino más prometedor para encontrar un *itemset* maximal considerando los 2-*itemsets* más frecuentes. Esto se basa en la intuición de que los *itemsets* más frecuentes deben formar parte del *itemset* más largo. El procedimiento Extend es el mismo del explicado en Eclat.

En la fase final del procedimiento se desciende (*Bottom-up*) por cada clase de equivalencia sobre cada *itemset* no considerado en la primera fase, extendido con cada *itemset* ya procesado. En este algoritmo, los *itemsets* del conjunto S_2 (o sea, los no procesados) representan los prefijos de las clases de equivalencias que se forman al ser extendidos.

La principal ventaja del algoritmo MaxClique es la rápida detección de los *itemsets* maximales. No obstante, si se deseara utilizar este algoritmo para el descubrimiento de reglas de asociación, se necesitaría modificar el mismo ya que este método no determina todos los soportes de los subconjuntos de los *itemsets* maximales.

3.2.2.3 Algoritmo MAFIA

Como su nombre lo indica, el algoritmo MAFIA (*Maximal Frequent Itemset Algorithm,* por sus siglas en inglés) calcula directamente los *itemsets* maximales, denominados MFI (*Maximal*

Frequent Itemsets); los cuales por definición son frecuentes (ver definición en epígrafe 3.1). Este alcanza la mayor eficiencia cuando los *itemsets* en la base de datos son muy grandes, lo cual sucede por lo general en las bases de datos muy densas [Burdick, 2001]. Esta eficiencia se garantiza sobre todo por el hecho de que, al igual que otros algoritmos anteriormente comentados, MAFIA usa una representación VTV de la base de datos. Este algoritmo, además de trabajar con los *itemsets* maximales, opera con los itemsets cerrados.

Para la presentación del algoritmo MAFIA Burdick *et al*., expusieron cinco variaciones, denominadas: Simple, PEP, FUHT, HUTMFI y MAFIA, siendo la última la integración de las anteriores. Estas variaciones recorren en profundidad el árbol de búsqueda. En estas se consideran como cabeza (*head*) al *itemset* que representa cada nodo del árbol, y como cola (*tail*) los posibles ítems sufijos que a cada nodo se le puede adicionar para formar los nodos (*itemsets*) hijos.

Debe notarse que estos conceptos de *head* y *tail* se corresponden con los conceptos de prefijo de la clase de equivalencia (*head*) y los sufijos de los miembros de esa clase (*tail*), o los miembros en si considerando la notación de Zaki *et al.* [Zaki, 1997a]. Además, no es difícil comprender que la variación Simple es una versión del algoritmo Eclat, mientras que FUHT es un algoritmo que recuerda la fase inicial de MaxEclat. Por último, téngase en cuenta que todas las variaciones presentadas son algoritmos recursivos.

Variación Simple

En el algoritmo simple se recorre el árbol lexicográfico primero en profundidad. En un nodo *N*, cada ítem de la cola del nodo es generado y contado como una posible extensión. Si el soporte de *N*.Head ∪ {ítem} es menor que el mínimo soporte, entonces no es necesario seguir extendiéndose por esa rama, según la clausura descendente del soporte. Si ninguna extensión cumple con el soporte entonces el nodo es considerado una hoja (*leaf*).

Cuando se alcanza una hoja en el árbol se está en presencia de un candidato a incluirse en el MFI. Sin embargo, un supraconjunto frecuente del nodo pudiera haber sido insertado en el MFI. Por consiguiente, se necesita chequear por cada *itemset* candidato si existe en MFI un supraconjunto de este. Si no existe un supraconjunto, entonces se agrega el candidato al MFI. Es importante notar que, con este recorrido en profundidad, nunca se tendrán que eliminar elementos del MFI debido a que los *itemsets* son insertados en orden lexicográfico; similar criterio fue considerado por Borgelt en su experimentación con Eclat [Borgelt, 2003].

Algoritmo: Simple
Entrada: *C* – Nodo actual
Entrada/Salida: *MFI* – Conjunto de itemsets maximales

```
1)  foreach ítem i ∈ C.Tail do begin
2)    newNode = new Node(C.Head ∪ {i}) ;
3)    if "newNode.Head is frequent" then
4)      Simple(newNode, MFI)
5)  end
6)  if "C is a leaf" and "C.Head is not in MFI" then
7)    MFI = MFI ∪ {C.Head};
```

Fig. 26. Pseudocódigo del algoritmo Simple

Variación Parent Equivalent Pruning (PEP)

Esta variación compara los conjuntos de transacciones de cada par padre-hijo. Sea X la cabeza del nodo N y Y un elemento en la cola, si se verifica la condición $D(X) \subseteq D(Y)$ entonces cualquier transacción que contenga a X también contendrá a Y. Si esta condición se cumple entonces se garantiza que para cualquier *itemset* frecuente Z que contenga a X pero no a Y, su supraconjunto ($Z \cup Y$) es frecuente. Como se buscan los *itemsets* maximales, si los cubrimientos del nodo cabeza y su hijo son iguales, entonces puede podarse el ítem sufijo del hijo de la cola del padre moviendo el ítem de la cola a la cabeza.

Algoritmo: PEP
Entrada: *C* – Nodo actual
Entrada/Salida: *MFI* – Conjunto de itemsets maximales

```
 1)  foreach ítem i ∈ C.Tail do begin
 2)    newNode = new Node(C.Head ∪ {i});
 3)    if newNode.Support = C.Support then
 4)      "Move i from C.Tail to C.Head"
 5)    else
 6)      if "newNode.Head is frequent" then
 7)        PEP(newNode, MFI);
 8)  end
 9)  if "C is a leaf" and "C.Head is not in MFI" then
10)    MFI = MFI ∪ {C.Head};
```

Fig. 27. Pseudocódigo del algoritmo PEP

Variación Frequent Head Union Tail (FHUT)

Este método se basa en que el mayor itemset frecuente posible en el subárbol de raíz N es su HUT (*Head Union Tail*; o sea, cabeza unión cola). Si el HUT es frecuente entonces no es necesario explorar ningún subconjunto del HUT y se puede podar el subárbol completo.

El método FHUT explora primero el camino más a la izquierda de cada nodo. Este método tiene como desventaja que el camino más a la izquierda contiene la mayor cantidad de ítems entre todos los caminos del sub-árbol, y en el momento de hacerse la poda ya fue generada y contada una gran cantidad de *itemsets*.

Algoritmo: FHUT
Entrada: *C* – Nodo actual
IsHUT – Bandera que indica si el elemento a agregar es el más a la izquierda
Entrada/Salida: *MFI* – Conjunto de itemsets maximales

```
1)  foreach ítem i ∈ C.Tail do begin
2)    newNode = new Node(C.Head ∪ {i}) ;
3)    IsHUT = "whether i is the leftmost child in the tail"
4)    if "newNode.Head is frequent" then
5)      FHUT(newNode, MFI, IsHUT) ;
6)  end
7)  if "C is a leaf" and "C.Head is not in MFI" then
8)    MFI = MFI ∪ {C.Head};
9)  if IsHUT and "All extensions are frequent" then
10)   "Stop exploring this subtree and go back up tree to
       when IsHUT was changed to True";
```

Fig. 28. Pseudocódigo del algoritmo FHUT

Variación Head Union Tail in MFI (HUTMFI)

Hay dos métodos para determinar si un *itemset X* es frecuente. Uno de ellos es contando directamente el soporte de *X*, el otro es chequeando si algún supraconjunto de *X* ya ha sido declarado frecuente; FHUT usa el primer método. En el método HUTMFI se procede de la segunda forma. En este se determina si un supraconjunto de HUT está en el MFI; si un supraconjunto existe entonces el HUT tiene que ser frecuente y el sub-árbol correspondiente puede ser podado.

Algoritmo: HUTMFI
Entrada: *C* – Nodo actual
Entrada/Salida: *MFI* – Conjunto de itemsets maximales

```
1)  HUT = C.Head ∪ C.Tail;
2)  if "HUT is in MFI" then
3)    "Stop searching and return";
4)  foreach ítem i ∈ C.Tail do begin
5)    newNode = new Node(C.Head ∪ {i});
6)    if "newNode.Head is frequent" then
7)        HUTMFI(newNode, MFI);
8)  end
9)  if "C.Head is not in MFI" then
10)   MFI = MFI ∪ {C.Head};
```

Fig. 29. Pseudocódigo del algoritmo HUTMFI

Variación MAFIA

En la siguiente figura se presenta el pseudocódigo del algoritmo MAFIA, el cual integra los métodos anteriores.

Algoritmo: MAFIA
Entrada: *C* – Nodo actual
IsHUT – Bandera que indica si el elemento a agregar es el más a la izquierda
Entrada/Salida: *MFI* – Conjunto de itemsets maximales

```
 1)  HUT = C.Head ∪ C.Tail;
 2)  if "HUT is in MFI" then
 3)    "Stop searching and return";
 4)  "Count all children, use PEP to trim the tail,
     and reorder by increasing support"
 5)  foreach ítem i ∈ C.Trimmed_Tail do begin
 6)    IsHUT = "whether i is the first item in the tail";
 7)    newNode = new Node(C ∪ {i});
 8)    MAFIA(newNode, IsHUT, MFI);
 9)  end
10)  if IsHUT and "All extensions are frequent" then
11)    "Stop search and go back up subtree";
12)  if "C is a leaf" and "C.Head is not in MFI" then
13)    MFI = MFI ∪ {C.Head};
```

Fig. 30. Pseudocódigo del algoritmo MAFIA

A igual que MaxClique, la principal ventaja del algoritmo MAFIA es la rápida detección de los *itemsets* maximales. No obstante, de forma similar, si se deseara utilizar este algoritmo para el descubrimiento de reglas de asociación, se necesitaría su modificación para determinar todos los soportes de los subconjuntos de los *itemsets* maximales.

El algoritmo logra su mayor efectividad en bases de datos muy densas, en las cuáles la cantidad de itemsets frecuentes es muy grande y en donde la mayoría de estos tienen supraconjuntos frecuentes insertados en el MFI.

3.3 Métodos ascendentes

En el epígrafe 3.1 se clasificaron los métodos de generación de los *itemsets* frecuentes en descendentes y ascendentes, atendiendo a la dirección del recorrido del árbol de búsqueda. En el caso de los métodos ascendentes, sus recorridos se realizan desde supraconjuntos de los *itemsets* fronteras (o maximales) hasta los *itemsets* fronteras (o maximales), pudiendo efectuarse tanto en profundidad como a lo ancho. En la bibliografía consultada, el único método encontrado con estas características es el algoritmo *TopDown*, el cual se explica a continuación.

3.3.1 Algoritmo TopDown

El algoritmo *TopDown* fue propuesto por Zaki *et al.,* junto a los métodos Eclat, MaxEclat, Clique y MaxClique [Zaki 1997b]. Al igual que estos, *TopDown* considera agrupamientos de maximales potenciales, en particular de los asociados con los cliques de hipergrafos uniformes maximales.

A diferencia de los métodos Eclat, MaxEclat, Clique y MaxClique, que descienden por los sub-retículos restringidos por los grupos definidos, el algoritmo TopDown asciende[2] en cada sub-retículo a partir del *itemset* maximal potencial asociado. Esta ascensión se realiza recursivamente en profundidad. A continuación se presenta este algoritmo.

Algoritmo: TopDown

```
1)  C = { K²m | each element is a maximal clique };
2)  Imax = { ⋃ Ij | K²m ∈ C };
           Ij∈H(K²m)
3)  HT = ∅;
4)  F = ∅;
5)  foreach itemset X ∈ Imax do
6)    Top-Down(X, HT, F);
7)  Answer = "Maximal itemsets in F";
```

Fig. 31. Pseudocódigo del algoritmo TopDown

Procedimiento: Top-Down
Entrada: X – Itemset maximal potencial
Entrada/**Salida**:
HT – Hash table con itemsets infrecuentes
$F = \{F_2, \dots\}$ – Lista de conjuntos k-itemsets maximales potenciales

```
 1)  k = |X|;
 2)  if X ∉ Fk then begin
 3)    Intersect(X);
 4)    if X.Support ≥ minsup then
 5)      Fk = Fk ∪ {X}
 6)    else
 7)      if k > 3 then
 8)        forall (k-1)-itemset Y ⊂ X do
 9)          if Y ∉ HT then begin
10)            Top-Down(Y, HT, F);
11)            if Y.Support < minsup then
12)              HT = HT ∪ {Y};
13)          end
14)  end
```

Fig. 32. Pseudocódigo del proecedimiento Top-Down

El procedimiento "Intersect" permite calcular el soporte de un *itemset.* Este cálculo se realiza considerando que la representación elegida para los *itemsets* es del tipo VTL, por lo que los soportes pueden determinarse con la intersección de las tidList asociadas con cada ítem contenido. Teniendo en cuenta que la búsqueda es ascendente y pudieran examinarse reiteradamente algunos *itemsets*, se propone utilizar una tabla de *hash* (*HT*) para controlar los

[2] El nombre TopDown se deriva de la orientación natural de un retículo (conjunto universo en el extremo superior y el vacío en el inferior); por su parte, nuestra denominación ascendente se ha tomado de la orientación del árbol de búsqueda en la mayoría de los métodos existentes cuyas raíces son los 1-itemsets.

itemsets no frecuentes ya analizados. Adicionalmente, se tiene una lista (F) de conjuntos de *k-itemsets* detectados como maximales potenciales.

La principal desventaja de este algoritmo es su costo computacional. Como puede observarse, este precisa de una primera fase donde se calcula L_2, y posteriormente el conjunto C de cliques maximales. Adicionalmente, si se requiriera determinar los soportes de todos los *itemsets* frecuentes, la información procesada al ascender hasta la frontera o borde no puede ser aprovechada, encareciendo aún más el costo computacional del método. Además, el cálculo del soporte por intersección de k ítems (procedimiento Intersect) introduce una sobrecarga importante respecto a la clásica intersección de 2 ítems.

En la experimentación realizada en el trabajo referenciado, Zaki *et al*., señalaron que el algoritmo *TopDown* tuvo un comportamiento similar a Eclat y considerablemente inferior a MaxClique. No obstante lo anterior, este algoritmo es sólo una variante de un esquema ascendente, no descartándose la posibilidad de lograrse otras con mejores desempeños, quedando abierta su investigación.

3.4 Algoritmo CBMine

De forma general, el algoritmo CBMine (*Compressed Binary Mine*), desarrollado por varios de los autores de este reporte, sigue la lógica del algoritmo Apriori de Agrawal *et al.*, incluyendo la heurística que este utiliza. Sin embargo, a diferencia de este, CBMine no requiere mantener cada conjunto de *itemsets* candidatos (C_k) completo en memoria, ya que las representaciones y estrategias de cálculo del soporte empleados permiten optimizaciones de estos procesos.

En una versión preliminar (ver. 1.0) del CBMine se propuso un método que se ajustaba plenamente a la lógica del Apriori clásico, incluyendo a su vez algunos de los problemas de este algoritmo, los cuales fueron resueltos en la siguiente versión (ver. 1.1) que ha sido divulgada a la comunidad internacional.

3.4.1 Versión 1.0 del CBMine

En este algoritmo se procesa inicialmente la base de datos eliminando los ítems que no son frecuentes, obteniéndose un conjunto de ítems frecuentes (también conocidos como ítems únicos), y un conjunto de transacciones "filtradas" cuyo tamaño es sustancialmente menor que la base original.

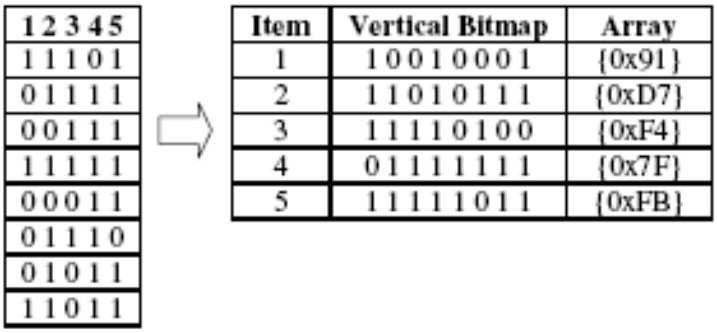
El conjunto de las transacciones filtradas pudiera ser representada como una matriz binaria (por transacción) de m x n, donde m es el total de transacciones y n la cantidad de ítems filtrados (ver Fig. 33 para una matriz de 8x5).

Tid	Items
1	1 2 3 5
2	2 3 4 5
3	3 4 5
4	1 2 3 4 5
5	4 5
6	2 3 4
7	2 4 5
8	1 2 4 5

1 2 3 4 5
1 1 1 0 1
0 1 1 1 1
0 0 1 1 1
1 1 1 1 1
0 0 0 1 1
0 1 1 1 0
0 1 0 1 1
1 1 0 1 1

Fig. 33. Ejemplo de matriz binaria por transacción

Sin embargo, para optimizar los procesos de cálculo del soporte se decidió por una representación por ítem de esa matriz (Fig. 34); o sea, del tipo VTV. En este caso, las secuencias binarias de cada ítem se almacenan como arreglos de números enteros, dependientes de la longitud de la palabra del procesador (en las versiones realizadas en palabras de 32 *bits*).



1 2 3 4 5
1 1 1 0 1
0 1 1 1 1
0 0 1 1 1
1 1 1 1 1
0 0 0 1 1
0 1 1 1 0
0 1 0 1 1
1 1 0 1 1

Item	Vertical Bitmap	Array
1	1 0 0 1 0 0 0 1	{0x91}
2	1 1 0 1 0 1 1 1	{0xD7}
3	1 1 1 1 0 1 0 0	{0xF4}
4	0 1 1 1 1 1 1 1	{0x7F}
5	1 1 1 1 1 0 1 1	{0xFB}

Fig. 34. Ejemplo de matriz binaria por ítem (palabras de 8 bits)

En el algoritmo se mantiene en memoria un arreglo "frontera", que no es más que un arreglo con los conjuntos frecuentes de orden *k*; a partir de los cuales se genera cada candidato, y posteriormente frecuente de orden *k*+1. Los candidatos se generan en orden lexicográfico lo cual permite buscar dicotómicamente en el proceso de poda. El conteo de los soportes se realiza mediante operaciones de AND lógico sobre las columnas de los ítems presentes en el conjunto candidato. Cuando para dos ítems dados *a* y *b* se realiza bloque a bloque la operación lógica AND, se obtiene como resultado la intersección de las respectivas columnas, por tanto la cantidad total de bits activos (cantidad de unos) será igual al soporte del conjunto {*a*, *b*}. El algoritmo culmina cuando a partir de un arreglo frontera no se genere ningún conjunto frecuente.

A continuación se muestra el pseudocódigo correspondiente.

Algoritmo: CBMine-v1.0

```
 1)  L_1 = {large 1-itemsets};
 2)  w = word size;
 3)  q = [m/w];
 4)  forall transaction t_i ∈ D do begin
 5)    s = [i/w];
 6)    forall filtered item j ∈ t_i do
 7)      W_{z,j} = W_{z,j} + 2^{(w-1)-(i-1) mod W}; // Encode database
 8)  end
 9)  for ( k = 2; L_{k-1} ≠ ∅; k++ ) do begin
10)    C_k = apriori_gen(L_{k-1}); // New candidates
11)    forall candidate c ∈ C_k do
12)      for ( s:=1; s ≤ q; s++) do begin
13)        forall item j ∈ c do
14)          A = A and W_{z,j};
15)          c.Support += Bitcount(A); //Hamming weigth of A
16)      end
17)    L_k = {c ∈ C_k | c.Support ≥ minsup};
18)  end
19)  Answer = ⋃_k L_k;
```

Fig. 35. Pseudocódigo del algoritmo CBMine versión 1.0

De forma general, el algoritmo presenta algunas desventajas importantes:

1. Aunque los conjuntos candidatos se generan en orden lexicográfico hay que chequear, en el nivel *k*, si los conjuntos frecuentes del nivel *k*-1 coinciden en los *k*-2 primeros ítems.
2. Se recalculan varias veces los mismos AND; por ejemplo, para determinar que el conjunto {*a*, *b*, *c*} es frecuente se realiza el AND de las columnas correspondientes a los ítems *a*, *b* y *c* respectivamente. Más adelante, en el próximo nivel, para determinar si el conjunto {*a*, *b*, *c*, *d*} es frecuente se necesita recalcular el AND de las columnas correspondientes a los ítems *a*, *b*, *c* y la columna correspondiente al ítem *d*.
3. A medida que aumentan los niveles aumenta la cantidad de bloques de la matriz compactada que son iguales a cero, manteniéndose ocupada memoria ociosa y haciéndose más lento el conteo del soporte.

3.4.2 Versión 1.1 del CBMine

En esta versión del algoritmo se le dio solución a las desventajas planteadas con anterioridad y se obtuvieron mejores resultados en cuanto a los tiempos de ejecución. A continuación se explican en detalles las mejoras realizadas [Hernández, 2005].

Para solucionar la primera desventaja se concibió una nueva estructura, la cual recuerda al *Hash Tree* de Agrawal *et al.* [Agrawal, 1994], pero sobre un arreglo y no para contar el soporte de los conjuntos candidatos sino para almacenarlos. El *Hash Tree* original tiene dos tipos de nodos: nodos internos y nodos hojas. Los nodos internos tienen una tabla de *hash* a partir de la

cual, de acuerdo al ítem que entre, se decide la rama del árbol que se toma. En los nodos hojas hay una lista de referencias a los conjuntos candidatos que tienen como prefijo común el camino desde la raíz hasta la hoja. De esta forma, en ese algoritmo, para cada iteración se construye un *Hash Tree* y se recorre la base de datos tomando por cada transacción los *k-itemsets* candidatos existentes e incrementando su soporte. Como se puede ver, en cada hoja se encuentran referencias a los conjuntos de ítems con los *k*-2 primeros ítems iguales.

La estructura de datos definida en esta versión para almacenar los *k*-itemsets es una lista de tripletas <*Prefix*, *CA*, *Suffixes*> asociadas con las clases de equivalencias existentes en L_k, conteniendo:

- *Prefix*: El (*k*-1)-*itemset*, prefijo idéntico de varios *k-itemsets* frecuentes, los que son miembros de la clase de equivalencia definida por el prefijo.
- *CA*: Una estructura tipo VTV pero comprimida asociada con el prefijo, la cual se explica posteriormente.
- *Suffixes*: Lista no vacía de los ítems sufijos de los *k-itemsets* frecuentes de la clase de equivalencia.

Esta estructura recuerda a la utilizada por MAFIA para representar cada nodo del árbol y su recorrido descendente. En MAFIA se asume también una tripleta similar a esta (aunque no planteado explícitamente) para representar cada nodo del árbol. En este método se utilizaron los conceptos de *head* (para el prefijo) y *tail* (para los sufijos). Incluso, la estructura incluye una representación tipo VTV asociada con el prefijo. Sin embargo, aparte de que en MAFIA sólo se representan listas independientes para cada clase de equivalencia y en CBMine listas de *itemsets* a todo lo ancho de cada nivel, la lista de MAFIA representa a los *itemsets* del nivel asociados con el prefijo (*head*), mientras que CBMine representa a los *itemsets* del nivel asociados con prefijo $\cup$ {ítem sufijo}. Por tales razones, la cola (*tail*) de MAFIA son todos los posibles ítems sufijos de la clase de equivalencia, mientras que en CBMine son sólo los frecuentes. Además, los vectores binarios en CBMine se asocian con el prefijo de la clase de equivalencia y no con cada elemento de ella, como sucede con MAFIA, lográndose una economía significativa en la memoria requerida.

Para solucionar la segunda desventaja de la versión anterior, se optó por la estructura ya presentada. En particular, al almacenar para los *k-itemsets* de la misma clase de equivalencia sólo el vector binario (comprimido) del prefijo, se optimiza el proceso del paso 13 de la versión 1.0 de CBMine.

La tercera desventaja se solucionó con una nueva forma de compresión de los *bitmaps* verticales, denominada “lista de enteros comprimida” (*Compressed Integer-List*). Esta estructura es una lista *CA* de pares <s, B_s>, con $B_s \neq 0$, siendo s la posición del entero en la lista no comprimida y B_s el entero no nulo correspondiente. Vale aclarar que en todo momento se mantiene en memoria las listas no comprimida de los ítems frecuentes para lograr mayores eficiencias en cuanto al tiempo de procesamiento.

En este nuevo algoritmo se mantuvo el orden lexicográfico y la búsqueda dicotómica, adaptándola a la nueva estructura usada para almacenar los *itemsets*. A continuación se presenta el pseudocódigo del algoritmo.

Algoritmo: CBMine-v1.1

```
1)  L1 = { large 1-itemsets }; // Scanning the database
2)  PL2 = { < Prefix1, CA_Prefix1, Suffixes_Prefix1 > };
3)  for( k = 3; PL_k-1 ≠ ∅; k++ ) do
4)    forall <Prefix, CA, Suffixes> ∈ PL_k-1 do
5)      forall item j ∈ Suffixes do begin
6)        Prefix' = Prefix ∪ {j};
7)        CA' = CompactAnd(CA, I_j);
8)        forall j' ∈ Suffixes and j' > j do
9)          if Prune(Prefix' ∪ {j'}, PL_k-1) and
               Support(CompactAnd(CA', I_j')) ≥ minsup
10)         then Suffixes'= Suffixes' ∪ {j'};
11)       if Suffixes' ≠ ∅
12)       then PL_k = PL_k ∪ {<Prefix', CA', Suffixes'>};
19)     end
20) Answer = ⋃k L_k; // L_k is obtained from PL_k
```

Fig. 36. Pseudocódigo del algoritmo CBMine versión 1.1

El procedimiento CompactAnd permite realizar la intersección entre una lista de enteros comprimida y una no comprimida, obteniendo la lista comprimida correspondiente.

3.4.3 Resultados experimentales

A continuación se presenta los resultados obtenidos con la implementación de la versión 1.1 del CBMine. Estos resultados fueron comparados con el desempeño de otros algoritmos eficientes que generan *itemsets* frecuentes, como son MIHP, FP-Growth [Bodon, 2006], MAFIA [Calimlim, 2006] y Patrica Trie-Mine[3].

El algoritmo MIHP empleado fue una implementación realizada por los autores, ya que no fue posible acceder a alguna versión desarrollada por Holt.

En el caso de MAFIA, este tiene como parámetros de ejecución los siguientes: -FI, -MFI; el primero para calcular todos los itemsets frecuentes y el segundo para calcular sólo los *itemsets* maximales. Es válido esclarecer que la opción -MFI tiene un costo computacional mucho menor; no obstante, para aplicar el algoritmo para el descubrimiento de las reglas de asociación se debe usar la opción –FI para generar todas las reglas representativas. En el caso de la opción -FI de MAFIA, los resultados no fueron ploteados dado que el tiempo de ejecución es mayor que el de CBMine.

Los experimentos fueron desarrollados con cuatro bases de datos: Kosarak, Webdocs, Reuters y TDT. La base de datos Kosarak fue suministrada por Ferenc Bodon al repositorio de FIMI (ver Ref. 3 al pie de página) y contiene (anónimos) datos de flujos de cliqueo de un portal húngaro de noticias. Webdocs fue construido de una colección de documentos HTML de la Web, donado por Claudio Lucchese *et al.,* a FIMI (ver Ref. 3 al pie de página). Reuters fue originalmente coleccionado y etiquetado por Carnegie Group, Inc. y Reuters ltd. TDT contiene

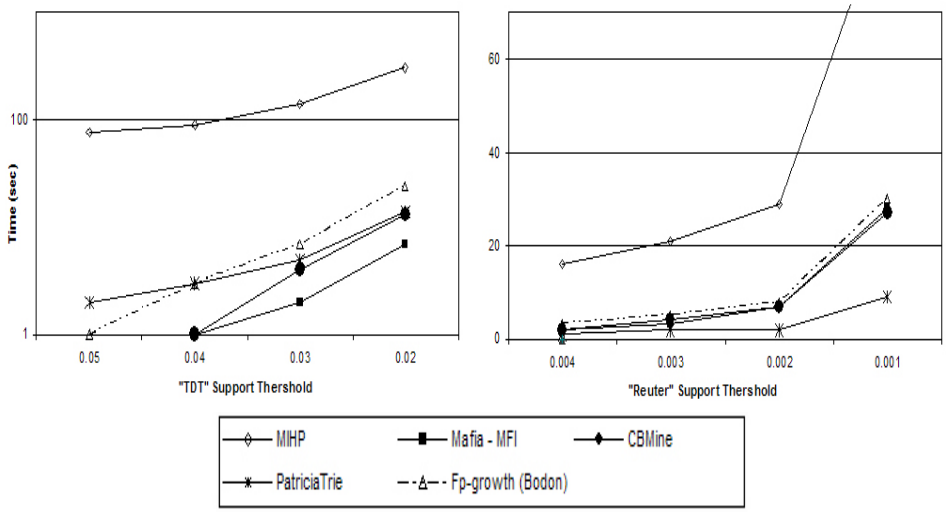
[3] FIMI- repository. (2003) http://fimi.cs.helsinki.fi/src

noticias recogidas diariamente de seis fuentes en inglés por un período de seis meses (enero a junio de 1998). Reuters y TDT le fueron suprimidas las palabras nulas (*stop-words*) y lematizadas mediante *TreeTagger*[4].

Table 2. Características principales de las bases de datos

	Webdocs	Kosarak	Reuters	TDT
Transacciones	1704140	990007	18146	8169
Ítems	5266562	41935	13915	55532
Promedio de ítems por transacción	175.98	8.1	21.2	133.5

Estas experimentaciones fueron realizadas en una PC a 2.66 GHz, Intel P4 y 512 Mbytes de RAM. El sistema operativo empleado fue Windows XP. Los tiempos de ejecución fueron obtenidos usando funciones de C Standard. En este trabajo, los tiempos de ejecución incluyen tanto el tiempo del CPU como el tiempo de E/S.



[4] http://www.ims.uni-stuttgart.de/projekte/corplex/Tree Tagger/ DecisionTreeTagger .html

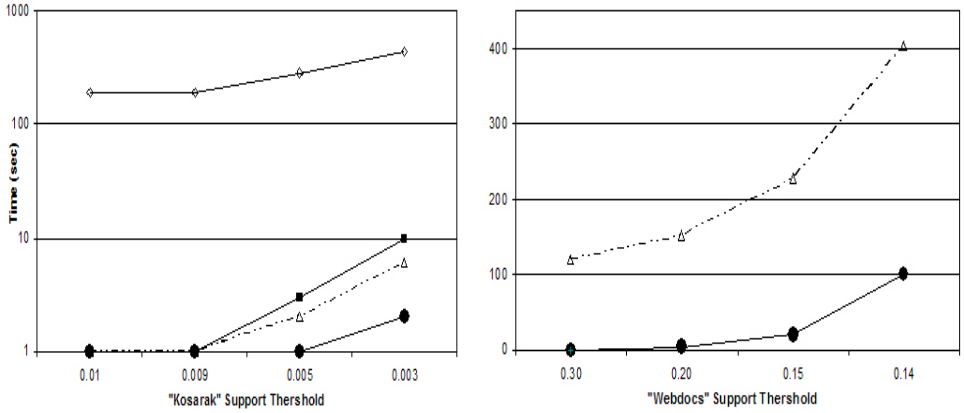


Fig. 37. Comparación de los tiempos de ejecución

Como muestran los gráficos, CBMine tiene el mejor rendimiento. Como una observación particular, MAFIA –MFI tiene tiempos de ejecución muy similar que CBMine; sin embargo, con –FI para calcular todos los *itemsets* frecuentes este tiene un costo computacional mayor. El algoritmo *Patricia Trie-Mine* muestra un buen desempeño en tres de las bases más pequeñas pero en las grandes, como Kosarak, el tiempo de ejecución es mayor que una hora. En el caso de Webdocs, fue posible sólo correr CBMine y FP-Growth, siendo CBMine el mejor. Cuando el tiempo de ejecución de un algoritmo fue muy grande o produjo desbordamiento de memoria, sus resultados no fueron ploteados en las gráficas.

4. Métodos paralelos de generación de itemsets

Desde su consolidación como método de la Minería de Datos, el descubrimiento de Reglas de Asociación es uno de los campos más prolíferos en el desarrollo de algoritmos computacionales, tanto secuenciales como paralelos. Simultáneamente con el proceso evolutivo de dichos algoritmos, se han ampliado las posibilidades de aplicación de las Reglas de Asociación y han aumentado también los volúmenes de información de las bases de datos que se desean minar, esto último ha traído como consecuencia que aún usando los algoritmos secuenciales más eficientes, sea prácticamente imposible la reducción del soporte a probar sin que se produzca una explosión combinatoria de las reglas que se obtienen y sin que sea posible obtener los resultados en un tiempo relativamente breve.

La situación antes mencionada reafirma la importancia de la aplicación de la Computación Paralela para el minado de Reglas de Asociación, área de investigación que se mantiene muy activa a nivel internacional. En este epígrafe presentamos un resumen de los principales algoritmos paralelos para el minado de Reglas de Asociación.

Los principales desafíos de la Computación Paralela son: el balanceo de la carga, la minimización del costo de la comunicación inter-procesadores, la reducción de los requerimientos de sincronización de los procesadores y el uso efectivo de la memoria de la que dispone cada procesador. Estos aspectos deben ser también tomados en cuenta en el desarrollo

de algoritmos eficientes para el minado de Reglas de Asociación, referencias básicas a considerar lo constituyen: [Freitas, 1998], [Freitas, 1998a] y [Skillicorn, 1999].

Sin pretender ser absolutos, puede afirmarse que hasta el presente la gran generalidad de los algoritmos paralelos para el minado de Reglas de Asociación, están basados en el algoritmo secuencial Apriori. Un excelente estudio hecho por Zaki en 1999 [Zaki, 1999] clasifica diferentes algoritmos propuestos hasta ese año, según la estrategia de balanceo de la carga utilizada, la arquitectura y el tipo de paralelismo empleado. Otras referencias importantes lo constituyen los trabajos: [Agrawal, 1996], [Kumar, 1997], [Zaki1, 997], [Zaki, 1997a], [Zaki, 2001] y [Zaiane, 2001].

La tarea de Minería de Datos conocida por el nombre de Reglas de Asociación puede ser descompuesta en dos pasos. El primero consiste en encontrar todos los *itemsets* frecuentes. El segundo paso consiste en la formación de reglas tipo implicación entre los *itemsets* frecuentes hallados en el primer paso, con una confianza especificada por el usuario. Dado que el segundo paso no es intensivo en cuanto a cálculo, el primer paso es el objetivo principal de los investigadores y como hemos referido la base fundamental de los algoritmos desarrollados lo constituye el algoritmo Apriori, el cual fue tratado en la sección 4.1.1.

4.1 Algoritmos paralelos para determinar los itemsets frecuentes

Para generar Reglas de Asociación en paralelo, se necesita tener en cuenta algunas relaciones como por ejemplo: partes del algoritmo con mayor costo computacional, comunicación entre los procesos que pueden ser ejecutados de forma concurrente, uso de la memoria, sincronización y el empleo de información específica del problema.

El algoritmo PDM, propuesto en [Park, 1995], trata de paralelizar el DHP. Cada nodo calcula los *itemsets* frecuentes globales intercambiando sus conteos de soporte de los conjuntos candidatos. Con el fin de aplicar la técnica de *hashing*, todos los nodos tienen que enviar el resultado de su correspondiente tabla *hash*. En este algoritmo se propone una técnica para reducir el número de mensajes consistente en lo siguiente: cada nodo selecciona y le comunica a los restantes, aquellas celdas de la tabla de *hash* cuyo contador total es mayor que un cierto umbral, de esta forma no todas las entradas de la tabla de *hash* tienen que ser enviadas.

En [Agrawal, 1996], fueron propuestos tres algoritmos para el minado de RA en un sistema multiprocesador del tipo nada-compartido o de memoria distribuida y fueron denominados: *Count Distribution* (CD), *Data Distribution* (DD) y *Candidate Distribution* (CDD). Los tres algoritmos antes mencionados han constituido siempre un modelo de referencia para los investigadores que desarrollan algoritmos paralelos para el minado de Reglas de Asociación, de ahí que por su importancia serán expuestos en detalle más adelante.

En [Han, 1997], fueron propuestos dos algoritmos para el minado de Reglas de Asociación en paralelo. Uno es el *Intelligent Data Distribution* (IDD), y el otro es el *Hybrid Distribution* (HD). El IDD usa eficientemente la memoria compartida de la computadora paralela empleando para ello un esquema de particionamiento inteligente de los candidatos y usa mecanismos eficientes de comunicación para el movimiento de datos entre procesadores. El HD particiona dinámicamente el conjunto de candidatos para mantener un buen balanceo de carga.

Otro algoritmo, denominado DMA (*Distributed Mining of Association Rules,* por sus siglas en inglés), para minado de reglas de asociación en bases de datos distribuidas fue propuesto en [Cheung, 1996b]. DMA genera un número pequeño de conjuntos candidatos y sólo requiere

O(n) mensajes para intercambio del contador de soporte para cada conjunto candidato, donde n es el número de emplazamientos en una base de datos distribuida.

DMA fue implementado inicialmente sobre un sistema distribuido basado en red, y fue mejorado con la versión paralela denominada FPM (*Fast Parallel Mining,* por sus siglas en inglés) implementada sobre computadora paralela IBM SP2 [Cheung, 1998]. FPM es una mejora del algoritmo CD. Se mantiene el esquema simple de intercambio de conteos de soporte de CD. La principal diferencia en FPM es la incorporación de la poda distribuida y global para reducir el tamaño del conjunto candidato.

El CCPD (*Common Candidate Partitioned Database*, por sus siglas en inglés) presentado en [Zaki, 1996] implementa el algoritmo CD sobre una SGI *Power Challenge* de memoria compartida con algunas mejoras. En este algoritmo se proponen varias técnicas para generar y contar a los candidatos de forma eficiente en un ambiente de memoria compartida. En este algoritmo los *itemsets* frecuentes son agrupados en clases de equivalencia determinadas por un prefijo común (generalmente el primer ítem) y genera los candidatos a partir de cada clase de equivalencia. Obsérvese que el agrupamiento de los *itemsets* frecuentes podrá no reducir el número de candidatos pero si reduce el tiempo para su generación. Se introduce también un método de chequeo denominado "*short-circuited subset*" para realizar el conteo eficientemente de los candidatos en cada transacción.

El algoritmo HPA (*Hash-based Parallel mining of Association rules,* por sus siglas en inglés) [Shintani, 1996], utiliza una técnica de *hashing* para distribuir los candidatos entre los diferentes procesadores, es decir, cada procesador emplea la misma función *hash* para determinar los candidatos asignados a él. En el conteo del soporte, el algoritmo HPA, a diferencia del *Candidate Distribution,* no replica selectivamente la base de datos, en su lugar lo que hace es comunicar al procesador que corresponda el subconjunto de los *itemsets* de tamaño k obtenidos de cada una de las transacciones de su base de datos local, aplicando la misma función con la que fueron distribuidos los candidatos. Este paso se realiza de la siguiente manera, si el ID de un k-*itemset* se corresponde con el de su procesador incrementa el contador en su tabla *hash*, así como el de aquellos candidatos notificados por otros procesadores, mientras que comunica al procesador que corresponda los candidatos que no les pertenecen. De esta forma el subconjunto de *itemsets* obtenidos de una transacción se le comunica sólo a un procesador en lugar de a todos los procesadores. Un procesador llamado coordinador es el responsable de realizar el conteo global, determinar el conjunto L_k y comunicárselo a todos los procesadores para la siguiente iteración. En el mismo artículo [Shintani, 1996] se presenta una versión mejorada de HPA denominada HPA-ELD (HPA *with Extremely Large Itemsets Duplication*), que permite alcanzar un mejor balanceo de la carga entre todos los procesadores.

En [Zaki, 1997a], [Zaki, 1997b], [Zaki, 1997c] se presenta el método Eclat. Este algoritmo utiliza un esquema de agrupamiento de los *itemsets* en clases de equivalencia y los particiona en subconjuntos disjuntos entre los procesadores. Al mismo tiempo aplica una forma de representación vertical de la base de datos y la replica selectivamente de forma tal que cada procesador posee la porción de la base que necesita para realizar sus cálculos. Después de la fase inicial el algoritmo elimina la necesidad de posterior comunicación o sincronización. Posteriormente el algoritmo escanea la partición local de la base de datos tres veces minimizando así el sobrecosto de las operaciones de entrada/salida. A diferencia de otros algoritmos que lo antecedieron, Eclat utiliza operaciones simples de intersección para calcular los *itemsets* frecuentes y no emplea complejas estructuras de tablas de *hash*. La principal deficiencia de éste algoritmo está en encontrar la heurística apropiada para lograr un balanceo

adecuado de la carga entre los procesadores a partir del particionamiento del conjunto L_2, pues como se conoce las clases de equivalencia no poseen la misma cardinalidad.

Posteriormente en [Zaki, 1997d] se presenta un conjunto de algoritmos con diferentes esquemas de particionamiento y conteo de los *itemsets* candidatos. Al igual que Eclat, todos ellos asumen una represtación vertical de la base de datos (tidList por cada item), lo cual facilita la operación de intersección de los TID de los ítems que conforman un *itemset* La base de datos es duplicada de forma selectiva para reducir la sincronización. Dos de los algoritmos (Par-Eclat y Par-MaxEclat) se basan en las clases de equivalencia formadas a partir del primer ítem de los candidatos, mientras que los otros dos (Par-Clique y Par-MaxClique) utilizan el clique de hipergrafo maximal para particionar los candidatos. En este caso, un vértice del hipergrafo se corresponde con un ítem y una arista entre k vértices corresponde a un *itemset* conteniendo los ítems asociados con los k vértices; por lo tanto, un clique es un subgrafo donde todos sus vértices están conectados. Una característica de los algoritmos (Par-MaxEclat y Par-MaxClique) es que pueden determinar los *itemsets* maximales.

En [Shintani 1998] se propone un algoritmo paralelo para el minado de reglas de asociación con jerarquía de clasificación denominado HPGM (*Hierarchical Hash Partitioned Generalized association rule Mining*). En este algoritmo se utiliza completamente el espacio de memoria disponible identificando la ocurrencia frecuente de itemsets candidatos y copiándolos a todos los procesadores, teniendo en cuenta qué *itemsets* frecuentes pueden procesarse localmente sin ninguna comunicación. De esta forma, se puede reducir efectivamente la asimetría de carga entre procesadores.

4.1.1 Algoritmos CD, DD y CDD

En [Agrawal, 1996], Rakesh Agrawal y John Shafer describieron tres algoritmos paralelos para obtener todos los *itemsets* frecuentes y un algoritmo paralelo para la generación de reglas a partir de los *itemsets* frecuentes. Los algoritmos asumieron una arquitectura de tipo nada-compartido o de memoria distribuida, donde cada uno de los N procesadores tiene memoria y disco propios. Los procesadores están conectados por una red de comunicación y sólo pueden comunicarse a través de pase de mensajes. Los datos son distribuidos equitativamente en los discos conectados a los procesadores.

Los tres algoritmos ya mencionados anteriormente son:

- *Count Distribution* (CD): Se enfoca en la minimización de la comunicación. Este algoritmo usa el principio simple de permitir "cálculos redundantes en paralelo en procesadores *idle* para de esta manera disminuir la comunicación".
- *Data Distribution* (DD): Intenta usar de forma más efectiva la memoria principal compartida del sistema. En este algoritmo, cada procesador cuenta candidatos mutuamente exclusivos. De esa forma, mientras aumenta el número de procesadores se pueden contar más candidatos en una iteración. La desventaja de este algoritmo es que cada procesador debe enviar sus datos locales a los demás procesadores en cada iteración.
- *Candidate Distribution* (CDD): Este algoritmo explota las características propias del problema a resolver ya sea para reducir la sincronización entre los procesadores, como para segmentar la base de datos basándose en los patrones que soportan las distintas transacciones. Este algoritmo incorpora además balanceo de carga. El CDD particiona los datos y los candidatos de tal forma que cada procesador pueda proceder independientemente.

4.1.1.1 Algoritmo Count Distribution

Este algoritmo utiliza el principio simple de permitir "cálculos redundantes en paralelo para disminuir la comunicación". La primera iteración es especial. Para todas las demás iteraciones k>1, el algoritmo trabaja como sigue:

1. Cada procesador P^i genera todo el C_k, usando todos los *itemsets* frecuentes L_{k-1} creados al final de la iteración k-1. Observe que como todos los procesadores tienen el mismo L_{k-1}, ellos generarán C_k idénticos.
2. El procesador P^i hace una iteración sobre su partición D^i y cuenta el soporte local para los candidatos en C_k.
3. El procesador P^i intercambia los conteos locales de C_k con todos los demás procesadores para crear los conteos globales de C_k. Los procesadores están forzados a sincronizar este paso.
4. Cada procesador P^i calcula L_k a partir de C_k.
5. Cada procesador P^i decide independientemente si terminar o pasar a la siguiente iteración. Esta decisión será idéntica en todos los procesadores ya que estos tienen idéntico L_k.

En la primera iteración, cada procesador P^i genera dinámicamente sus candidatos locales C_1^i, en dependencia de los ítems que se encuentren presentes en la partición local D^i. Por tanto, los candidatos contados por diferentes procesadores pueden no ser idénticos y debe tenerse mucho cuidado en el intercambio de los conteos locales para determinar el C_1 global.

4.1.1.2 Algoritmo Data Distribution

Este algoritmo se describe de la siguiente forma:

Iteración 1: El mismo procedimiento que en CD.

Iteración $k > 1$:

1. El procesador P^i genera C_k a partir de L_{k-1}. Asumiendo que la cantidad de procesadores es N, cada procesador se queda sólo con la n-ésima parte de los *itemsets* formando el subconjunto candidato C_k^i para el que realizará el conteo de soporte de cada uno de sus integrantes. Con el id del procesador se determina cuáles *itemesets* conformarán C_k^i y esto puede ser llevado a cabo sin comunicación alguna entre los procesadores, en este caso los *itemsets* son asignados usando un *round-robin*. Los conjuntos C_k^i son disjuntos y su unión es el C_k original.
2. El procesador P^i desarrolla el conteo de soporte para los *itemsets* en su conjunto de candidatos local C_k^i usando datos locales y datos enviados por otros procesadores.
3. Al final de la iteración, cada procesador P^i calcula L_k^i usando el C_k^i local. Nuevamente, los L_k^i son disjuntos y su unión es L_k.
4. Los procesadores intercambian L_k^i para que cada procesador tenga el L_k completo para la generación del C_{k+1} de la siguiente iteración. Este paso requiere que los procesadores se sincronicen. Habiendo obtenido el L_k cada procesador puede decidir independiente e idénticamente si terminar o continuar a la próxima iteración.

4.1.1.3 Algoritmo Candidate Distribution

El algoritmo de distribución de candidatos particiona los datos y los candidatos de tal forma que cada procesador puede proceder independientemente. En algún paso l, donde l es determinado heurísticamente, este algoritmo divide los itemsets frecuentes del conjunto $L_{l\text{-}1}$ entre los procesadores de tal forma que un procesador P^i puede generar un único C_m^i ($m \geq l$) independiente de los otros procesadores ($C_m^i \cap C_m^j = \phi$, $i \neq j$).

Al mismo tiempo, los datos son particionados tal que cada procesador puede contar los candidatos en C_m^i independiente de los demás procesadores. A continuación se describe en detalles este algoritmo.

Iteración $k < l$: use CD o DD.

Iteración $k = l$:

1. Se particiona $L_{k\text{-}1}$ entre los N procesadores de tal forma que los conjuntos $L_{k\text{-}1}$ estén bien balanceados. Más abajo discutimos como se hace este particionamiento. Se guarda con cada itemset frecuente en $L_{k\text{-}1}$ a que procesador se le asignó. Este particionamiento se hace idénticamente en paralelo para cada procesador.
2. El procesador P^i genera C_k^i, lógicamente usando sólo la partición de $L_{k\text{-}1}$ asignada a él. Note que P^i aún tiene acceso a todo $L_{k\text{-}1}$, y que, por tanto, puede usarse una poda estándar mientras se genera C_k^i en esta iteración.
3. P^i realiza el conteo global para los candidatos en C_k^i y al mismo tiempo, la base de datos es particionada en DR^i, que es el conjunto de datos local al procesador P^i luego del particionamiento de $L_{k\text{-}1}$.
4. Después que P^i ha procesado todos sus datos locales y no recibe más datos de los restantes procesadores, prepara N-1 *buffers* de recepción asíncronos para recibir los L_k^j de los demás procesadores. Estos L_k^j son necesarios para podar C_{k+1}^i en el paso de poda de la generación de candidatos.
5. El procesador P^i calcula L_k^j a partir de C_k^i y asíncronamente lo trasmite a los otros N-1 procesadores usando N-1 envíos asíncronos.

Iteración $k > l$:

1. El procesador P^i recibe todos los *itemsets* frecuentes que le han enviado otros procesadores. Estos *itemsets* son usados en el paso de poda de la generación de candidatos, pero no en el paso de unión. Los *itemsets* recibidos por el procesador j pudieran ser de longitud k-1, menores que k-1 (procesador más lento), o mayores que k-1 (procesador más rápido). El procesador P^i mantiene un seguimiento de los *itemsets* frecuentes de mayor longitud enviados por cada procesador P^j. Los *buffers* de recepción de los *itemsets* frecuentes son vueltos a publicar después del procesamiento.
2. P^i genera C_k^i usando el L_{k-1}^i local. Ahora puede ocurrir que P^i no haya recibido L_{k-1}^j de los demás procesadores, por tanto P^i necesita ser cuidadoso a la hora de podar. Necesita distinguir un *itemset* (un subconjunto de tamaño k-1 de un *itemset* candidato) que no esté presente in ningún L_{k-1}^j a partir de un *itemset* que esté presente en algún L_{k-1}^i pero que este conjunto aún no ha sido recibido por el procesador P^i. Esto lo hace probando $L_{l\text{-}1}$ usando prefijos de longitud l-1 del *itemset* en cuestión (recuerde que el reparticionamiento se

efectúa en la iteración *l*), encontrando al procesador responsable de este. Y chequeando si L_{k-1}^{j} ha sido recibido desde este procesador.

3. P^i hace una iteración sobre DR^i y cuenta C_k^i. Entonces calcula L_k^i a partir de C_k^i y transmite de forma asíncrona L_k^i a los demás procesadores usando *N*-1 envíos asíncronos.

4.1.2 Otros algoritmos paralelos reportados en la literatura

Además de los antes mencionados, se pueden citar los siguientes algoritmos paralelos:

- PEAR & PPAR [Mueller, 1995].
- PAR MINER(X) [Shen, 1999].
- TreeProjection [Agarwal, 2000].
- MLFPT (Multiple Local Frequent Pattern Tree) [Zaiane, 2001].
- PAR-DCI (Parallel Direct Count & Intersect) [Orlando, 2002].
- DATA-VP [Coenen, 2003].
- Distributed Decision Miner (DDM) [Schuster, 2004].

4.2 Algoritmo paralelo ParCBMine

En este epígrafe describiremos la versión paralela del algoritmo CBMine y al cual hemos denominado ParCBMine (Parallel CBMine).

ParCBMine aprovecha las ventajas de la representación vertical de la base de datos, descritas en CBMine y combina adecuadamente los modelos de programación paralela de memoria compartida y distribuida a través del uso de las bibliotecas *Pthreads* (para la programación multihilos) y MPI (para la programación basada en pase de mensajes) respectivamente.

La combinación de la programación multihilos con la de pase de mensajes en ParCBMine, se hizo sobre la base de que el algoritmo fue implementado sobre un *cluster* de nodos SMP (*Symmetric Multi-Processing*) para el procesamiento en paralelo administrado con sistema operativo Linux, compuesto por nodos *dual procesor*. Los procesadores son del tipo Xeon dotados con la tecnología "*hyperthreading"*, lo que posibilita en cada nodo puedan ejecutarse de forma concurrente hasta 4 hilos.

Si bien el algoritmo no está ceñido a la cantidad de hilos reales (dígase procesadores) que podrían desplegarse en cada nodo, esto es un elemento importante en la escalabilidad del mismo, pues permite emplear de mejor manera la información global que se encuentra en la memoria compartida de cada nodo, como por ejemplo en la generación y el conteo del soporte de los candidatos. A esto es lo que muchos autores denominan aprovechar el "paralelismo intranodos" y de cierta forma hemos incorporado también algunos aspectos del algoritmo *Candidate Distribution*, empleando en este caso la programación multihilo a traves del uso de la biblioteca *Pthreads*.

Desde el punto de vista de la estrategia de particionamiento de los datos, ParCBMine al igual que el *Count Distribution*, adopta un particionamiento horizontal de la base de datos aplicando de esta forma un "paralelismo internodos", es decir si el *cluster* posee *n* nodos, la base de datos se divide equitativamente entre los *n* nodos y en este caso la comunicación entre los nodos se realiza mediante pase de mensajes empleando la biblioteca MPI.

Como nuestro algoritmo está orientado a una arquitectura paralela basada en un *cluster* de SMP, es importante señalar que de haber empleado sólo el modelo de memoria distribuida

basada en el pase de mensajes, como ha sido señalado por otros autores, se hubiera afectado considerablemente el rendimiento del algoritmo.

Es conocido que cualquier algoritmo basado en el *Count Distribution*, en la medida que el número de particiones de la base de datos aumenta, se degrada la escalabilidad del algoritmo ya que en el proceso de sincronización de los procesos para realizar el conteo global del soporte de los itemsets que conforman el conjunto candidato, la cantidad de información que recibe cada proceso MPI en cada iteración es igual a (N-1) x $|C_k|$, donde: N es la cantidad de procesos MPI a los que les fue asignado cada partición de la base de datos y $|C_k|$ es el cardinal del conjunto candidato generado en cada iteración y para el que cada proceso MPI determinó de forma local el conteo del soporte. Téngase en cuenta que para cualquiera que sea N, el cardinal de C_k no varía, por lo que un aspecto que ha sido y continúa siendo objeto de investigación es reducir el conjunto C_k local aplicando estimaciones probabilísticas del soporte local, véase [Cheung, 1998] y [Schuster, 2004].

A los elementos antes mencionados se agrega también el hecho de que por tratarse de nodos SMP no se aprovecharía de forma eficiente la localidad de los datos ya que todos los procesos MPI que se corran en cada nodo tratarán de distribuirse de forma equitativa la memoria física total del nodo, reservando la misma cantidad de memoria para estructuras de datos que son redundantes y a su vez grandes consumidoras de memoria como L_{k-1} y C_k, así como para la partición de la base de datos que le fue asignada.

De esta forma y considerando los elementos antes mencionados, se describe a continuación el algoritmo ParCBMine.

4.2.1 Descripción del algoritmo ParCBMine

Considerando una arquitectura máster-esclavos, típica de los *clusters* paralelos, en la primera iteración, el nodo master o coordinador determina el conjunto L_1 global y particiona la base de datos D entre los N nodos que conforman el *cluster*.

La primera iteración es diferente a las demás. Para todas las demás iteraciones k>1, el algoritmo trabaja como sigue:

1. Cada proceso *master-thread* P^j (j=1, N) genera todo el conjunto C_k, usando todos los *itemsets* frecuentes L_{k-1} creados al final de la iteración k-1. Observe que como todos los procesos tienen el mismo L_{k-1}, ellos generarán C_k idénticos. En particular los procesos P^i (i=1, N x *MaxThreads*) que corren en un mismo nodo comparten las estucturas de memoria reservadas para L_{k-1}, C_k y D^j (j=1, N).
2. El proceso *master-thread* P^j abre (*MaxThreads*-1) nuevos hilos y cada uno de estos hilos de procesos hace una iteración sobre la partición D^j correspondiente a su nodo y cuenta el soporte local de una parte de los candidatos en C_k, ya que cada hilo contará el soporte de los elementos candidatos de C_k cuyo número de orden es congruente con h = ((número de orden del candidato) módulo *MaxThreads*).
3. De esta forma el conjunto C_k local del nodo j, se particiona de forma aproximadamente equitativa (téngase en cuenta que $|C_k|$ no necesariamente tiene que ser múltiplo de *MaxThreads*) y cada hilo de proceso realiza el conteo de soporte de sus candidatos sin necesidad de realizar sincronización en el acceso a memoria, ya que aprovechando la representación vertical de la base de datos, el conteo del soporte se realiza sobre una estructura de memoria reservada para cada candidato.

4. El proceso *master-thread* P^j le envía al nodo master o coordinador los conteos locales de C_k, para que este realice una operación all_reduce y obtenga los conteos globales de C_k. Los procesos *master-thread* P^j están forzados a sincronizar en este paso.
5. El nodo máster o coordinador calcula L_k a partir de C_k. Si L_k no es vacío el coordinador se lo envía a los procesos master-thread P^j y se pasa a la siguiente iteración.

Nótese que a diferencia del *Count Distribution*, hemos sustituido la palabra procesador por proceso, ya que dadas las características del *hardware* de nuestro *cluster,* la cantidad de procesos es mayor que la cantidad de procesadores y puede ser expresada de forma general por *N* x *MaxThreads*, donde: *N* es el número de nodos y *MaxThreads* es la cantidad máxima de hilos por nodo, en nuestro caso particular *MaxThreads* es 4 considerando el uso de la tecnología *Hyperthreading*.

A diferencia del algoritmo PAR-DCI [Orlando 2002], en el que se vuelve a particionar la base de datos local de cada nodo en tantas porciones como hilos fueran posibles desplegar, en el paso 2 de ParCBMine se adoptó una variante más eficiente consistente en distribuir en forma de *round-robin* el conteo del soporte de los candidatos entre los diferentes hilos.

4.2.2 Consideraciones finales del algoritmo

Sin dudas, los tres algoritmos paralelos propuestos por Rakesh Agrawal y John Shafer en [Agrawal, 1996], continúan siendo un punto de referencia obligada en el desarrollo de algoritmos paralelos para el minado de Reglas de Asociación.

Haciendo una valoración general de dichos algoritmos podemos decir que el CD reduce el costo en la comunicación a expensas de ignorar la memoria física del sistema. En un ambiente de *cluster* de *workstations* donde se utilicen nodos monoprocesadores probablemente sea este el enfoque ideal, sin embargo puede no serlo en el caso donde los nodos sean del tipo SMP, ya que se desaprovecharía la posibilidad de combinar los modelos de memoria compartida y distribuida. Para alcanzar implementaciones eficientes basadas en el CD, continúa siendo un problema latente el determinar nuevas heurísticas que permitan reducir el cardinal del C_k local obtenido por cada procesador.

El DD puede ayudarnos a explorar este aspecto ya que explota íntegramente la memoria física a riesgo de tener un gran peso en las comunicaciones. La habilidad de este algoritmo de contar en una misma iteración *n* veces tantos candidatos como CD, hace de este un fuerte contendiente.

Con el tercer algoritmo denominado CDD, si se incorpora un conocimiento detallado del problema pueden procurarse los beneficios de CD y DD [Agrawal, 1996], sin embargo, encontrar una heurística que permita a partir de un paso $k=l$, realizar un particionamiento de los *itemsets* candidatos de forma tal que no sea necesario a partir de ese paso, efectuar la sincronización entre los procesadores y obtener un balanceo adecuado de la carga entre los procesadores, sigue constituyendo un reto para los investigadores que trabajan en la paralelización de algoritmos para el minado de Reglas de Asociación.

Por último, debemos resaltar la inclusión en este trabajo de nuestra propuesta de un nuevo algoritmo paralelo basado en el algoritmo secuencial CBMine, y sustentado en los principios del CD, en el que se introducen también algunos aspectos del CDD, sobre la base de combinar adecuadamente la programación paralela basada en el modelo de pase de mensajes con la programación multihilo. Este nuevo algoritmo continúa en fase de desarrollo y esperamos en próximas versiones presentar nuevos resultados orientados a reducir el nivel de sincronización entre los procesadores y alcanzar un mejor balanceo de la carga entre ellos.

5. Características de las reglas de asociación

En epígrafes anteriores se analizaron diferentes métodos para la obtención de los conjuntos de ítems (*itemsets*) frecuentes. Sin embargo, como fue señalado en el epígrafe 2, la generación de itemsets frecuentes es sólo el primer paso para la obtención de las reglas de asociación.

Como es conocido, en la generación de los *itemsets* se utiliza un valor de umbral mínimo (*minsup*) para considerarlos como frecuentes. De forma similar, en la generación de las RA se utilizan medidas de calidad para considerar como relevantes a las reglas obtenidas, siendo la medida más utilizada la Confianza, existiendo un valor de umbral mínimo (*minconf*) para decidir que una regla es confiable. Estos valores de umbral no sólo permiten obtener *itemsets* y reglas interesantes, sino además posibilitan la reducción de los conjuntos generados debido a las grandes dimensiones que pueden alcanzar estos.

Con el fin de obtener un conjunto de RA razonablemente compacto, y que a la vez contenga el máximo de información relevante sobre los ítems y relaciones consideradas, se analizarán a continuación varios tipos de reglas de asociación; además, se expondrán otras medidas de calidad empleadas en la generación de las reglas.

5.1 Tipos de reglas de asociación

En la literatura consultada se han encontrado varios tipos de RA, los cuales permiten definir subconjuntos de reglas, optimizando la ejecución de los algoritmos, reduciendo los volúmenes de reglas a almacenar, y posibilitando al mismo tiempo la recuperación del conjunto original de reglas relevantes o confiables. Estos tipos de RA son las siguientes:

- Reglas de asociación representativas (RAR).
- Reglas de asociación de mínima condición y máxima consecuencia (RAMM).
- RA generalizadas (RAG).

A continuación se expondrán en detalles cada una de estas reglas.

5.1.1 Reglas de asociación representativas

La cantidad de reglas de asociación que pueden ser generadas en un problema real, sobre todo cuando se trabaja con documentos y particularmente con noticias, puede ser extremadamente grande; para aliviar este problema Marzena Kryszkiewicz introdujo el concepto de regla de asociación representativa [Kryszkiewicz, 1998a]. Un conjunto de reglas de asociación representativas tiene la característica de ser el menor conjunto a partir del cual pueden generarse todas las reglas que satisfacen los umbrales *minsup* y *minconf*, mediante un operador de cubrimiento.

Definición. *Cubrimiento de una regla de asociación* – El cubrimiento (C) de una regla $X \Rightarrow Y$, es un conjunto de reglas que verifican la siguiente expresión:

$$C(X \Rightarrow Y) = \{X \cup Z \Rightarrow V \mid Z,V \subseteq Y, Z \cap V = \varnothing, V \neq \varnothing\}. \quad (10)$$

Como puede observarse, las reglas que generan el operador de cubrimiento son aquellas en las que se mueven al antecedente parte de los ítems del consecuente, manteniendo como consecuente parte de (o todos) los ítems no movidos.

Considerando el operador de cubrimiento puede enunciarse el concepto de regla de asociación representativa.

Definición. *Regla de asociación representativa* – El conjunto de las reglas de asociación representativas que satisfacen los valores de umbral *minsup* y *minconf*, indicándose por simplicidad como RAR, es aquel que verifica la siguiente expresión:

$$RAR = \left\{ r \in RA \mid \neg \exists r' \in RA, r' \neq r, r \in C(r') \right\}. \quad (\mathbf{11})$$

En esa expresión se ha indicado con las siglas RA a las reglas de asociación que satisfacen los valores de umbral del soporte y la confianza.

Nótese que las RAR son las reglas que no pertenecen al cubrimiento de ninguna otra. Además, las RAR están formadas por un conjunto conciso y sin pérdida de información, ya que todas las RA pueden obtenerse a partir de la unión de los cubrimientos de todas las RAR.

5.1.2 Reglas de asociación de mínima condición y máxima consecuencia

Formalmente, las reglas de asociación de mínima condición y máxima consecuencia pueden definirse como se indica a continuación.

Definición. *Regla de asociación de mínima condición y máxima consecuencia* – El conjunto de las reglas de asociación de mínima condición y máxima consecuencia que satisfacen los valores de umbral *minsup* y *minconf*, indicándose por simplicidad como RAMM, es aquel que verifica la siguiente expresión:

$$RAMM = \left\{ r : X \Rightarrow Y \in RAR \mid \neg \exists r' : X' \Rightarrow Y' \in RAR, r \neq' r, X' \subseteq X, Y \subseteq Y' \right\} \quad (\mathbf{12})$$

En [Kryszkiewicz, 1998b] se demuestra que RAMM $\subseteq$ RAR, por lo que estas reglas conforman un conjunto de menor cardinalidad. Además, este tipo de regla facilita los análisis de contextos o antecedentes con un mínimo de ítems. No obstante, debe resaltarse que este tipo de regla no permite obtener todas las RA, lo cual representa una restricción significativa.

5.1.3 Reglas de asociación generalizadas

Shrikant y Agrawal introdujeron en 1995 el problema del minado de reglas de asociación generalizadas [Srikant, 1995]. Una regla de asociación generalizada (RAG) es un tipo de regla que se obtiene cuando los ítems se expresan a través de una taxonomía de conceptos. Los soportes en estos tipos de reglas se determinan considerando que una transacción *T* "soporta" un ítem de una RAG si ese ítem está en *T* o es un ancestro de algún ítem de *T*. Se dice que una transacción *T* "soporta" el *itemset X* si *T* "soporta" cada ítem de *X*. A continuación se presenta una definición de este tipo de regla.

Definición. *Regla de Asociación Generalizada* – Una regla de asociación generalizada es una regla $X \Rightarrow Y$ válida cuyos ítems representan conceptos pertenecientes a jerarquías o taxonomías, verificándose que ningún ítem en Y es un concepto ancestro de algún ítem en X.

La razón de exigir la condición de "no ancestro" es debido a que una regla $X \Rightarrow$ ancestro(X) es trivial, con una confianza del 100% y por lo tanto, redundante.

Por ejemplo, dada la taxonomía que plantea que un pantalón es una ropa de vestir y que toda ropa de vestir es una ropa, entonces pudiera inferirse la regla "las personas que compran ropa de vestir tienden a comprar zapatos" si el soporte y confianza asociados con todos los tipos de ropa de vestir, junto al ítem zapato, satisfacen los umbrales correspondientes, aún cuando las reglas "las personas que compran pantalones tienden a comprar zapatos" y "las personas que compran ropa tienden a comprar zapatos" no se hayan obtenido directamente por no satisfacer los umbrales mínimos definidos.

Como es sabido, la mayoría de los trabajos sobre reglas de asociación no consideran la presencia de taxonomías y restringen las reglas de asociación a los ítems hojas de las taxonomías [Agrawal, 1993], [Agrawal, 1994], [Savasere, 1995], [Zaki, 1997a], [Han, 2000]. Sin embargo, la consideración de este tipo de regla pudiera resultar de gran interés en determinadas aplicaciones en las que los ítems hojas de las taxonomías no alcanzan soportes altos. Este tipo de reglas de asociación, además, pueden facilitar la poda de reglas redundantes o poco interesantes.

Las RAG pueden ser también de gran utilidad cuando se aplican a conjuntos de documentos por el carácter intrínsicamente taxonómico de los términos del lenguaje natural. Ideas similares fueron aplicadas por Feldman y Hirsh en el sistema FACT [Feldman 1996]. Con igual basamento, Montes y Gómez en su tesis doctoral propuso el descubrimiento de asociaciones en jerarquías de grafos conceptuales [Montes, 2002].

5.2 Medidas de interés de las reglas de asociación

Uno de los problemas principales en el descubrimiento de reglas de asociación es el desarrollo de una medida que permita evaluar la significación de las reglas descubiertas. Una regla que es interesante para un usuario pudiera no serlo para otro, por lo cual resulta muy difícil definir una medida de interés ideal. No obstante, una medida de interés objetiva que se base en métodos estadísticos o lógicos ayuda a eliminar las reglas innecesarias y reducir el espacio de búsqueda. A continuación se hará un análisis de distintas medidas de interés propuestas para el cálculo de las reglas de asociación que han sido utilizadas en diferentes aplicaciones.

5.2.1 Soporte y confianza

Las medidas más utilizadas para evaluar las reglas de asociación son el soporte y la confianza, considerándose interesantes las que superen valores de umbral asociados [Agrawal, 1993]. Como es conocido, el soporte expresa la probabilidad *a priori* de la ocurrencia de un *itemset* o de una regla de asociación (ver las definiciones respectivas en el epígrafe 2). Mientras tanto, la confianza expresa la probabilidad condicional de que una transacción contenga al *itemset Y* dado que contiene al *itemset X* (ver definición de confianza en epígrafe 2). Sin embargo, las reglas que cumplan con los valores de umbral mínimo de soporte y confianza no siempre resultan

interesantes. Por otra parte, la definición de estos valores de umbral mínimo no siempre resulta una tarea fácil para los usuarios. Estas limitaciones serán argumentadas a continuación.

5.2.1.1 Limitaciones de la confianza

La principal limitación de la confianza es que no expresa con precisión todos los tipos de dependencia estadística entre consecuente y antecedente.

A modo de ejemplo, supóngase que en una base de datos se verifican los siguientes soportes: Sup(X) = 0.5, Sup(Y) = 0.7 y Sup($X \Rightarrow Y$) = 0.3. Según (3), Conf($X \Rightarrow Y$) = 0.6, lo que decidiría que esa regla fuera elegible si *minconf* = 0.5. Sin embargo, considerando que la probabilidad *a priori* de Y, según su soporte, es superior que si se condiciona por X, entonces la ocurrencia de Y en el contexto de X no es más probable (y por ende no necesariamente presenta una significación mayor) que la que Y posee en toda la base de datos. Observe que la significación o no de la disminución de la probabilidad condicional en relación con la probabilidad a priori del consecuente es correlativa, indicándose esto como una dependencia negativa. La dependencia negativa (o sea, la relación de la presencia de X con la ausencia de Y) pudiera ser de interés en determinados problemas.

Si en el ejemplo anterior se tuviera que Sup($X \Rightarrow Y$) = 0.35, entonces se verificaría la propiedad de independencia probabilística entre X e Y al comprobarse que Sup($X \cup Y$) = Sup(X) * Sup(Y). Sin embargo, la confianza sería aún mayor, eligiéndose la regla como interesante a pesar de no evidenciarse tal dependencia entre consecuente y antecedente.

Por lo tanto, la confianza tiene como limitante que no es capaz de detectar la independencia estadística ni la dependencia negativa entre consecuente y antecedente, ya que no considera el soporte del consecuente.

5.2.1.2 Limitaciones del soporte

El soporte expresa la fuerza o representatividad de un elemento en la base de datos. Un principio común es asumir que mientras mayor sea el soporte de un *itemset* mejor (más representativo, más interesante) será este. Sin embargo, esto no siempre es válido.

Para ejemplificar lo anterior, supóngase que Sup(X) = 0.3, Sup(Y) = 0.95 y Sup($X \Rightarrow Y$) = 0.3. Según (3), Conf($X \Rightarrow Y$) = 1.0, lo que decidiría que esa regla fuera elegible con cualquier umbral de confianza. Puede notarse que el *itemset* Y está presente en gran parte de las transacciones, por lo que no es sorprendente que se obtengan reglas con altas confianzas para una gran cantidad de *itemsets* antecedentes; sin embargo, esto no significa que tales reglas reflejen una real dependencia.

En general, cuando el soporte del consecuente es alto, no es posible afirmar con certeza que exista una fuerte dependencia, aún cuando la confianza indique un alto interés. Cuando el Sup(Y) es alto y Conf($X \Rightarrow Y$) $\leq$ Sup(Y), las relaciones entre consecuente y antecedente pueden ser de independencia o dependencia negativa; pero cuando Conf($X \Rightarrow Y$) > Sup(Y), la dependencia positiva pudiera resultar dudosa. Una posible solución a esta limitante del soporte es el concepto de regla de asociación muy fuerte (ver epígrafe 6.2.4).

5.2.1.3 Limitaciones de los umbrales mínimos

Una de las limitaciones que tiene la decisión de reglas interesantes por sus soportes y confianzas es la necesidad de definir valores de umbral mínimo. Como ha sido señalado reiteradamente, un valor de *minsup* alto evita la explosión combinatorial en el descubrimiento de *itemsets*

frecuentes; esto se logra a expensas del no descubrimiento de patrones interesantes con bajos soportes.

En el epígrafe anterior se expuso algunas situaciones en las que soportes altos no siempre determinan reglas interesantes. Adicionalmente, es conocido que la mayoría de las reglas con altos soportes son obvias y conocidas, mientras que las reglas con bajos soportes son las que ofrecen nuevas percepciones interesantes, tales como desviaciones o excepciones.

Otra limitación en la definición de un valor de umbral de soporte adecuado es su carácter no uniforme en muchas de las aplicaciones. Es conocido que los soportes naturales de los ítems no son uniformes; por ejemplo, las ventas de productos de altos dividendos, como los televisores y otros equipos electrodomésticos, ocurren menos frecuentemente que otros de bajo costo (artículos de limpieza de uso cotidiano, etc.).

Una alternativa a la definición de un único valor de umbral para el soporte es la propuesta de Liu *et al.* [Liu, 1999]. En este método los usuarios pueden especificar soportes mínimos diferentes para los ítems de la base de datos. En este, el soporte mínimo de un *itemset* se toma como el menor de los soportes mínimos de los ítems presentes en el *itemset*. A continuación se presenta una formalización de este método.

Definición. Sea $I = \{i_1, i_2, \ldots, i_n\}$ el conjunto de ítems de una base de datos, $ms(i_k)$ el soporte mínimo permitido para el ítem i_k, y X un *itemset*, $X \subset I$. El *itemset* X es frecuente o extenso si se verifica la siguiente expresión.

$$Sup(X) > \min_{i_k \in X} ms(i_k) \qquad (\mathbf{13})$$

Basándose en este método, Lin *et al.,* propusieron eximir al usuario de la definición de uno o múltiples soportes mínimos, estimando los soportes mínimos por diferentes criterios [Lin, 2002]. Según este trabajo, los soportes mínimos de cada ítem pueden calcularse a partir de sus soportes en la base de datos. Una de las propuestas realizadas es la siguiente expresión:

$$ms(i_k) = Sup(i_k) * minconf\ . \qquad (\mathbf{14})$$

En esta propuesta, la expresión dada en (13) permite decidir si un *itemset* es frecuente. En este método propuesto por Lin *et al.* no sólo evita la definición de un soporte mínimo, sino que además garantiza con el cálculo de los *itemsets* frecuentes la satisfacción de las confianzas mínimas en algunos casos. Esto se puede comprobar con el siguiente lema.

Lema. Sea $X \cup Y$ un *itemset* frecuente, de acuerdo a las expresiones (13) y (14), y sea i_k el ítem de menor soporte de ese *itemset* según (13). Si $i_k \in X$, entonces la confianza de la regla de asociación $X \Rightarrow Y$ satisface el umbral *minconf*.

Este lema puede demostrarse al comprobarse que $\text{Sup}(X \cup Y) \geq \text{Sup}(i_k) * minconf$ y que $\text{Sup}(X) \leq \text{Sup}(i_k)$, lo que sigue que $\text{Sup}(X \cup Y) / \text{Sup}(X) \geq minconf$.

En el trabajo referenciado se han propuesto otras expresiones para el cálculo de $ms(i_k)$, ofreciendo diferentes posibilidades.

5.2.2 Convicción

La convicción introducida por Brin *et al.,* permite medir la fuerza de la implicación de una regla, en oposición a la concurrencia de los ítems [Brin 1997]. A continuación se presenta su definición.

Definición. *Convicción de una regla de asociación* – La convicción (*conviction*) de una regla de asociación $X \Rightarrow Y$ está dada por la siguiente expresión.

$$Conv(X \Rightarrow Y) = \frac{Sup(X) * Sup(\neg Y)}{Sup(X \cup \neg Y)} = \frac{1 - Sup(Y)}{1 - Conf(X \Rightarrow Y)} \quad (\mathbf{15})$$

En esa expresión, el término $\neg Y$ representa la ausencia del *itemset* Y en la transacción. Esta medida se expresa con valores positivos sin que exista un límite superior; los valores inferiores a 1 indican dependencias negativas entre el consecuente y el antecedente, los valores superiores a 1 indican dependencias positivas, mientras que 1 indican una total independencia. Como puede notarse, la variabilidad de esta medida es alta, lo cual limita su aplicabilidad.

A modo de ejemplo, supóngase que Sup(X) = 0.6, Sup(Y) = 0.9 y Sup($X \cup Y$) = 0.54. No es difícil comprobar que Conv($X \Rightarrow Y$) = 1, verificándose que Y es independiente de X. Mientras tanto, con un simple incremento de Sup($X \cup Y$) la convicción se incrementaría rápidamente, tendiendo a infinito al aproximarse a 0.6.

5.2.3 Ascenso

El ascenso (*lift*) es una medida propuesta por Berry y Linoff en 1997 [Berry, 1997], y también por Silvertein bajo el nombre de interés (*interest*) [Silvertein, 1998]. Esta medida permite determinar el grado de dependencia entre los itemsets, siendo su definición la siguiente.

Definición. *Ascenso de una regla de asociación* – El ascenso (*lift*) de una regla de asociación $X \Rightarrow Y$ está dado por la siguiente expresión.

$$Lift(X \Rightarrow Y) = \frac{Sup(X \cup Y)}{Sup(X) * Sup(Y)} \quad (\mathbf{16})$$

Esta medida se comporta de forma similar a la convicción; o sea, se expresa con valores positivos hasta el infinito, indicando el valor 1 la independencia, los menores a 1 la dependencia negativa y los mayores la dependencia positiva entre el consecuente y el antecedente. Sin embargo, esta medida presenta una desventaja adicional, su carácter simétrico; o sea, Lift($X \Rightarrow Y$) = Lift($Y \Rightarrow X$), lo cual no es por lo general un comportamiento deseable.

No obstante las limitaciones anteriores, y por la simplicidad de su formulación y la posibilidad de discriminar la dependencia positiva de la negativa, Wen-Yang Lin *et al.,* propusieron la combinación del ascenso positivo con los soportes y confianzas para decidir si una regla es interesante o no [Lin, 2002].

5.2.4 Factor de certeza

El factor de certeza es una medida propuesta por Shortliffe y Buchanan para el tratamiento de la incertidumbre de las reglas de producción en el sistema experto MYCIN, convirtiéndose en un paradigma en estos tipos de sistemas [Shortliffe, 1975]. En el marco de las reglas de asociación fue propuesto por Berzal *et al.,* en un trabajo del 2001[Berzal, 2001]. A continuación se presenta su definición.

Definición. Factor de certeza. El factor de certeza (*certainty factor*) de una regla de asociación $X \Rightarrow Y$ está dada por la siguiente expresión.

$$CF(X \Rightarrow Y) = \begin{cases} \dfrac{Conf(X \Rightarrow Y) - Sup(Y)}{1 - Sup(Y)}, & \text{si} \quad Conf(X \Rightarrow Y) > Sup(Y) \\ \dfrac{Conf(X \Rightarrow Y) - Sup(X)}{Sup(Y)}, & \text{si } Conf(X \Rightarrow Y) < Sup(Y) \\ 0 \qquad\qquad\qquad\qquad , & \text{en otro caso} \end{cases} \tag{17}$$

Esta medida expresa (en forma de índice) el por ciento del incremento (o decremento) de la probabilidad condicional de Y respecto a X relativo a la magnitud del intervalo definido por la probabilidad *a priori* de Y. Esta medida se expresa en el intervalo [-1, 1], indicando los valores positivos el incremento relativo de la probabilidad condicional, los negativos el decremento, y el valor 0 la independencia entre el consecuente y el antecedente.

Berzal *et al.,* muestran la superioridad del factor de certeza respecto a la confianza, superando las limitaciones anotadas para esa medida. Además, indican que el factor de certeza tampoco es una buena medida para evaluar las dependencias negativas. En particular, muestran que el factor de certeza no permite discriminar la dirección de la dependencia negativa, al verificarse la siguiente expresión:

$$\text{Si} \quad CF(X \Rightarrow Y) < 0, \quad \text{Entonces} \quad CF(X \Rightarrow Y) = CF(Y \Rightarrow X). \tag{18}$$

Por lo anterior, esta medida se ha propuesto para determinar sólo las dependencias positivas. En este caso se seleccionan las reglas si sus soportes superan *minsup* y sus factores de certeza superan un valor de umbral mínimo *minCF*. Conociendo, de los trabajos sobre el factor de certeza en los sistemas expertos, que CF($X \Rightarrow Y$) ≤ Conf($X \Rightarrow Y$), entonces el valor de umbral *minCF* a utilizar pudiera ser algo menor, o muy similar, que el de la confianza.

Con el propósito de resolver las limitaciones que tiene el soporte (ver epígrafe 6.2.1.2), Berzal *et al.,* propusieron un tipo de regla de asociación denominada "muy fuerte".

Definición. *Regla de asociación muy fuerte* – Una regla de asociación $X \Rightarrow Y$ es muy fuerte (*very strong*) si ambas reglas $X \Rightarrow Y$ y $\neg Y \Rightarrow \neg X$ son fuertes.

Una regla de asociación es fuerte si su soporte y factor de certeza superan los umbrales correspondientes. La razón de este tipo de regla es que si Sup(Y) (o Sup(X)) es muy alto, entonces Sup($\neg Y \Rightarrow \neg X$) es muy bajo y esa regla no será fuerte.

Puede verificarse que las reglas muy fuertes satisfacen las siguientes expresiones.

$$\text{Si } Sup(X)+Sup(Y)>1, \text{Entonces } X \Rightarrow Y \text{ es muy fuerte si y sólo si } X \Rightarrow Y \text{ es fuerte} \quad (19)$$
$$\text{Si } Sup(X)+Sup(Y)<1, \text{Entonces } X \Rightarrow Y \text{ es muy fuerte si y sólo si } \neg Y \Rightarrow \neg X \text{ es fuerte}$$
$$\text{Si } Sup(X)+Sup(Y)=1, \text{Entonces } X \Rightarrow Y \text{ es fuerte si y sólo si } \neg Y \Rightarrow \neg X \text{ es fuerte}$$

Estas expresiones permiten simplificar los procedimientos de cálculos. No obstante, la aplicación de este tipo de regla exige la determinación de los soportes de itemsets negativos.

5.2.5 Interés de Dong y Li

El interés (*interestingness*) propuesto por Dong y Li es una medida de la importancia de la regla de asociación considerando su diferencia respecto a otras reglas presentes en su vecindad [Dong, 1998]. La vecindad de una regla de asociación está dada por el conjunto de todas las reglas que se encuentran dentro de cierta distancia de la primera. La fórmula propuesta para el cálculo de la distancia se define de la siguiente forma.

Definición. *Distancia entre dos reglas* – Dados tres números reales no negativos δ_1, δ_2, δ_3, se define la distancia ($Dist(R_1, R_2)$) entre dos reglas R_1: $X_1 \Rightarrow Y_1$ y R_2: $X_2 \Rightarrow Y_2$ como:

$$Dist(R_1,R_2) = \delta_1 * \left|(X_1 \cup Y_1)\Theta(X_2 \cup Y_2)\right| + \delta_2 * \left|X_1 \Theta X_2\right| + \delta_3 * \left|Y_1 \Theta Y_2\right|. \quad (20)$$

El operador Θ denota la diferencia simétrica entre dos conjuntos; o sea, $X \Theta Y$ sintetiza la expresión $(X-Y)\cup(Y-X)$.

Esta definición de distancia permite otorgar variadas escalas de importancia a las diferencias entre las diferentes partes de las reglas de asociación:

- Diferencias entre los conjuntos de ítems que conforman la regla.
- Diferencias entre los ítems que pertenecen al antecedente.
- Diferencias entre los ítems que pertenecen al consecuente.

Diferentes valores para δ_1, δ_2, δ_3, pueden ser utilizados para reflejar las preferencias de los usuarios. Considerando valores que cumplan $\delta_1 > \delta_2 > \delta_3$, se estaría otorgando más peso a las diferencias entre los conjuntos de ítems que conforman la regla en cuestión, favoreciendo así que dos reglas que estén formadas por el mismo conjunto de ítems aun cuando tengan diferencias en el antecedente o consecuente estén más cercanas (distancia menor) que aquellas formadas con un conjunto de ítems diferentes. Los autores proponen como valores posibles los siguientes:

$$\delta_1 = 1, \delta_2 = \frac{n-1}{n^2}, \delta_3 = \frac{1}{n^2}. \quad (21)$$

Definición. *La r-vecindad de una regla de asociación* – La *r*-vecindad de una regla de asociación R_0 (r > 0), denotada como $N(R_0, r)$, se define como el siguiente conjunto:

$$N(R_0,r) = \{R \mid Dist(R,R_0) \le r, R \text{ es una regla potencial}\}. \quad (22)$$

Considerando lo anterior, el interés de una regla está determinado por su distinción respecto a las reglas que forman su vecindad. Esta distinción se expresa a través de dos conceptos introducidos por Dong y Li, regla interesante con confianza inesperada y regla interesante con vecindad esparza.

5.2.5.1 Regla interesante con confianza inesperada

La confianza inesperada de una regla es un concepto que se define a través de la media y la desviación estándar de las confianzas de las reglas que se encuentran en su vecindad.

Si se considera a M como el conjunto de las reglas de asociación que satisfacen los valores de umbrales *minsup* y *minconf*, R_0 una regla de asociación ($R_0 \in M$) y $r > 0$, entonces la media y la desviación estándar en una r-vecindad se define de la siguiente forma.

Definición. *Media de la confianza en una r-vecindad* – La media de la confianza en la r-vecindad de una regla R_0 se define como el promedio de las confianzas de las reglas del conjunto $M \cap N(R_0, r) - \{R_0\}$; este valor se denota como $avg\_conf(R_0, r)$.

Definición. *Desviación estándar de la confianza en una r-vecindad* – La desviación estándar de la confianza en la r-vecindad de una regla R_0 se define como la desviación estándar de las confianzas de las reglas del conjunto $M \cap N(R_0, r) - \{R_0\}$; este valor se denota como $std\_conf(R_0, r)$.

Considerando estos dos conceptos, se puede introducir la definición de regla interesante con confianza inesperada.

Definición. *Regla interesante con confianza inesperada* – Una regla R_0 es interesante, del tipo de confianza inesperada, en su r-vecindad si el valor dado por la siguiente expresión es alto.

$$\left\| conf(R_0) - avg\_conf(R_0, r) \right| - std\_conf(R_0, r) \right| \quad (23)$$

El criterio de alto pudiera estar dado por un valor de umbral mínimo. Como puede observarse, esta medida permite seleccionar reglas cuyas confianzas se diferencien significativamente, tanto por exceso como por defecto, de la media de su vecindad.

5.2.5.2 Regla interesante con vecindad esparza

La vecindad esparza se produce cuando la r-vecindad de una regla representa un conjunto de muchas reglas potenciales pero muy pocas de ellas cumplen con los valores de umbral mínimo establecidos. En este caso se considera que la regla es interesante por encontrarse aislada.

Definición. *Regla interesante con vecindad esparza* – Una regla R_0 es interesante, del tipo aislada, si su r-vecindad es inesperadamente esparza; o sea, si el número de reglas potenciales en el conjunto $N(R_0, r)$ es grande, pero el número de reglas minadas $|M \cap N(R_0, r)|$ es relativamente pequeño.

La medida del interés, dado por la confianza inesperada o la vecindad esparza, representa otra forma de descubrir reglas de asociación interesantes. En esta medida, el interés de una regla está dado no sólo por su soporte y confianza, sino además por los soportes y confianzas de las reglas de su vecindad. Esta medida ofrece también una forma de particionamiento del conjunto *M*.

No obstante los aspectos teóricos y las consideraciones razonables en su presentación, esta medida requiere una mayor validación experimental para su aceptación y aplicabilidad. Además, su empleo requiere una precisa elección del tamaño de la vecindad, lo que adiciona elementos de complejidad al método propuesto.

5.2.6 Confianza neta

La confianza neta fue propuesta por Kwang-Il Ahn y Jae-Yearn Kim en el 2004 con el propósito de estimar la importancia del consecuente de una regla respecto a su antecedente [Ahn, 2004]. Su definición puede enunciarse de la siguiente forma.

Definición. *Confianza neta de una regla de asociación* – La confianza neta (*netconf*) de una regla de asociación se define según la siguiente expresión:

$$netconf(X \Rightarrow Y) = Conf(X \Rightarrow Y) - Conf(\neg X \Rightarrow Y). \quad (24)$$

Tras esta definición se tiene la siguiente consideración, cuando Y es frecuente en transacciones que no contienen a X, la regla $X \Rightarrow Y$ suele no ser interesante aún cuando Conf($X \Rightarrow Y$) es alta. La confianza neta permite resolver esta situación. El dominio de esta medida es el intervalo [-1, 1], lo cual se puede deducir de su formulación.

La confianza neta permite medir la independencia entre consecuente y antecedente. Esto puede verificarse con las siguientes transformaciones:

$$netconf(X \Rightarrow Y) = \frac{Sup(X \cup Y)}{Sup(X)} - \frac{Sup(Y) - Sup(X \cup Y)}{1 - Sup(X)} = \frac{Sup(X \cup Y) - Sup(X) * Sup(Y)}{Sup(X) * (1 - Sup(X))} \quad (25)$$

Según (25), cuando netconf($X \Rightarrow Y$) = 0 se verifica la independencia entre esos *itemsets*, ya que Sup($X \cup Y$) = Sup(X) * Sup(Y).

También se puede verificar que la confianza neta permite discriminar la dirección de la dependencia, al comprobarse que usualmente netconf($X \Rightarrow Y$) ≠ netconf($Y \Rightarrow X$). Sus autores resaltan que este indicador permite medir la fuerza de la implicación en ambas direcciones. No obstante, tal afirmación no es evaluada en toda su dimensión.

Otra propiedad que resaltan Ahn y Kim es que si netconf($X \Rightarrow Y$) es positivo entonces la dependencia es positiva y si es negativa la dependencia también es negativa. Esto se puede comprobar por la expresión (25). Por otra parte, sus autores muestran que la confianza neta permite medir la dependencia positiva y negativa en una escala simétrica respecto a Sup($X \cup Y$). Estas dos propiedades confieren a esta medida características interesantes cuando se precise analizar dependencias negativas, limitaciones presentes en el factor de certeza.

5.2.7 Interés basado en probabilidades y la teoría de la información

La determinación de las dependencias entre variables es un área bien estudiada en estadística. La prueba de hipótesis es un método a menudo usado en estos casos; para probar la dependencia entre variables se propone una hipótesis nula sobre la independencia de las variables. Una de las pruebas preferidas para ello es el χ^2 de Pearson [Sese, Morishita] [Tan, 2000].

Para aplicar esta prueba en una regla del tipo $X \Rightarrow Y$ se precisa de su tabla de contingencia.

Tabla 3. Tabla de contingencia de X y Y

	Y	$\neg Y$	
X	f_{XY}	$f_{X\neg Y}$	f_X
$\neg X$	$f_{\neg XY}$	$f_{\neg X\neg Y}$	$f_{\neg X}$
	f_Y	$f_{\neg Y}$	

La expresión para esta prueba está dada de la siguiente forma:

$$\chi^2 = \sum_{i\in\{X,\neg X\},j\in\{Y,\neg Y\}} \frac{(f_{ij} - E(f_{ij}))^2}{E(f_{ij})}. \quad (26)$$

Los valores esperados se definen considerando la hipótesis nula; o sea, $E(f_{ij}) = n * (f_i/n) * (f_j/n)$, siendo n la cantidad de transacciones.

Este método, aunque estadísticamente sólido, tiene algunos inconvenientes, siendo el principal el que no permite decidir la fuerza y dirección de la dependencia entre consecuente y antecedente.

Otros métodos que resuelven la anterior limitación son el coeficiente ϕ de Pearson, el coeficiente λ de Goodman y Kruskal y la incertidumbre basada en la teoría de la información [Sese, Morishita] [Tan, 2000], [Jaroszewicz, 2002]. No obstante, estos métodos tienen como inconveniente la complejidad computacional de sus formulaciones, sobre todo cuando se tratan grandes volúmenes de datos.

6. Métodos de generación de reglas de asociación

Para la generación de las reglas de asociación existen diferentes métodos. A continuación se analizarán los más significativos.

6.1 Método clásico de generación de reglas de asociación

El método clásico de generación de las reglas de asociación fue propuesto por Agrawal y Srikant [Agrawal, 1994]. En este método, por cada *itemset* frecuente se generan todas las reglas posibles considerando todos los subconjuntos del *itemset*.

La lógica de este método se muestra en la siguiente figura.

Algoritmo: GenRules
Entrada: L – Conjunto de itemsets frecuentes, formado por $\cup L_k$ obtenidos por algún algoritmo
Salida: RA – Conjunto de las reglas de asociación

```
1)  forall itemset Z ∈ L do
2)    forall itemset X ⊂ Z and X ≠ ∅ do
3)      if Z.Support / X.Support > minconf then
4)        RA = RA ∪ {X ⇒ Z\X};
```

Fig. 38. Pseudocódigo del algoritmo GenRule

Nótese que la cantidad de reglas de asociación que pudieran generarse es considerablemente alta. Una alternativa a esto es la generación de los tipos de reglas presentadas en el epígrafe 5.1.

6.2 Métodos de generación de RAR y RAMM

Marzena Kryszkiewicz, quien propuso las RAR y las RAMM, definió los algoritmos que permiten generar estos tipos de regla [Kryszkiewicz, 1998a]. A continuación se definen estos algoritmos.

6.2.1 Generación de reglas de asociación representativas

Las reglas de asociación representativas, descritas en el epígrafe 5.1.1, conforman el conjunto mínimo a partir del cual se puede generar todas las reglas de asociación válidas. El algoritmo propuesto por Kryszkiewicz encuentra su validez en dos propiedades derivadas de la definición de RAR dada en (10). Estas propiedades son las siguientes.

Propiedad. Una regla $X \Rightarrow Y$ es una RAR si y sólo si no existe ninguna otra regla $X' \Rightarrow Y'$ que verifique alguna de las siguientes expresiones:

(**27**)

$$X' = X \wedge X \cup Y \subset X' \cup Y';$$

$$X' \subset X \wedge X \cup Y = X' \cup Y'. \quad (\mathbf{28})$$

O sea, una regla de asociación es representativa si no existe ninguna otra regla que involucre un supraconjunto de todos los ítems pero con igual antecedente, o si no existe ninguna otra regla que involucre al mismo conjunto de ítems pero con un antecedente menor.

Propiedad. Toda regla de asociación representativa satisface la siguiente expresión:

$$X \Rightarrow Z \setminus X \in \mathrm{RAR} \equiv \neg\exists Z', Z \subset Z' \wedge Sup(X) = Sup(Y). \quad (\mathbf{29})$$

Esta propiedad se comprueba fácilmente al observar que si existiera una regla $X \Rightarrow Z \backslash X$ con esas características, entonces tendría el mismo soporte y confianza que $X \Rightarrow Z \backslash X$, siendo la primera una regla de asociación válida que verifica (27).

A continuación se muestra el algoritmo de generación de las RAR; en este se utiliza la variable A_i para los *itemsets* de tamaño i que pueden emplearse como antecedentes de las reglas.

Algoritmo: FastGenAllRepresentatives
Entrada: L – Conjunto de itemsets frecuentes, formado por $\cup L_k$ obtenidos por algún algoritmo
Salida: RAR – Conjunto de reglas de asociación representativas

```
 1)  forall itemset Z = {Z1, …, Zk} ∈ Lk, k ≥ 2 do begin
 2)    maxsup = max( { Z'.Support | Z ⊂ Z' ∈ Lk+1 } ∪ {0} );
 3)    if (Z.Support ≠ maxSup) then begin
 4)      A1 = { {Z1}, {Z2}, ...,{Zk} };    // Create 1-antecedents
 5)      for ( I = 1; Ai ≠ ∅ and i < k; i++ ) do begin
 6)        forall i-antecedent X ∈ Ai do begin
 7)          "find Y ∈ Li such that Y = X";
 8)          X.Support = Y.Support;
 9)          // Is X ⇒ Z\X a valid AR?
10)          if Z.Support / X.Support ≥ minconf then
11)            // Isn´t any longer AR X ⇒ Z'\X that cover X ⇒
    Z\X?
12)            if maxsup / X.Support < minconf then begin
13)              RAR = RAR ∪ {X ⇒ Z\X};
14)              // antecedents of RAR are not extended
15)              Ai = Ai \ {X};
16)            end
17)          Ai+1 = apriori_gen(Ai);    // Compute (i+1)-antecedent
18)        end
19)      end
20)    end
21)  end
```

Fig. 39. Pseudocódigo del algoritmo FastGenAllRepresentatives

El algoritmo FastAllGenRepresentatives toma como entrada todos los *itemsets* frecuentes generados y calcula todas las reglas de asociación representativas. Para ello, se toma cada *k-itemset* frecuente Z, a partir del cual pueden generarse k reglas. Primero, se calcula el máximo (*maxsup*) de los soportes de los (k+1)-*itemsets* frecuentes supraconjuntos de Z. Si el soporte de Z es igual a *maxsup*, ninguna RAR puede ser generada, según la propiedad descrita en (28). Si en el paso 12 se comprueba que *maxsup* / X.Support ≥ *mnconf* entonces $X \Rightarrow Z \backslash X$ no es una RAR, ya que existe un supraconjunto que puede formar una RA válida que contradice la propiedad dada por (26). En el paso 15 se elimina al antecedente X de la regla incluida en *RAR*, evitando la verificación de la propiedad dada por (27).

Observe que le procedimiento "apriori_gen" del paso 17 coincide con el procedimiento de igual nombre utilizado en el algoritmo Apriori descrito en uno de los primeros epígrafes. Este procedimiento garantiza extender en un ítem cada uno de los antecedentes no considerados, teniendo en cuenta todas las combinaciones permitidas por *Z*.

6.2.2 Generación de reglas de asociación de mínimo antecedente y máximo consecuente

El algoritmo propuesto por Kryszkiewicz para la generación de las RAMM sigue la siguiente lógica:

1. Calcular el conjunto *RAR*, según el algoritmo FastAllGenRepresentatives.
2. Eliminar de *RAR* las reglas que no satisfacen las condiciones de mínimo antecedente y máximo consecuente.

6.3 Método de generación de reglas de asociación generalizadas

El problema de descubrir las reglas de asociación generalizadas se descompone en tres partes [Srikant, 1995]:

1. Encontrar todos los *itemsets* frecuentes.
2. Usar los itemsets frecuentes para generar las reglas deseadas.
3. Eliminar todas las reglas no interesantes del conjunto.

Para el cálculo de los itemsets frecuentes Srikant *et al.,* proponen tres algoritmos, los cuales serán analizados a continuación.

6.3.1 Algoritmo básico

En este método, para verificar si una transacción *T* soporta el *itemset X*, se incluye en *T* todos los ancestros de cada ítem presente en la transacción. Sea T' la transacción extendida por sus ancestros, entonces *T* soporta a *X* si T' es un supraconjunto de *X*. Con esta extensión se puede determinar los *itemsets* frecuentes mediante cualquiera de los métodos ya descritos.

6.3.2 Algoritmo Cumulate

El nombre de este algoritmo sugiere que todos los *itemsets* de cierto tamaño son contados en una pasada. Este agrega al método anterior las siguientes optimizaciones:

- Filtrar los ancestros que serán añadidos a las transacciones:
  Los ancestros añadidos son sólo aquellos que estén al menos en un *itemset* candidato de la iteración actual. Debe recordarse que los algoritmos de búsqueda de los conjuntos candidatos por lo general generan *itemset* de tamaño *k* a partir de conjuntos frecuentes de tamaño *k*-1. De esta forma se está eliminando los ancestros para todos aquellos ítems que no pertenecen a ningún *itemset* candidato. Además, si un ítem original no se encuentra en ningún *itemset* frecuente puede ser también eliminado.
- Realizar un pre-cálculo de los ancestros de cada ítem:
  Con esto se evita el recálculo de los ancestros de cada ítem atravesando el grafo de taxonomías, considerando sólo aquellos que están presentes en alguna transacción.
- Podar los *itemsets* que contengan un ítem y su ancestro:
  Esto es posible pues el soporte de un *itemset X* que contiene tanto a *x* como a su ancestro $\hat{x}$ será igual al del *itemset* $X - \{\hat{x}\}$. Además, si F_k, el conjunto de los *k-itemsets* frecuentes, no incluye ningún *itemset* que contiene un ítem y su ancestro, entonces el conjunto de *itemsets*

candidatos C_{k+1} tampoco los incluirá, pues estos serán podados por contener subconjuntos infrecuentes.

6.3.3 Algoritmo Stratification

La motivación de este algoritmo para el cálculo de las RAG parte de la siguiente idea. Sean los *itemsets* X_1 = {Ropa, Zapato}, X_2 = {Abrigo, Zapato} y X_3 = {Chaqueta, Zapato} los candidatos para *itemsets* frecuentes, donde "chaqueta" es un concepto más específico que (es un tipo de) "abrigo" y este, a su vez, más específico que "ropa". Si se conoce que X_1 no cumple con el umbral de soporte mínimo establecido, entonces no tenemos que contar las apariciones de X_2 y X_3 pues estos tampoco la cumplirán. Luego, en vez de contar las apariciones de todos los candidatos de un tamaño determinado en una misma iteración, sería más rápido contar primero las apariciones de X_1, y si este cumple con el soporte mínimo entonces contar las apariciones de X_2 y en caso de que este cumpliera, pues se contaría las del candidato X_3. La principal desventaja que tiene esta aproximación es el costo adicional de realizar múltiples iteraciones sobre la base de datos. En el desarrollo de este algoritmo, teniendo en cuenta la heurística planteada anteriormente, se presentan tres versiones diferentes las cuales se comentan a continuación.

7.3.3.1 Versión Stratify

Esta versión considera el orden parcial que induce una taxonomía sobre el conjunto de *itemsets*. Para cada *itemset X* se determina su nivel de profundidad basado en la siguiente fórmula.

$$prof(X) = \begin{cases} 0 & \text{, si } X \text{ no es hijo de ningún itemset} \\ \max\left(\left\{prof(\hat{X}) \mid \hat{X} = parent(X)\right\}\right) + 1 & \text{, en otro caso} \end{cases} \quad (\mathbf{30})$$

La idea general de esta versión es sencilla, se comienza primeramente contando las apariciones de todos los *itemsets* candidatos C_0 que están en el nivel de profundidad cero. Luego de eliminar aquellos candidatos que sean descendientes de los *itemsets* de C_0 que no cumplan con el soporte mínimo, se pasa a calcular el soporte de los que quedaron con nivel de profundidad uno (C_1). O sea, la heurística seguida es la siguiente, determinar los soportes de los candidatos de profundidad i (C_i) y eliminar todos los *itemsets* descendientes no frecuentes. Esta heurística se repite hasta terminar de contar el soporte de todos los *itemsets* candidatos de la iteración. Es importante plantear que, en función de reducir la cantidad de iteraciones sobre la base de datos, puede tomarse en cuenta de que si la cantidad de candidatos en el nivel de profundidad n es relativamente pequeña, entonces pueden procesarse a la vez los candidatos a diferentes niveles de profundidad (n, n+1,...).

Existe un compromiso entre la cantidad de *itemsets* procesados y la cantidad de pasadas sobre la base de datos para el cálculo del soporte. En un extremo se puede realizar una pasada sobre la base de datos para los candidatos de cada nivel, minimizando la cantidad de *itemsets* candidatos a procesar cada vez pero perdiendo tiempo en múltiples pasadas sobre la base de datos. En el otro extremo, se tiene procesar en una única pasada todos los candidatos, idea considerada en *Cumulate*, disminuyendo el costo de múltiples pasadas sobre la base de datos pero procesando muchos *itemsets* cuyos ancestros no tienen soporte mínimo. Una heurística alternativa (determinada empíricamente) utilizada para mejorar esta versión es que en cada iteración se procese al menos el 20 % del conjunto de candidatos.

6.3.3.2 Versión Estimate

Esta versión considera el hecho que los *itemsets* candidatos de alto nivel en la jerarquía que no son frecuentes determinan la no necesidad de calcular el soporte de sus itemsets descendientes. Para determinar los itemsets de alto nivel no frecuentes, se propone la realización de un muestreo de la base de datos.

Considerando lo anterior, esta versión primero determina los soportes de los candidatos en la muestra seleccionada, definiendo un conjunto de candidatos (denotado como C'_k) con los que se espera cumplan con el soporte mínimo y con los que no lo cumplan pero que sean hijos de alguno que sí se espera que lo cumpla. Como esto fue decidido producto de un muestreo, pudiera ocurrir que un *itemset* candidato, que se esperaba no cumpliera el soporte mínimo, sí lo cumple al calcular su soporte en la base de datos completa; en este caso, se realizaría una iteración extra con los descendientes de ese itemset (este conjunto se denota como C''_k).

Con el objetivo de reducir el efecto del error de muestreo, la implementación realizada por los autores del método incluye en el conjunto de candidatos C'_K aquellos itemsets cuyos soportes en la muestra sean superior al 90% del soporte mínimo y los itemsets no frecuentes en la muestra pero cuyos padres tengan un soporte superior al 90% del soporte mínimo.

6.3.3.3 Versión EstMerge

La versión *EstMerge* requiere una segunda iteración con el conjunto C''_k debido a los posibles errores derivados de una estimación por muestreo; sin embargo, el tamaño de este conjunto es usualmente pequeño. Por tales motivos, en esta versión se propone procesar esos *itemsets* en la iteración k+1; o sea, junto con procesamiento del conjunto C_{k+1}. Además, se propone como heurística suponer a todos esos *itemsets* como frecuentes y generar el conjunto de los candidatos C_{k+1} a partir de los *itemsets* frecuentes obtenidos de C'_k y todos los candidatos de C''_k.

La heurística antes mencionada pudiera aumentar el número de candidatos generados, pero no afecta la corrección del algoritmo. A continuación se muestra el pseudocódigo de esta versión, el cual incluye además las optimizaciones propuestas en la versión *Cumulate*.

Algoritmo: EstMerge

```
1)  L_1 ={ Frequent 1-itemsets };
2)  D_s, = "Sample of the database in the first pass";
3)  k = 2;
4)  C''_1 = ∅; // Candidates of size k to be counted with k+1
5)  while L_{k-1} ≠ ∅ or C''_{k-1} ≠ ∅ do begin
6)    C_k = "Candidates of size k generated from L_{k-1} ∪ C''_{k-1}";
7)    "Calculate X.estSupport, the support of each X in C_k over
      D_s";
8)    C'_k = { X ∈ C_k | X.estSupport > 0.9*minsup ∨ ( X´ =
      parent(X)
                        X´ ∈ C_k ∧ X´.estSupport > 0.9*minsup ) };
9)    "Calculate X.Support, the full support for each X in C'_k ∪
      C''_{k-1}";
10)   C_k = C_k \ { X∈C_k | ∀X´∈C´_k, X´=ancestor(X) ∧ X´.Support ≤
      minsup };
11)   C''_k = C_k \ C'_k ;
12)   L_k = { X ∈ C´_k | X.Support > minsup };
13)   L_{k-1} = L_{k-1} ∪ { X ∈ C''_{k-1} | X.Support > minsup };
14)   k = k+1;
15) end
16) Answer = ⋃_k L_k;
```

Fig. 40. Pseudocódigo de la versión EstMege del algoritmo Stratification

Srikant *et al.,* analizaron el problema de la elección del tamaño de la muestra, presentando resultados importantes. En particular, que el tamaño de la muestra debe incrementarse con el decremento del soporte mínimo, y que la probabilidad de que el soporte estimado sea una fracción del soporte real depende del tamaño de la muestra y no del de la base de datos.

Nótese que los tres algoritmos anteriores trabajan con la misma filosofía del algoritmo Apriori, introduciendo únicamente el problema de la generación de conjuntos candidatos que mezclen itemsets de cualquier profundidad dada la taxonomía que los caracteriza. Sin embargo, el algoritmo básico descrito arriba evidentemente procesa muchos *itemsets* innecesarios si se considera la información de los conceptos involucrados en la taxonomía. El segundo algoritmo, aunque hace tres importantes optimizaciones teniendo en cuenta lo anterior, requiere en una pasada contar el soporte de todos los conjuntos candidatos de tamaño *k*, imposibilitando eliminar algunos candidatos dado el valor de su profundidad. Por último, el tercer algoritmo tiene en cuenta lo anterior (o sea, genera *itemsets* comenzando por los niveles superiores de la jerarquía), pero como máximo un 20% de los candidatos no son tomados en consideración, por lo que algunas reglas de asociación no podrían descubrirse. Los resultados experimentales, como era de esperar, arrojan que el de mejor resultado (en tiempo) es Stratification (en su versión *EstMerge*), seguido por *Cumulate*, con una gran diferencia ambos respecto al algoritmo básico.

Conclusiones

Como ha podido apreciarse, en este trabajo se ha realizado un estudio detallado de muchas de las consideraciones metodológicas y de los métodos de generación secuenciales de los *itemsets* frecuentes y del descubrimiento de las reglas de asociación, particularmente para bases de datos textuales, como son las fuentes noticiosas.

A su vez, se ha propuesto un nuevo método de generación de los *itemsets* frecuentes, llamado CBMine, el cual sigue un esquema descendente a lo ancho. Este método introduce una forma de representación de los *itemsets* frecuentes y una forma de compresión de los *bitmaps* verticales asociados con los itemsets, mostrando rendimientos, en tiempo, superiores a otros métodos conocidos, particularmente en base de datos grandes y esparzas.

Además, se realiza un estudio preliminar de los métodos paralelos de generación de los *itemsets* frecuentes, proponiéndose el algoritmo ParCBMine como una versión paralela del método antes mencionado.

Considerando la representación propuesta de compresión vertical, podrían evaluarse otras posibilidades de este método. Particularmente, como el método CBMine fue aplicado bajo un esquema tipo Apriori, sería conveniente evaluar sus posibilidades en combinación con los otros esquemas propuestos en la literatura. Estos y otros aspectos podrán ser incluidos en futuros trabajos.

Referencias Bibliográficas

[Agrawal, 1993] Agrawal R., Imielinski T., Swami A.N.: Mining Association Rules Between Sets of Items in Large Databases. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207-216, Washington, D.C., 1993.

[Agrawal, 1994] Agrawal R. and Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th VLDB Conf., 1994, pp. 487-499.

[Agrawal, 1996] Agrawal R. and Shafer J.C.: Parallel Mining of Association Rules design, implementation and experience, Technical Report RJ10004, IBM Research Report, February 1996.

[Agrawal, 1999] Agrawal R., Agrawal C., and Prasad V. V. V. Depth first generation of large itemsets for association rules. IBM Tech. Report RC21538, July 1999.

[Agarwal, 2000] Agarwal, R., Aggarwal C., and Prasad V.: A Tree Projection Algorithm For Generation of Frequent Itemsets. Journal of Parallel and Distributed Computing, 2000.

[Ahn, 2004] Ahn, KI, and Jae-Yearn Kim . Efficient Mining of Frequent Itemsets and a Measure of Interest for Association Rule Mining. *Journal of Information & Knowledge Managemen*t, Vol 3, No 3, 245-257, 2004.

[Bayardo, 1998] Bayardo R. J. Efficiently mining long patterns from databases. In SIGMOD, pp. 85-93, 1998.

[Berry, 1997] Berry, M and G Linoff. *Data Mining Techniques for Marketing, Sales, and Customer Support*. New York: John Wiley & Sons, Inc, 1997.

[Berzal, 2001] Berzal F., Blanco I., Sanchez D. and Vila M.A. A new framework to assess association rules. *Proceedings of the 4th International Conference on Intelligent Data Analysis*, 95-104, 2001.

[Brin, 1997] Brin, S, R Motwani, JD Ullman and S Tsur. Dynamic itemset counting and implication rules for market basket data. *Proceedings of the 1997 ACM-SIGMOD International Conference on Management of Data*, 255-264, 1997.

[Bodon, 2005] Bodon, F., Schmidt-Thieme, L.: The Relation of Closed Itemset Mining, Complete Pruning Strategies and Item Ordering in Apriori-based FIM algorithms (Extended version). In Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD) 2005, Porto, Portugal.

[Bodon, 2006] Bodon F. A C++ Frequent Itemset Mining Template Library. In: Bodon's Home page, May 10th, 2006. http://www.cs.bme.hu/~bodon/en/index.html

[Borgelt, 2003] Borgelt, C.: Efficient Implementations of Apriori and Eclat. In Proc. of the 1st Workshop of Frequent Item Set Mining Implementations (FIMI 2003), 2003.

[Burdick, 2001] Burdick, D., Calimlim, M. and Gehrke, J.: Mafia: A maximal frequent itemset algorithm for transactional databases. In Proc. of the ICDE 2001, Heidelberg, Germany, 2001.

[Calimlim, 2006] Calimlim M., Gehrke J. Himalaya Data Mining Tools: Mafia. In: Home page of Himalaya Data Mining Group, May 10th, 2006. http://himalaya-tools.sourceforge.net/

[Chen, 1996] Chen, M.S., Han, J. and Yu, P.S.: Data Mining: An Overview from a Database Perspective. IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No.6, Dec. 1996, pp. 866-883.

[Cheung, 1996a] Cheung, D.W., Han, J., Ng, V.T. and Wong, C.Y.: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. Proceedings of the 12th IEEE International Conference on Data Engineering, February 1996, 106-114.

[Cheung, 1996b] Cheung, D.W., Han, J., Ng, V.T., Fu, A.W.C., and Fu, Y.: A Fast Distributed Algorithm for Mining Association Rules, Proceedings of PDIS, 1996.

[Cheung, 1998] David W. Cheung and Yongqiao Xiao, Effect of Data Skewness in Parallel Mining of Association Rules, Proceedings of the 2nd Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 48-60, Melbourne, Australia, April 1998.

[Coenen, 2003] Frans Coenen, Paul Leng and Shakil Ahmed, T-Trees, Vertical Partitioning and Distributed Association Rule Mining, Proceedings ICDM 2003: 513-516, Melbourne, Florida, USA, 2003.

[Dong, 1998] Dong, G, and J Li.. Interestingness of discovered association rules in terms of neighbourhood-based unexpectedness. In X. Wu, R Kotagiri, and K Korb, editors. *Proceedings of the Second Pacific-Asia Conference on Knowlegde Discovery and Data Mining* (PAKDD'98), 72-86. Melbourne, Australia, April 1998.

[Fayyad, 1996] Fayyad, U.M., Piatetsky-Shapiro, G. and Smyth, P.: From Data Mining to knowledge Discovery: An Overview. Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996, pp 1-34.

[Feldman, 1996] Feldman R., Hirsh H.: Mining associations in text in the presence of background knowledge. Proc. of the Second International Conference on Knowledge Discovery from Databases, 1996.

[Feldman, 1998] Feldman, R. and Hirsh, H.: Finding Associations in Collections of Text. Machine Learning and Data Mining: Methods and Applications, R. Michalski, I. Bratko, and M. Kubat (editors), John Wiley and Sons, 1998, pp. 223-240.

[Freitas, 1998] Freitas, A.A., Lavington, S.H.: Mining Very Large Databases with Parallel Processing, Kluwer Academic Publishers, 1998.

[Freitas, 1998a] Freitas, A.A.: A Survey of Parallel Data Mining", Proc 2nd Int Conf on the Practical Applications of Knowledge Discovery and Data Mining, 1998.

[Gopalan, 2003] Gopalan R.P., Sucahyo Y.G.: Efficient frequent item set mining using a compressed prefix tree with pattern growth. Proceedings of 14th Australasian Database conference, Adelaide, Australia, 2003.

[Gopalan, 2004] Gopalan R.P., Sucahyo Y.G.: High Performance Frequent Patterns Extraction using Compressed FP-Tree. Proceedings of the SIAM International Workshop on High Performance and Distributed Mining, Orlando, USA, April 2004.

[Gardarin, 1998] Gardarin, G., Pucheral, P. and Wu F., "Bitmap based algorithms for mining association rules", Proc. of the BDA conf., pp 157-175, October 1998.

[Han, 1997] Han, E.H., Karypis, G., and Kumar,V.: Scalable Parallel Data Mining For Association Rules, Proceedings of the ACM SIGMOD Conference, pp. 277-288, 1997.

[Han, 2000] Han, J., Pei J., Yin Y.: Mining Frequent Patterns without Candidate Generation. Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), 2000.

[Hernández, 2005] Hernández Palancar j., Hernández León R., Medina Pagola J., Hechavarría Diaz A.: Mining Frequent Patterns Using Compressed Vertical Binary Representations. Proc. of Workshop Foundation of Semantic Oriented Data and Web Mining (FDM), ICDM'05, USA, 2005.

[Hipp, 2000] Hipp J., Güntzer U., Nakhaeizadeh G.: Algorithms for Association Rule Mining – A General Survey and Comparison. ACM SIGKDD, 2000.

[Holt, 1999] Holt J.D., Chung S.M: Efficient Mining of Association Rules in Text Databases. CIKM, ACM, 1999.

[Holt, 2001] Holt J.D., Chung S.M.: Multipass Algorithms for Mining Association Rules in Text Databases. Knowledge and Information Systems, Vol. 3, No.2, Springer-Verlag, 2001, pp. 168-183.

[Holt, 2002a] Holt J.D., Chung S.M.: Mining association rules using inverted hashing and pruning. Inf. Process. Lett. 83(4): 211-220 (2002)

[Holt, 2002b] John D. Holt, Chung S.M.: Mining Association Rules in Text Databases Using Multipass with Inverted Hashing and Pruning. Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2002), pp. 49-56.

[Jaroszewicz, 2002] Jaroszewicz S. and Simovici D.A: Pruning Redundant Association Rules Using Maximum Entropy Principle. Proc. of PAKDD 2002, Springer-Verlag Berlin Heidelberg, 2002.

[Joshi, 2000] Joshi M., Han E.H., Karypis, G., Kumar V.: Efficient Parallel Algorithms for Mining Associations. Large-scale Parallel and Distributed Data Mining, LNCS/LNAI, Vol. 1759, Springer-Verlag, 2000.

[Kryszkiewicz, 1998a] Kryszkiewicz M.: Representative Association Rules. Proceedings of PAKDD'98 Melbourne, Austria. Lectures Notes in Artificial Intelligence 1394. Research and Development in Knowledge Discovery and Data Mining. Springer-Verlag. 1998. pp 198-209.

[Kryszkiewicz, 1998b] Kryszkiewicz M.: Representative Association Rules and Minimum Condition Maximum Consequence Association Rules. In proceedings of Principles of Data Mining and knowledge Discovery, 1998. Second European Symposium, PKDD'98. Nantes, France. September 23-26. 1998.

[Kumar, 1997] Eui-Hong (Sam), Han, Karypis, G., Kumar, V.: Scalable Parallel Data Mining for Association Rules, Department of Computer Science, University of Minnesota, 1997.

[Lin, 2000] Lin T.Y.: Data Mining and Machine Oriented Modeling: A Granular Computing Approach. Journal of Applied Intelligence, Kluwer, Vol. 13, No. 2, 2000, pp. 113-124.

[Lin, 2002] Lin W.Y., Tseng M.C., Su J.H.: A Confidendece-Lift Support Specification for Interesting Associations Mining. Proc. of PAKDD 2002, Springer-Verlag Berlin Heidelberg 2002.

[Liu, 1999] Liu, B., Hsu, W., M, Y.: Mining association rules with multiple minimum supports. Proc. of ACM-SIGKDD´99, 1999, pp. 337-341.

[Montes, 2002] Montes y Gómez M.: Minería de texto empleando la semejanza entre estructuras semánticas. Tesis doctoral, INAOE, México, 2002.

[Mueller, 1995] Mueller, M.: Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 1995.

[Orlando, 2002] Orlando, S., Palmerini, P., Perego, R., and Silvestri, F.: A Scalable Multi-Strategy Algorithm for Counting Frequent Sets Washington, USA Pg. 19-30 Proceedings of the 5th Workshop on High Performance Data Mining, in conjunction with Second International SIAM Conference on Data Mining, April 2002

[Park, 1995] Park, J.S., Chen, M.S., and Yu, P.S.: Efficient Parallel Data Mining for Association Rules, Proceedings of the International Conference on Information and Knowledge Management, pp. 31-36, Baltimore, Maryland, 22-25 May 1995.

[Park, 1997] Park J.S., Chen M.S., Yu P.S.: Using a Hash-Based Method with Transanction Trimming and Database Scan Reduction for Mining Association Rules. IEEE Trans. on Knowledge and Data Engineering, Vol. 9, No. 5, pp. 813-825, 1997.

[Pietracaprina, 2003] Pietracaprina A., Zandolin D.: Mining frequent itemsets using Patricia Tries. Proc. of the Workshop on Frequent Itemset Mining Implementations, FIMI03, 2003.

[Savasere 1995] Savasere A., Omiecinski E., Navathe S.: An efficiente algorithm for mining association rules in large databases. Proc. of the 21st. Conf. on Very Large Databases (VLDB'95), Switzerland, 1995.

[Schuster, 2004] Schuster, A., and Wolff, R.: Communication-Efficient Distributed Mining of Association Rules. Data Mining and Knowledge Discovery, 8(2), March 2004.

[Shenoy, 2000] Shenoy, P., Haritsa, J.R., Sudarshan, S., Bhalotia, G., Bawa, M., Shah, D., "Turbo-Charging Vertical Mining of Large Databases". Proc. ACM SIGMOD Intl. Conf. Management of Data, May 2000.

[Shintani, 1996] Shintani, T., and Kitsuregawa, M.: Hash based parallel algorithms for mining association rules. In Proc. of 4th Int. Conf. on Parallel and Distributed Information Systems, 1996

[Shintani, 1998] Shintani, T., and Kitsuregawa, M.: Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy, Proceedings ACM SIGMOD International Conference on Management of Data, SIGMOD 1998, June 2-4, 1998, Seattle, Washington, USA, pp 25-36.

[Shortliffe, 1975] Shortliffe, E, and B Buchanan. A model of inexact reasoning in medicine. *Mathematical Biosciences 23*, 351-379, 1975.

[Shen 1999] Li Shen , Hong Shen , Ling Cheng, New algorithms for efficient mining of association rules, Information Sciences: an International Journal, v.118 n.1-4, p.251-268, Sept. 1999

[Silvertein, 1998] Silvertein, C, Brin, S., and Motwani. R.: Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1), 39-68, 1998.

[Skillicorn, 1999] Skillicorn, D.: Parallel Data Mining, Department of Computing and Information Science Queen's University, Kingston, 1999.

[Srikant, 1995] Srikant R., Agrawal R: Mining Generalized Association Rules. Proc. of VLDB. Zurich, Switzerland, 1995.

[Tan, 2000] Tan P.N., Kumar V.: Interestingness Measures for Association Patterns: A Perspective. University of Minnesota, Computer Science and Engineering, Technical Report No. 00-036, 2000.

[Zaki, 1996] Zaki, M.J., Ogihara, M., Parthasarathy, S., and Li, W.: Parallel Data Mining for Association Rules on Shared-Memory Multiprocessors, Technical Report TR 618, University of Rochester, Computer Science Department, May 1996.

[Zaki, 1997a] Zaki M.J., Parthasarathy S., Ogihara M., Li, W.: New algorithms for fast discovery of association rules. Proc. of the 3rd Int. Conf. on KDD and Data Mining (KDD'97), EU, 1997.

[Zaki, 1997b] Zaki M.J., Parthasarathy S., Ogihara M., Li, W.: New algorithms for fast discovery of association rules. Technical Report 651, Computer Science Department, The University of Rochester, New York, 1997.

[Zaki, 1997c] Zaki M.J., Parthasarathy S., Li, W.: A Localized Algorithm for Parallel Association Mining, Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures, 1997.

[Zaki, 1997d] Zaki M.J., Parthasarathy S., Ogihara, M., Li, W.: New Parallel Algorithms for Fast Discovery of Association Rules, Data Mining and Knowledge Discovery, Vol. 1, No. 4, pp. 343-373, December 1997.

[Zaki, 1999] Zaki, M.J.: Parallel and Distributed Association Mining: A Survey, IEEE Concurrency, October-December 1999.

[Zaki, 2001] Zaki M.J., Parthasarathy S., Ogihara M., Li, W.: Parallel Data Mining for Association Rules on Shared Memory Systems. February 28, 2001.

[Zaiane, 2001] Zaiane, O.R., El-Hajj, M., Lu, P.: Fast Parallel Association Rule Mining without candidacy generation, Technical Report TR01-12, Department of Computing Sciences, University of Alberta, Canada, 2001.

RT_003, Mayo 2007
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Margarita Ilisástigui Avilés
Diseño de Portada: DCG Matilde Galindo Sánchez
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
Ciudad de La Habana. Cuba. C.P. 12200
Impreso en Cuba