ISSN 2007-9737

FP-MAXFLOW: Un algoritmo para la minería de patrones relevantes de longitud máxima

Arnaldo Díaz-Molina, Luciano García-Garrido

Universidad de La Habana, Facultad de Matemática y Computación, La Habana, Cuba

arnaldomail1986@gmail.com, luciano@matcom.uh.cu

Resumen. Los algoritmos para el reconocimiento y clasificación de itemsets en bases de datos transaccionales aspiran a satisfacer un resultado indispensable en la obtención posterior de correlaciones en minería de asociación: encontrar los itemsets de ocurrencia más frecuente los que se constituyen en patrones para establecer posteriormente las correlaciones entre los mismos. El estado del arte muestra que la mayoría de los algoritmos obtienen el conjunto completo de itemsets en la búsqueda de patrones, mientas que otros obtienen los patrones de longitud máxima. Ambas estrategias tienen limitaciones para la obtención de las correlaciones: obtener el conjunto completo de itemsets implica analizar muchos resultados de poca relevancia, mientras que concentrarse solo en los de longitud máxima implica la pérdida de información importante. El objetivo de este trabajo es ofrecer un nuevo algoritmo, denominado FP-MAXFLOW, para obtener un conjunto de itemsets que tomados como patrones permiten establecer correlaciones entre conjuntos de itemsets relevantes de la mayor longitud posible y evitando redundancias. Tal resultado se obtiene con un solo recorrido de la base de datos. Los análisis comparativos demuestran que FP-MAXFLOW es competitivo con otros algoritmos que figuran entre los más utilizados.

Palabras clave. Bases de datos transaccionales, minería de asociación, reconocimiento de patrones, itemsets frecuentes.

FP-MAXFLOW: An Algorithm for Mining Maximum Relevant Patterns

Abstract. Algorithms for itemsets recognition and classification have been widely studied. The state of the art shows that most of the algorithms obtain all possible itemsets, others only obtain maximal ones. Both approaches have limitations for getting relevant correlations. Obtaining all itemsets imply many irrelevant patterns. By obtaining only maximal patterns important information could be ignored. The goal of this work is to offer an algorithm for obtaining the patterns which more efficiently could describe the correlations. The new algorithm, called FP-MAXFLOW is able to extract these, information efficiently with one database scan. The comparative studies show that it is a competitive solution according to other algorithms which are among the most used.

Keywords. Transactional databases, association mining, patterns recognition, frequent itemsets.

1. Introducción

La minería de reglas de asociación entre elementos en bases de datos (BD) (transaccionales), constituye en el presente una actividad relevante para la toma de decisiones en la solución de muchos problemas en diversos dominios [1].

Una caracterización del problema parte de considerar un BD transaccional, siendo cada transacción un conjunto finito de items (artículos). Las transacciones a su vez están compuestas de diversos (sub) conjuntos finitos de items (itemsets), los que pueden ocurrir repetidos en las diversas transacciones.

El objetivo general consiste en la extracción de aquellos itemsets que ocurren de manera frecuente en las transacciones. Un itemset es frecuente si cumple un valor límite de frecuencia, conocido como “minimum support” [2].

Dependiendo del dominio, el límite de frecuencia puede ser absoluto o relativo, teniendo en el segundo caso un significado probabilista.

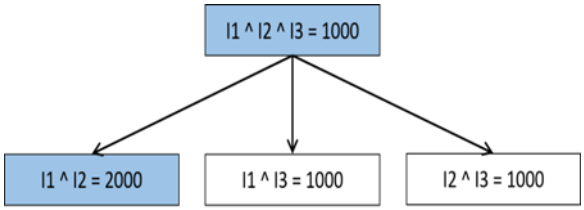

Fig. 1. Ejemplos de itemsets frecuentes maximales, cerrados y simples, donde:

I1 ^ I2 ^ I3=1000	(P1)
I1 ^ I2=2000	(P2)
I1 ^ I3=1000	(P3)
I2 ^ I3=1000	(P4)

Los límites de frecuencia deben ser cuidadosamente seleccionados: Un límite alto podría ignorar información importante. Un límite bajo implica la detección de conocimiento redundante, debido a que, como se ha demostrado [3] si un itemset es frecuente, todos sus subitemsets propios también lo son. Un itemset tiene en general una gran cantidad de subitemsets propios. Para un conjunto de 100 items, la cantidad total de combinaciones se puede calcular como:

$$\binom{100}{1}+\binom{100}{2}+\cdots+\binom{100}{100}=2^{100}-1\approx 1.27\times 10^{30}.$$

Para evitar esta explosión combinatoria, los itemsets se pueden especializar en maximales (maximal itemsets) o cerrados (closed itemsets).

En un BD, S, un itemset X es maximal si no existe ningún súper-itemset propio Y de X y X satisface el límite de frecuencia. Un itemset X es cerrado, si no existe ningún súper-itemset propio Y de X con la misma frecuencia de X, y X satisface el límite de frecuencia [3].

Según las definiciones anteriores, dados los siguientes itemsets se puede clasificar a P1 como maximal y a P2 como cerrado. En la figura 1, se pueden observar estas propiedades con mayor facilidad.

A partir del conjunto de itemsets cerrados obtenidos para un límite de frecuencia se puede generar todo el conjunto de itemsets frecuentes.

Por esta razón en muchos dominios es preferible extraer solo los itemsets cerrados y no el conjunto completo.

Sin embargo, el conjunto de itemsets cerrados no representa el conjunto de itemsets relevantes en todos los dominios. Existen dominios que requieren un nivel de exactitud alto en los resultados. En estos casos, se deben extraer los itemsets donde la correlación entre todos sus items es relevante sin necesidad de incluir a otros, y que por lo tanto representan patrones significativos para el dominio. Analizando un BD real, estos patrones más importantes siempre aparecerán de manera independiente en alguna proyección de BD. Como se explica en [3], una proyección de BD es un subconjunto de las transacciones originales que terminan en el mismo item.

Para la tabla de ejemplo 1 y una frecuencia límite de 2, el itemset {I1, I5} es frecuente, pero puede descartarse porque en la proyección de BD a la que pertenece: {I5}, siempre aparece acompañado por I2 o I3, por lo que representa un patrón incompleto.

Los itemsets intermedios que no cumplen la condición anterior podrían ser falsos positivos y aportar información incompleta. Suponga, por ejemplo, que en un dominio tecnológico se evalúa el estado del equipamiento para aumentar la productividad. Es irrelevante concluir que la relación entre dos equipos A y B es influyente en la productividad si la interacción de estos con otros equipos ha sido frecuente, siendo necesario ampliar el estudio para descubrir relaciones de mayor relevancia.

En función de este requisito para estos dominios, en este trabajo se determina que: *Un itemset es relevante si es frecuente, aparece de forma independiente en alguna proyección de BD y no es redundante*. Las condiciones de relevancia y redundancia de itemsets dependen de cada dominio. En este trabajo, *se clasifica a un itemset como no redundante si aporta información que no es posible inferir a partir de los supersets propios inmediatos*. Se clasifica a un superset propio A de B como inmediato si entre A y B no existe otro itemset C.

Dados los itemsets A= {1, 2}, B = {1, 2, 3} y C = {1, 2, 3, 5}, B es superset propio e inmediato de A porque entre A y B no existe otro itemset.

Sin embargo, C es superset propio de A pero no es inmediato porque entre A y C se encuentra B.

Tabla 1. BD de ejemplo 1

ID	Items
1	I1, I2, I5
2	I1, I2, I3, I5
3	I1, I2, I3
4	I1, I2, I7
5	I1, I3, I5

Si se representan los itemsets en una estructura arbórea, se puede observar que un itemset.

A es el prefijo de todos sus supersets propios inmediatos que siguen las mismas ramificaciones. Luego, si un itemset A tiene la misma frecuencia que la suma de todos sus supersets propios inmediatos S_A, y que son sus descendientes en la estructura arbórea, entonces se puede concluir que la frecuencia de A se puede inferir a partir de S_A, y A puede descartarse en los resultados porque es redundante. Por ejemplo, dados los itemsets A = {I1, I2=4}, B = {I1, I2, I3=2} y C = {I1, I2, I5=2}, se puede observar que B y C son supersets propios inmediatos de A, y que la suma de sus frecuencias es igual a la frecuencia de A, por lo que A es redundante y puede inferirse analizando las frecuencias de B y C. Para comprender mejor estas propiedades considere el BD de la tabla 1.

Dado un límite de frecuencia igual a 2, un algoritmo para extraer todos los itemsets frecuentes obtendría:

I1 ^ I2 ^ I5=2	(P1)
I1 ^ I2 ^ I3=2	(P2)
I1 ^ I3 ^ I5=2	(P3)
I1 ^ I2 =4	(P4)
I1 ^ I3 =3	(P5)
I1 ^ I5 =3	(P6)
I2 ^ I3 =2	(P7)
I2 ^ I5 =2	(P8)
I3 ^ I5 =2	(P9)
I1=5	(P10)
I2=4	(P11)
I3=3	(P12)
I5=3	(P13)

Un algoritmo para extraer itemsets maximales solo obtendría a P1, P2 y P3, con lo cual se pierden relaciones importantes. Un algoritmo para obtener itemsets cerrados eliminaría a P7, P8 y P9 porque son subsets propios de P1, P2 y P3, respectivamente, y tienen la misma frecuencia. Del mismo modo eliminaría a P11, P12 y P13 debido a los supersets propios P4, P5 y P6, respectivamente. El listado de itemsets cerrados sería:

I1 ^ I2 ^ I5=2	(P1)
I1 ^ I2 ^ I3=2	(P2)
I1 ^ I3 ^ I5=2	(P3)
I1 ^ I2 =4	(P4)
I1 ^ I3 =3	(P5)
I1 ^ I5 =3	(P6)
I1=5	(P10)

Todos los itemsets del listado anterior no son relevantes. El itemset P6 aparece en la proyección de BD {I5}, pero siempre acompañado por los itemsets {I2}, {I2, I3} o {I3}, por lo que P6 por sí solo no es relevante. El itemset P10 aparece en todas las proyecciones de BD, pero siempre acompañado de otros itemsets, por lo que tampoco es relevante por sí solo. El itemset P4 aparece en la proyección {I2} y es relevante, pero tiene la misma frecuencia que la suma de P1 y P2, que son supersets propios inmediatos y tienen a P4 como prefijo, por lo que P4 es redundante y puede inferirse fácilmente a partir de P1 y P2.

El itemset P5 es relevante con respecto a la proyección {I3}, y no es posible inferir su frecuencia a partir de sus supersets propios inmediatos. Por lo tanto, P5 es relevante y no redundante. De forma similar, P1, P2 y P3 son relevantes y no redundantes, pues no existe ningún superset propio inmediato que permita inferir sus frecuencias.

El listado de itemsets relevantes y no redundantes es:

I1 ^ I2 ^ I5=2	(P1)
I1 ^ I2 ^ I3=2	(P2)
I1 ^ I3 ^ I5=2	(P3)
I1 ^ I3 =3	(P5)

Los itemsets relevantes son los que mejor podrían explicar las correlaciones entre las variables de un BD dada la frecuencia mínima establecida.

Obtenerlos es un procedimiento complejo y es necesario aplicar técnicas de poda que permitan

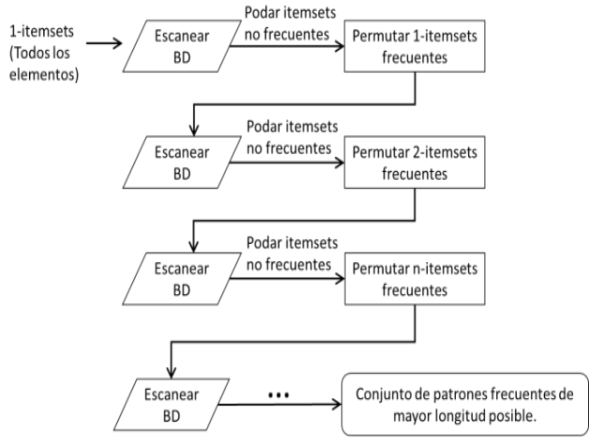

Fig. 2. Esquema de funcionamiento del algoritmo Apriori

reducir eficientemente el espacio de búsqueda. Pueden adoptarse varias estrategias.

Una estrategia simple sería obtener todo el conjunto de itemsets y luego realizar las operaciones de poda. Sin embargo, debido a la cantidad combinatoria de itemsets frecuentes esta operación es demasiado costosa y solo sería posible en BDs pequeñas.

El resto del artículo se organiza como sigue: primero se describen algunos de los trabajos más destacados relacionados con el tema de minería de itemsets frecuentes, luego se describe el algoritmo propuesto FP-MAXFLOW para la extracción eficiente de itemsets relevantes, a continuación, se presentan estudios comparativos entre el algoritmo propuesto y otros algoritmos del estado del arte, por último, se enuncian las conclusiones y las referencias bibliográficas.

2. Trabajos relacionados

Existen en la actualidad varios algoritmos para la minería de itemsets. Los más populares emplean la generación de candidatos o la construcción de modelos.

Una estrategia de búsqueda conocida como bottom-top identifica itemsets de longitud mayor a partir de otros ya identificados de longitud menor, un ejemplo de algoritmo que aplica esta estrategia es Apriori.

Otra estrategia conocida como top-bottom representa el procedimiento inverso y es más eficiente cuando existen itemsets frecuentes de longitud grande. Esta estrategia es utilizada fundamentalmente por los algoritmos que obtienen itemsets maximales. A continuación, se describen algunos de los algoritmos más relevantes del estado del arte.

2.1. Extracción con generación de candidatos

En la generación de candidatos los nuevos itemsets se generan mediante la combinación de otros que fueron identificados en etapas anteriores. El algoritmo Apriori, propuesto por R. Agrawal y R. Srikant en 1994, es el más utilizado actualmente para el reconocimiento de itemsets y tiene el valor adicional de que abrió una amplia línea de investigación y siempre debe ser objeto de reconocimiento. Se caracteriza por su robustez y facilidad de implementación [4]. Necesita realizar un recorrido completo del BD en cada iteración para actualizar las frecuencias de los itemsets e ignorar aquellos que no son frecuentes en futuras iteraciones. El algoritmo realiza iteraciones hasta que no puede encontrar más combinaciones que sean frecuentes. Los itemsets obtenidos en la iteración k se nombran k-itemsets, y se combinan para obtener los k+1-itemsets.

En la figura 2, se muestra el esquema general de este procedimiento. En función de este algoritmo se han propuesto algunas mejoras.

El algoritmo de Park, Chen y Yu [5], conocido como algoritmo PCY intenta mejorar el rendimiento de Apriori asumiendo que existe una gran cantidad de memoria desperdiciada en cada iteración. Utiliza tablas “hash” o diccionarios para reducir el tamaño de los candidatos k-itemsets, para k>1.

Otras variantes se orientan a descartar las transacciones que no contienen items frecuentes en la iteración k para formar itemsets en la iteración k+1. En este sentido, en [6], se propone que para cada k-itemset se mantenga una tabla con la dirección de las transacciones en donde fueron encontrados sus items, y cuando se generen los k+1-itemsets solo será necesario escanear las transacciones registradas en los k-itemsets combinados. Este método es costoso por la cantidad de memoria requerida.

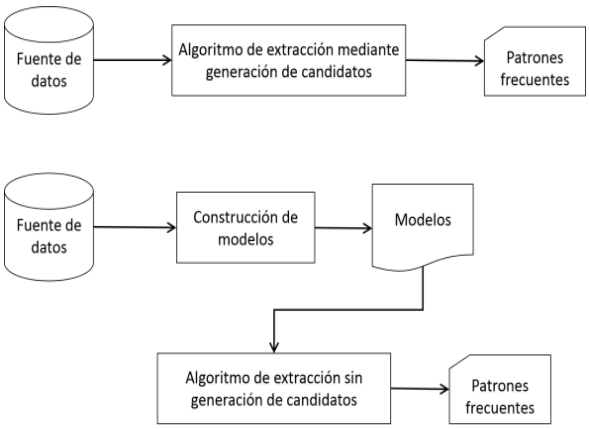

Fig. 3. Diferencias entre los enfoques con y sin generación de candidatos

Dos ventajas importantes son su facilidad de implementación y su eficiencia para generar los k+1-itemsets.

El particionamiento de los datos es otro tipo de estrategia ventajosa en dominios donde no es necesario obtener todo el conjunto de itemsets frecuentes, sino una muestra significativa [5]. El algoritmo "Randomized Algorithm" propone seleccionar de forma aleatoria subespacios de la masa de datos, asumiendo que estos registros describen el comportamiento general de todos los datos. Las particiones se almacenan en memoria y se utiliza cualquier algoritmo, como por ejemplo Apriori o PCY, para extraer el conjunto de itemsets frecuentes, donde el límite de frecuencia se ajusta al tamaño de las particiones. Una de las ventajas más importantes de "Randomized Algorithm" es que puede ser paralelizado con facilidad. Sin embargo, el uso excesivo de memoria puede ser un problema cuando las particiones son grandes. Otra limitación es que las muestras aleatorias podrían no contener los datos más importantes, sobre todo si existen relaciones estacionarias entre los itemsets.

La estrategia propuesta por Savasere, Omiecinski, y Navathe [7], conocida como SON, utiliza la técnica de particionamiento, pero en vez de unir todas las secciones aplica un algoritmo (puede ser Apriori, PYC o cualquier otro) a cada sección de forma independiente y los itemsets frecuentes son calculados mediante la operación de unión entre los resultados de cada sección.

En un segundo recorrido sobre BD se actualizan las frecuencias totales de los itemsets encontrados.

El algoritmo de Toivonen [8] selecciona una sección de los datos con una frecuencia límite pequeña. En la siguiente iteración actualiza las frecuencias no solo de los itemsets frecuentes sino de los bordes negativos. Los bordes negativos son aquellos itemsets que no son frecuentes, pero todos sus subitemsets propios inmediatos si lo son. Los subitemsets propios inmediatos de un itemset de longitud (n), se pueden calcular permutando sus items para obtener conjuntos no repetidos de longitud (n-1). Si no se encuentran bordes negativos para la muestra seleccionada el resultado es exacto y el algoritmo termina. En caso de existir alguno, hay que repetir el algoritmo para otra muestra y así sucesivamente.

La generación de candidatos puede ser ineficiente porque:

1. Es necesario generar una gran cantidad de candidatos mediante permutaciones, sobre todo si el límite de frecuencia es pequeño y existen itemsets frecuentes largos.
2. Es necesario escanear BD varias veces para actualizar las frecuencias de los itemsets.

Se han propuesto métodos que construyen modelos que codifican el contenido del BD y así el proceso de minería no se realiza sobre los datos originales, agilizando el tiempo de procesamiento. Estas variantes son más complejas de implementar en la mayoría de los casos.

2.2. Extracción sin generación de candidatos

Los algoritmos que siguen este esquema intentan construir un modelo que describa la información del BD. El procesamiento de estos modelos es más rápido que los recorridos sobre los datos originales, aunque este factor depende de la implementación. Otra ventaja de los modelos es que son construidos a conveniencia para que el proceso de minería sea más eficiente, a diferencia de los datos originales que pueden haber sido generados por sistemas transaccionales diseñados para la gestión de datos y no la extracción de información.

Para límites de frecuencia altos puede suceder que la estrategia de generación de candidatos sea

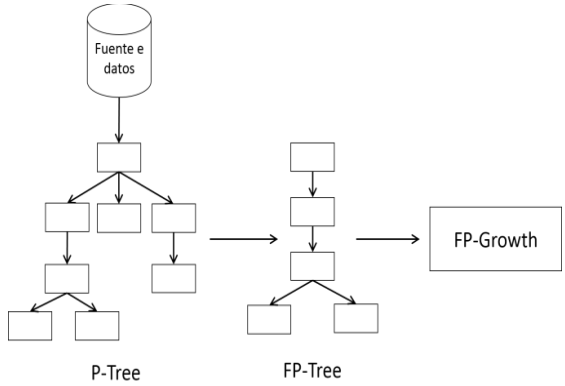

Fig. 4. Esquema de funcionamiento propuesto para el modelo P-Tree

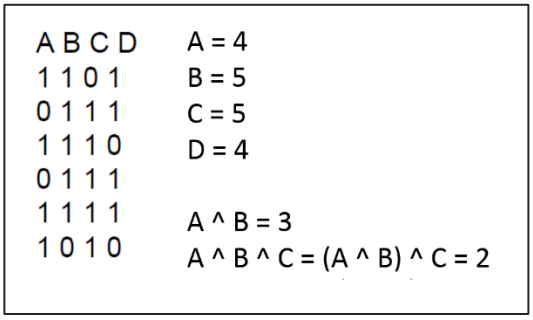

Fig. 5. Representación BITMAP y operaciones "AND" del algoritmo MAFIA

más eficiente que la construcción de modelos, debido a las operaciones de poda. Sin embargo, en la mayoría de los casos los algoritmos sin generación de candidatos han demostrado ser más eficientes en tiempo de procesamiento y en memoria utilizada. La diferencia de funcionamiento entre estos dos enfoques se puede observar en la figura 3.

El algoritmo FP-Growth, propuesto por Jiawei Han y otros autores en [9], es posiblemente el más popular. Recorre el BD al igual que el algoritmo Apriori en la primera iteración para almacenar en una tabla (L) la frecuencia de cada ítem. Los items son ordenados descendentemente en función de su frecuencia. En una segunda iteración sobre el BD construye una estructura arbórea llamada FP-Tree (Frequent Pattern Tree). El árbol FP-Tree almacena toda la información de forma resumida y es una de las estructuras más utilizadas actualmente en la minería de itemsets frecuentes.

La principal ventaja de esta estructura es que optimiza el espacio en memoria necesario para almacenar la información primaria. Tiene dos limitaciones importantes. La primera es la complejidad de procesamiento que implica un uso exhaustivo de procedimientos recursivos. La segunda es que si el BD cambia hay que reconstruir el árbol completo. No obstante, FP-Growth es uno de los algoritmos más eficientes para la minería de itemsets.

Algunas variantes mejoradas del algoritmo FP-Growth han sido publicadas. En [10], los autores proponen un algoritmo llamado FP-Growth*, que emplea una estructura llamada FP-array (Frequent Pair Array) que mejora el rendimiento del algoritmo FP-Growth en BDs dispersas y para límites de frecuencia bajos. Este algoritmo parte de la construcción inicial del FP-Tree y propone las nuevas estructuras como alternativa a las operaciones recursivas de FP-Growth. Según los autores el 80% del tiempo de procesamiento de FP-Growth se desperdicia en los recorridos transversales de los árboles. FP-Growth* mejora el rendimiento de su antecesor, pero la construcción de matrices implica mayor demanda de memoria. Al necesitar como parámetro un árbol FP-Tree, no se eliminan las limitaciones de esta estructura explicadas anteriormente.

Una propuesta que elimina por completo el empleo de árboles es SimpleARM [11]. Este algoritmo realiza un solo escaneo en el BD y utiliza matrices para codificar la información. Es efectivo para extraer itemsets de tamaños grandes en donde otros algoritmos como Apriori pueden ser ineficientes.

En un trabajo publicado en [12], los autores proponen una estructura llamada P-Tree que puede generarse con un solo escaneo de los datos originales. El nuevo modelo es similar a FP-Tree, pero almacena los items en el mismo orden de aparición. Su principal ventaja es que es posible actualizarlo sin necesidad de recorrer nuevamente todas las transacciones. El análisis propone construir el FP-Tree a partir del P-Tree, el cual puede estar almacenado completamente en memoria y/o en disco. En la figura 4 se muestra este enfoque:

El modelo P-Tree es más rápido de construir y procesar que el FP-Tree pero generalmente ocupa mayor espacio en memoria. No obstante, pueden

existir BDs en los que el FP-Tree no sea significativamente más pequeño que el P-Tree, lo cual dependerá del nivel de esparcimiento de los datos. Gracias a la alta disponibilidad de memoria de un computador convencional, en la mayoría de los casos es posible emplear esta estructura sin dificultades, lo cual también es demostrado por los autores en [12].

En [13] se propone un procedimiento que emplea redes neuronales ópticas para extraer itemsets en grandes BD. Para garantizar su eficiencia requiere implementaciones sofisticadas de paralelismo. Codifica las transacciones en bits para emplear menos memoria que los procedimientos convencionales, incluyendo Apriori y FP-Growth. Solo requiere recorrer el BD una vez.

El cálculo de frecuencias y la parametrización de la red se realizan simultáneamente. Sus limitantes consisten en la dificultad de las redes neuronales ópticas y en la necesidad de emplear hardware especializado para su implementación.

2.3. Extracción de itemsets maximales y cerrados

Como los itemsets frecuentes maximales tienen incluidos todas las combinaciones de itemsets frecuentes, algunos enfoques se concentran solo en extraer los maximales y así optimizar espacio en memoria y en disco.

En [14], se propone el algoritmo MaxMiner que extiende al algoritmo Apriori para extraer solo los itemsets maximales. En vez de realizar solo la poda de frecuencia, descarta además cualquier candidato que sea subitemset propio de un candidato frecuente de longitud máxima. Aunque esta técnica de poda reduce el tiempo de procesamiento, aún es necesario recorrer el BD varias veces.

MAFIA es un algoritmo propuesto en [15], utiliza una representación matricial binaria referida como representación BITMAP, donde la frecuencia de un itemset se encuentra esparcida en una columna. Las frecuencias de itemsets combinados se calculan mediante la operación “AND” entre las columnas. En la figura 5 se muestra la estrategia de cálculo de este algoritmo.

Las operaciones de MAFIA pueden ser costosas en BD con muchas transacciones.

Otros dos algoritmos son FP-MAX [16] que extiende a FP-Growth empleando un modelo adicional llamado MFI-Tree (Maximal Frequent Itemset Tree) que codifica solo los itemsets de longitud máxima; y una técnica propuesta en [17] que realiza operaciones de poda mediante intersecciones entre itemsets.

Existen otros algoritmos para extraer itemsets cerrados. El algoritmo AprioriClose es importante por razones históricas, ya que fue el primer algoritmo publicado para extraer itemsets cerrados. Otros algoritmos más eficientes son FPClose, Charm y LCM. Los algoritmos FPMax, FPClose, Charm y LCM son descritos en los estudios comparativos de este trabajo.

3. FP-MAXFLOW

FP-MAXFLOW es el algoritmo propuesto en este trabajo. Permite extraer eficientemente los itemsets más relevantes. Su desarrollo está inspirado en los algoritmos FP-Growth y Apriori.

Requiere dos parámetros de entrada, los registros de BD y el límite de frecuencia. Compacta las transacciones originales en modelos de datos que agilizan el tiempo de procesamiento. Propone tres estructuras de datos fundamentales: BP-Tree, BP-TransactionTree y BP-PatternTree. BP-Tree es una extensión de la estructura P-Tree propuesta en [12], que adiciona enlaces para agilizar los recorridos verticales por técnicas iterativas. Está diseñado para extraer fácilmente los patrones más importantes de longitud máxima sin necesidad de evaluar combinaciones de menor longitud. BP-Tree encapsula las funcionalidades comunes de BP-TransactionTree, especializado en la codificación de las transacciones originales; y de BP-PatternTree, encargado de codificar los patrones finales extraídos y en donde la determinación de patrones redundantes se facilita.

Para optimizar el tiempo de procesamiento utiliza varias técnicas de poda. Tres de estas técnicas son ampliamente utilizadas en la literatura: *Apriori property*, *Item skipping* y *Compressed Pattern Tree Subset Checking*.

Otra técnica es *Progressive Intersection Pruning (PIP)*, propuesta en este trabajo para reducir el número de operaciones de intersección.

En el desarrollo del algoritmo es importante la clasificación de los itemsets en reales o intermedios. Como cada nodo del árbol de transacciones representa un ítem, y el encadenamiento de todos los padres hasta el nodo raíz representa un itemset, entonces cada nodo distinto al nodo raíz codifica un itemset. Los itemsets reales son aquellos cuyo nodo final ha sido el final de una transacción de BD. Los itemsets previos al nodo final de una sucesión real, son considerados intermedios. En la figura 12, explicada más adelante, se sombrean los nodos que representan itemsets reales.

FP-MAXFLOW es un algoritmo que combina las estrategias de búsqueda top-down y bottom-top convenientemente. Genera los candidatos de itemsets relevantes de mayor longitud sin necesidad de evaluar combinaciones menores (top-down), pero las técnicas de poda se evalúan en itemsets de menor a mayor longitud para descartar mayor cantidad de itemsets no frecuentes (bottom-top). Los estudios comparativos demuestran que es un algoritmo competitivo.

3.1 Requisitos de los datos de entrada

Cada registro del BD debe ser un itemset, donde cada ítem representa un caso particular (instancia) de una variable. Los items siempre deben aparecer en el mismo orden. En el BD de la tabla 1, se presentan 5 items {I1, I2, I3, I5, I7}. Un registro con la información {I5, I1, I2}, no será procesado correctamente porque el algoritmo espera el formato {I1, I2, I5}.

Es posible procesar variables continuas con un procedimiento previo de discretización. En este caso, no deben aparecer dos instancias de una misma variable en un mismo registro. Si las clases están definidas por rangos numéricos, estos no deben solaparse.

3.2. Variables y estructuras de datos utilizadas

Se construyen tres estructuras de datos principales: BP-Tree, BP-TransactionTree y BP-PatternTree.

Un BP-Tree es una estructura arbórea binaria que encapsula las funcionalidades comunes de BP-TransactionTree y BP-PatternTree. A diferencia del FP-Tree empleado por FP-Growth, BP-Tree no necesita recorrer más de una vez el BD para ser construido. Es una extensión del P-Tree propuesto en [12].

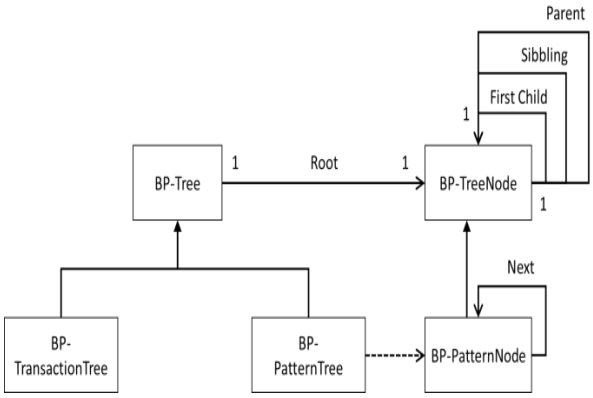

Fig. 6. Modelo de datos utilizado en FP-MAXFLOW

Dos de las principales ventajas de una estructura arbórea binaria con respecto a una general son la mayor velocidad en los recorridos verticales y la eficiencia para adicionar y eliminar nodos [18]. BP-Tree utiliza punteros en sus nodos que facilitan los recorridos verticales. No necesita ser convertido en un FP-Tree para obtener los itemsets, lo que es ventajoso con respecto al trabajo publicado en [12].

Al igual que P-Tree, ocupa mayor espacio en memoria que FP-Tree en muchos casos. El tamaño del árbol siempre dependerá de la cantidad de variables y del esparcimiento de los datos, nunca de la cantidad de registros procesados. Un árbol BP-Tree y sus nodos tienen la siguiente estructura:

BP-Tree:

- *Root:* Nodo raíz del árbol. Es un nodo virtual que no contiene información, pero servirá para iniciar los recorridos por el resto de los nodos.
- *Levels:* Niveles del árbol. Cada nivel representa un ítem, ejemplo: I5, donde se registran los nodos enlazados del ítem y su frecuencia total. Es una tabla equivalente a la tabla de frecuencias del FP-Tree.

BP-TreeNode:

- *id_item:* Identificador del ítem.

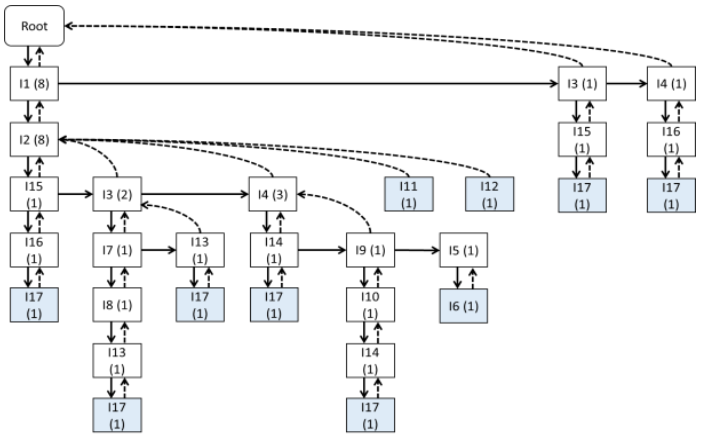

Fig. 7. Estructura BP-TransactionTree obtenida con el algoritmo 1

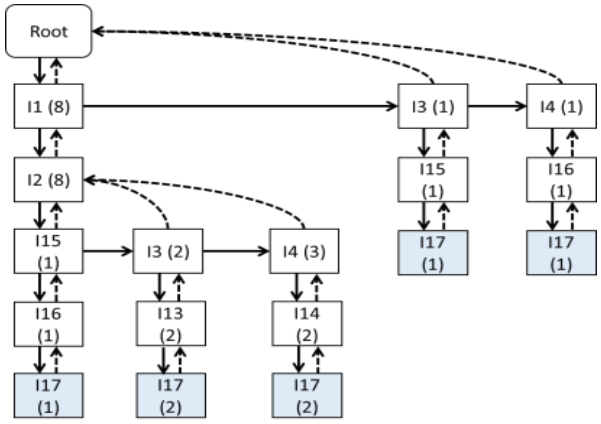

Fig. 8. Estructura BP-TransactionTree optimizada para una frecuencia mínima de 2

	I1	I2	I3	I4	I13	I14	I15	I16	I17
I1	0	6	3	2	2	2	1	0	5
I2		0	3	2	2	2	1	1	5
I3			0	1	2	0	1	0	3
I4				0	0	2	0	1	3
I13					0	0	0	0	2
I14						0	0	0	2
I15							0	1	2
I16								0	2
I17									0

Fig. 9. Matriz de transición de los items frecuentes para una frecuencia mínima de 2

- *frequency:* Frecuencia con que el ítem ha aparecido en la sucesión.
- *path_frequency:* Frecuencia que se activa solo cuando el nodo es el final de una sucesión. Sirve para discriminar entre itemsets reales e intermedios.
- *first_child:* Primer hijo del nodo.
- *sibbling:* Primer hermano del nodo. La sucesión completa de hermanos son el conjunto de hijos del nodo padre.
- *parent:* Nodo padre. Facilita los recorridos en ascenso sin procedimientos recursivos.

Un BP-TransactionTree es una extensión de BP-Tree orientado a codificar las transacciones del BD. Se construye de forma similar a un FP-Tree, pero los nodos se crean en el mismo orden de aparición de los datos originales. La estructura en niveles facilita la eliminación de nodos y la creación de proyecciones de BD. Un árbol BP-TransactionTree tiene la siguiente estructura:

BP-TransactionTree:

- *transition_matrix:* Matriz cuadrada que registra las frecuencias de transición de cada item a los posibles sucesores. Se utiliza en las operaciones de poda.
- *items_prune_map*: Array que indica si un ítem debe ser procesado o si es redundante. Se utiliza en las operaciones de poda.

Un BP-PatternTree es una extensión del BP-Tree especializado en la codificación de itemsets. Se utiliza para las proyecciones de BD y para los itemsets relevantes extraídos. Un árbol BP-PatternTree y sus nodos tienen la siguiente estructura:

BP-PatternTree:

- id_item: Identificador del ítem para el que se proyecta el BD. En el caso del árbol de itemsets relevantes este parámetro es nulo.

BP-PatternNode:

- next: Siguiente itemset dentro del nivel al que pertenece el nodo.
- id_items: Array de los items del itemset.
- map: Array binario utilizado para las operaciones entre itemsets.
- id_nofrequent_pattern: Identificador que se le asigna si el itemset que representa no es frecuente. Se les asigna también a todos los nodos descendientes. Esta variable se utiliza en las operaciones de poda.

Las variables de BP-PatternNode solo se activan para los itemsets reales, donde la variable “path_frequency” es mayor que cero. En la figura 6 se muestra el modelo de estructuras de datos utilizado por el algoritmo FP-MAXFLOW.

En la figura 6 se observa que la relación entre BP-PatternTree y BP-PatternNode es de dependencia y no de asociación. Se debe a que las especializaciones de BP-Tree heredan su misma estructura de nodos.

La especialización BP-PatternNode se utiliza localmente en las operaciones de minería, de ahí la relación de dependencia.

3.3. Técnicas de poda utilizadas

- *Apriori property:* Demuestra que todos los subsets propios distintos de nulo de un itemset frecuente son frecuentes también [4]. Del mismo modo se conoce que si un itemset no es frecuente, todos sus supersets propios tampoco lo serán.
- *Item skipping:* Cada ítem tiene una frecuencia total, registrada en el nivel correspondiente del árbol de transacciones. Cada proyección de BD tiene un ítem prefijo, y una tabla de frecuencias que indica la frecuencia de cada ítem anterior con respecto al prefijo. Si un ítem tiene la misma frecuencia total que en una o varias proyecciones de BD, solo se procesará en la de mayor prefijo. Como se explica en [19], cualquier proyección anterior a la mayor para estos items genera itemsets redundantes.
- *Compressed Pattern Tree Subset Checking*: Dado un árbol T, si un itemset A puede ser embebido en una ramificación $R \in T$, que contiene un itemset B, entonces se cumplen alguna de las siguientes propiedades: 1. A y B tienen la misma frecuencia, 2. La longitud de A es menor o igual a la longitud de B, 3. Todos los items de A están en B.

  Como se explica en [19], gracias a estas propiedades se pueden crear modelos que agilicen las comprobaciones de subsets propios.

- Progressive Intersection Pruning (PIP): Si la intersección entre dos itemsets A y B no es frecuente, se cumple que las intersecciones entre A y los supersets propios de B, entre B y los supersets propios de A, y entre los supersets propios de A y B, tampoco son frecuentes.

3.4. Extracción de itemsets

FP-MAXFLOW se compone de 4 etapas:

1. Construir un BP-TransactionTree con las transacciones originales.
2. Optimizar el BP-TransactionTree, donde se incluyen las podas del tipo “Item skipping”.
3. Construir proyecciones de BD y realizar el proceso de minería, donde se va construyendo un árbol del tipo BP-PatternTree de itemsets relevantes.
4. Podar árbol de itemsets relevantes según la técnica “Compressed Pattern Tree Subset Checking” y extraer los itemsets finales.

Todas estas etapas serán explicadas con el BD de ejemplo de la tabla 2 y una frecuencia mínima igual a 2.

3.4.1. Crear el árbol BP-Transaction Tree

El procedimiento de construcción del BP-TransactionTree es muy parecido al utilizado por FP-Growth para el FP-Tree, pero los nodos se van creando en el mismo orden en que aparecen las clases en las transacciones.

En el algoritmo 1, para cada nueva transacción se intenta buscar una transacción anterior con los mismos prefijos para incrementar la frecuencia de aparición de los nodos, y solo en el caso de no encontrar ningún resultado se crea una nueva ramificación. Para el ejemplo de la tabla 2 se obtiene la estructura BP-TransactionTree mostrada en la figura 7. Las flechas continuas indicadas hacia abajo representan el primer hijo de un nodo. Las flechas continuas indicadas hacia la derecha representan el primer hermano. Se observa que el nodo raíz tiene tres hijos {I1(8), I3(1) y I4(1)}. La frecuencia de cada nodo se representa entre paréntesis. Las flechas con líneas discontinuas representan el nodo padre.

Tabla 2. BD de ejemplo 2

ID	Items
1	I1 I2 I15 I16 I17
2	I1 I2 I3 I7 I8 I13 I17
3	I1 I2 I4 I14 I17
4	I1 I2 I3 I13 I17
5	I1 I2 I4 I9 I10 I14 I17
6	I3 I15 I17
7	I4 I16 I17
8	I1 I2 I4 I5 I6
9	I1 I2 I11
10	I1 I2 I12

Las otras variables quedan como sigue: levels_frequency = {I1(8), I2(8), I3(3), I4(4), I5(1), I6(1), I7(1), I8(1), I9(1), I10(1), I11(1), I12(1), I13(2), I14(2), I15(2) , I16(2) , I17(7)}

Por razones de espacio la matriz de transición se representa en la figura 9 solo con los items frecuentes. Esta matriz se calcula en el mismo proceso de adición de transacciones al árbol de la figura 7, y las columnas de los items no frecuentes pueden ser eliminadas o ignoradas. Los nodos sombreados representan los finales de transacción. Gracias a este tipo de estructura los recorridos verticales en ascenso se pueden realizar eficientemente sin procedimientos recursivos.

3.4.2. Optimizar el árbol BP-TransactionTree

Consiste en procesar solo los nodos que contengan clases frecuentes. La eliminación de los nodos no frecuentes se fundamenta en la propiedad Apriori. Para una frecuencia mínima igual a 2 se eliminan las clases: {I5(1), I6(1), I7(1), I8(1), I9(1), I10(1), I11(1), I12(1)}. En la figura 8 se muestra el árbol resultante. En el proceso de optimización también se definen los items redundantes según la técnica de poda “ítem skipping”. Atendiendo a las frecuencias totales y a la matriz de transición de la figura 9, se observa que los items {I3(3) I13(2), I14(2), I15(2), I16(2)} tienen la misma frecuencia total que en la proyección de {I17}, por lo que no es necesario procesar las proyecciones de BD para estos items.

Cada posición del vector de poda corresponde al ítem del mismo orden. En caso de tener

Algoritmo 1: Construir BP-TransactionTree
Entrada: Transacciones, T = {T1, T2, Tn}.
Salida: Estructura BP-TransactionTree.

```
01  BPTransactionTree tree =
    BPTransactionTree()

02  for each Ti ∈ T:
03      BPTreeNode parent = tree.root

04      for each Item ∈ Ti:
05          Increment
06          tree.levels[Item.id].frequency
            BPTreeNode child=
            parent.Find(Item.id)

07          if (child) then
08              child.IncrementFrequency(1)
09          else
10              parent.AddChild(Item.id)
11          end If

12          UpdateTransitionMatrix(Item, Ti);
13          parent = child
14      end For
15  end for

16  return tree
```

asignado un valor mayor que cero, indica que el ítem es redundante y la proyección en la que debe ser procesado es el valor asignado. Para la optimización del árbol de la figura 7 se obtiene el siguiente vector de poda:

I1	I2	I3	I4	I13	I14	I15	I16	I17
0	0	17	0	17	17	17	17	0

El algoritmo 2 muestra el proceso de optimización del BP-TransactionTree.

3.4.3. Extraer itemsets

El árbol obtenido en la etapa anterior se procesa desde los nodos hoja hasta el nodo raíz, en sentido inverso al orden original de las clases. En este caso el orden de procesamiento sería {I17, I4, I2}. La proyección {I1} es nula porque no existe ninguna clase que la anteceda.

Algoritmo 2: Optimizar BP-TransactionTree
Entrada: Estructura BP-TransactionTree, minimum support (min_sup).
Salida: BP-TransactionTree optimizado.

```
01  for each Level_i ∈ BPTransactionTree.levels:

02      if (Level_i.frequency < min_sup) then
03          DeleteLevelNodes(Level_i)
04      else
05          UpdateLevelPruneMap(Level_i)
06      end if

07  end for

08  return BPTransactionTree
```

A los efectos de este artículo, solo interesa obtener itemsets con dos o más items. Los itemsets de longitud unitaria pueden obtenerse directamente en la primera etapa del algoritmo.

Para cada clase se construye una proyección de BD mediante una estructura BP-PatternTree.

Utilizando la matriz de transición, se agregan solo los items que son frecuentes para la proyección. Para el caso de {I17} se obtiene el árbol de la figura 10.

La estructura BP-PatternTree se construye igual al BP-TransactionTree. Se diferencia en que a los nodos finales de cada transacción se le suma la frecuencia a la variable “path_frequency” y se le asignan las variables “id_items” y “map”, que codifican el itemset que contienen.

En la figura 10 se observa un nuevo tipo de enlace marcado en color azul. Estos enlaces pertenecen a la especialización de los nodos en BP-PatternNode. Se orientan desde un nodo final de transacción hacia el siguiente nodo final de transacción del mismo nivel. A diferencia del FP-Tree, estos enlaces no se crean para cada nodo a partir de la tabla de frecuencias, sino que en cada nivel solo existirán para las transacciones reales. Para la proyección de BD de la figura 10, existen dos transacciones finalizadas en {I16}. Las transacciones finalizadas en {I13, I14, I15}, no tienen más de un nodo.

El proceso de minería consta de 2 etapas.

Etapa 1. Calcular frecuencias totales

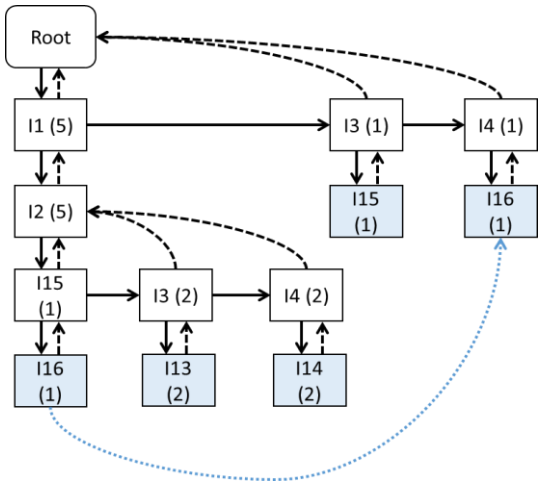

Fig. 10. BP-PatternTree con la proyección de BD para la clase {I17}

Los itemsets del árbol de proyección pueden estar incluidos dentro de otros itemsets, por lo que las frecuencias registradas en los nodos pueden no ser definitivas. Por ejemplo, si se tienen dos itemsets A: {I1, I3, I5 = 2} y B: {I1, I5 = 1}, la frecuencia total de B es 3, porque está incluido dentro de A. Gracias a la estructura en niveles este proceso se puede realizar eficientemente. La navegación entre los itemsets de un mismo nivel es fácil debido a los enlaces en cadena. Se establecen las siguientes reglas:

- Un itemset de un nivel L solo podrá estar incluido dentro de otro itemset del mismo nivel o superior, nunca inferior.
- Solo se realizarán chequeos de subsets propios contra los itemsets de transacciones reales, nunca contra itemsets intermedios. Estos itemsets pueden obtenerse en cadena a partir de los nodos donde la variable "path_frequency" es mayor que cero.
- La frecuencia de un itemset A se incrementará en la cantidad "path_ frequency" de un superset propio B si y solo si A es subset propio de B y no pertenece a la misma ramificación. Por ejemplo, dado A: {I1, I2 = 4} y B: {I1, I2, I3 = 2}, se cumple que A es subset propio de B, pero en la frecuencia asignada ya se consideró la transacción de B porque A está en la misma ramificación {I1, I2}.

Además de estas propiedades, FP-MAXFLOW agiliza aún más estos chequeos gracias a la propiedad Apriori. En vez de ir directamente a un itemset real, el algoritmo intenta descubrir los itemsets intermedios no frecuentes de menor longitud (comenzando por los hijos del nodo raíz). Si uno de estos itemsets intermedios no es frecuente, se ignora el resto de su descendencia. En caso de ser frecuente, se incrementa su frecuencia si es un itemset real y en cualquier caso se evalúan los itemsets mayores inmediatos. Los itemsets frecuentes luego de este proceso se registran como relevantes. Para el árbol de la figura 10 se obtiene:

I3 ^ I15 = 1	(P1)
I4 ^ I16 = 1	(P2)
I1 ^ I2 ^ I15 ^ I16 = 1	(P3)
I1 ^ I2 ^ I3 ^ I13 = 2	**(P4)**
I1 ^ I2 ^ I4 ^ I14 = 2	**(P5)**

Etapa 2. Extraer itemsets intermedios

Los itemsets no frecuentes pueden incluir otros que sí lo son. Una técnica eficiente para extraer los de mayor longitud posible es mediante la operación de intersección. En el caso de FP-MAXFLOW solo se consideran las intersecciones de los itemsets no frecuentes con el resto. Las intersecciones entre itemsets frecuentes no son necesarias porque cualquier resultado será un itemset no relevante, o hubiera sido obtenido en la etapa 1. En la tabla 3 se muestran las posibles intersecciones para los itemsets P1, P2 y P3 del listado anterior.

Las intersecciones frecuentes son itemsets relevantes. Se fundamenta en que al eliminar las porciones que hacen a un itemset no frecuente, los itemsets resultantes son las intersecciones y son itemsets que aparecen de forma independiente en al menos una transacción. En este proceso deben aplicarse técnicas de poda porque las operaciones de intersección son costosas.

La más sencilla es evitar calcular una misma intersección varias veces. Para evitarlo FP-MAXFLOW activa una bandera para el itemset que ya fue procesado. Como cada itemset no frecuente es comparado con el resto, los itemsets marcados pueden ignorarse en iteraciones futuras.

Tabla 3. Intersecciones para la proyección de BD {I17}

ID	Intersecciones
P1	{ I3 } { I15 }
P2	{ I4 } { I16 }
P3	{ I1, I2 }

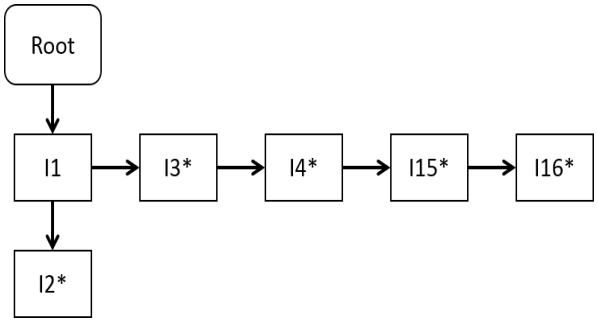

Fig. 11. Intersecciones para la proyección de BD {I17}

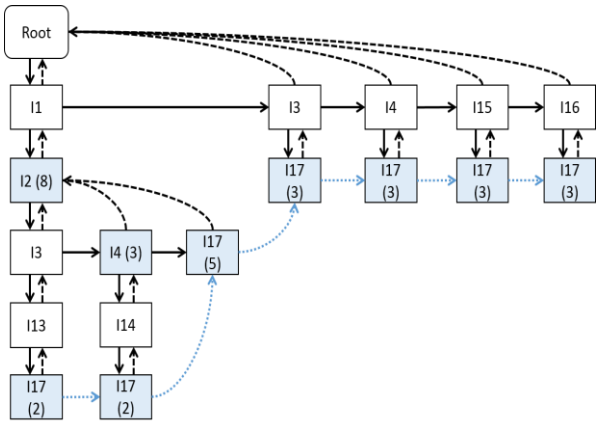

Fig. 12. Estructura BP-PatternTree con los itemsets frecuentes relevantes de las proyecciones procesadas

De no haber realizado esta poda las intersecciones para P3 serían {I1, I2, I15, I16}. Otra técnica de poda aplicada es PIP, explicada en 3.3. Para cada intersección debe calcularse su frecuencia total. Para administrar la repetición de los nuevos itemsets FP-MAXFLOW utiliza un árbol del tipo BP-Tree, en donde registra cada nueva intersección:

I1 ^ I2 ^ I3 ^ I13 ^ I17 = 2	(P1)
I1 ^ I2 ^ I4 ^ I14 ^ I17 = 2	(P2)
I1 ^ I2 ^ I17 = 5	(P3)
I3 ^ I17 = 3	(P4)
I4 ^ I17 = 3	(P5)
I15 ^ I17 = 2	(P6)
I16 ^ I17 = 2	(P7)

Es conveniente utilizar un árbol de intersecciones independiente para cada proyección de BD porque su magnitud es menor a un árbol empleado para todas las proyecciones. Las frecuencias de los itemsets relevantes de la figura 11 son:

I1 ^ I2 = 5	(P1)
I3 = 3	(P2)
I4 = 3	(P3)
I15 = 2	(P4)
I16 = 2	(P5)

Finalmente, los itemsets frecuentes relevantes para la clase {I17} son los itemsets relevantes se almacenan en un árbol de tipo BP-PatternTree.

El proceso de minería se realiza de igual forma para el resto de las clases no redundantes: {I2, I4}. El algoritmo 3 resume el proceso de minería explicado.

3.4.4. Procesar árbol de itemsets

En cada proyección de BD se registran los itemsets frecuentes relevantes obtenidos. Una forma eficiente de almacenar estos itemsets en memoria es mediante una estructura arbórea del tipo BP-PatternTree. Se obtiene el árbol de itemsets de la figura 12. Luego de procesar todas las proyecciones de BD no redundantes y no nulas, para cada itemset del árbol de itemsets relevantes se suman las frecuencias de los supersets propios inmediatos, siguiendo las ramificaciones del árbol.

En la figura 12, el itemset A: {I1, I2, I3, I13, I17=2} es superset propio inmediato del itemset B: {I1, I2=8}. El itemset C: {I1, I2, I4, I14, I17=2} es superset propio de A, pero no es inmediato,

Algoritmo 3: Extraer itemsets de una proyección de BD.
Entrada: Estructura BP-PatternTree para una proyección de BD (db_projection), minimum support (min_support), estructura BP-PatternTree para registrar los itemsets relevantes (patterns_tree).
Salida: patterns_tree actualizado.

```
01  BPTreeNode Root = db_projection.Root
02  for each Node_i ∈ Root.childs
03      UpdateInheritFrequency(Node_i,
04      patterns_tree)
05  end for

06  BPTree intersections = new  BPTree()
    CalculateIntersections(intersections)

07  Root = intersections.Root
08  for each Node_i ∈ Root.childs
09      UpdateInheritFrequency(Node_i,
        patterns_tree)
10  end for

11  delete intersections
12  return patterns_tree
```

Función: UpdateInheritFrequency
Entrada: Estructura BP-TreeNode (Node), estructura BP-PatternTree (patterns_tree)
Salida: Parámetros actualizados.

```
01  CalculateRealFrequency(Node)

02  if (Node.frequency >= min_support) then
03      if (Node.path_frequency > 0) then
04          patterns_tree .RegisterPattern(Node)
05      end if

06      for each Node_i ∈ Node.childs
07          UpdateInheritFrequency(
                Node_i, patterns_tree)
08      end for
09  end if
```

porque entre A y C se encuentra al itemset D: {I1, I2, I4=3}.

Los supersets propios inmediatos de un itemset A, que se encuentran en las ramificaciones sucesoras del nodo N_A en donde se almacena A, tendrán un prefijo en común igual a A.

Algoritmo 4: FP-MAXFLOW.
Entrada: Transacciones (T), mínimum support (min_support)
Salida: Itemsets relevantes.

```
01  BPTransactionTree tran = new
        BPTransactionTree()

02  for each T_i ∈ T
03      tran.Add(T_i)
04  end for

05  tran.Optimize(min_support)
06  BPPatternTree patterns = new
    BPPatternTree()

07  for each Item_i ∈ (tran.items_prune_map ==
    0)
08
        BPPatternTree projection = new
            BPPatternTree(Item_i)
09
10      for each Node_i ∈
11      transactions.levels[Item_i]
            projection.Add(
12          Node_i.GetSequence() )
13      end for
14
        projection.MinePatterns(patterns)
15      delete projection
16  end for

17  DeleteRedundancies(patterns)
18  Save(patterns)

    delete patterns
    delete transactions
```

Por lo tanto, si un itemset A tiene la misma frecuencia que la suma de las frecuencias de sus supersets propios inmediatos y sucesores en el árbol de patrones, entonces A es redundante y puede descartarse.

Dentro de un mismo nivel no hay redundancias.En el caso de la figura 12, todos los itemsets son relevantes y no redundantes.

El algoritmo 4 presenta el procedimiento FP-MAXFLOW completo.

ISSN 2007-9737

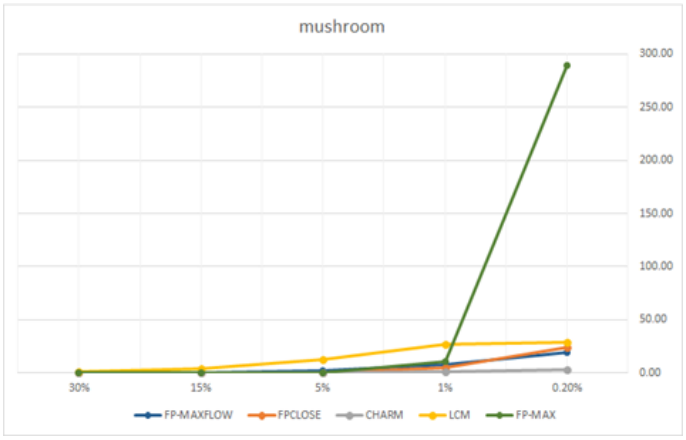

Fig. 13. Comparación de algoritmos para la BD mushroom

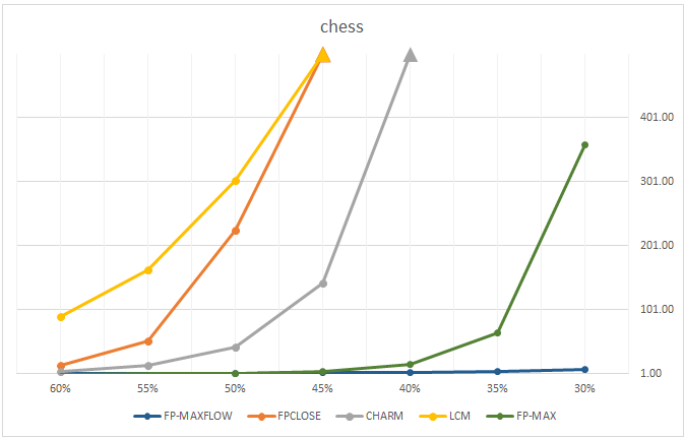

Fig. 14. Comparación de algoritmos para la BD chess

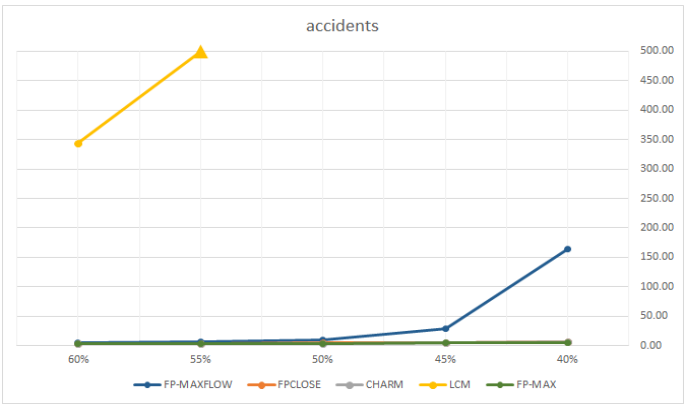

Fig. 15. Comparación de algoritmos para la BD accidents

4. Estudios comparativos

Se seleccionaron 5 BD publicadas en el repositorio FIMI (Frequent Itemset Mining Dataset Repository) para la minería de itemsets, disponible en [20]. De estas, tres son reales (mushroom, chess, accidents) y dos son artificiales (T10I4D100K, T40I10D100K).

Todas se pueden encontrar en el repositorio internacional "UCI Machine Learning Repository". El algoritmo es comparado con los algoritmos FPClose, Charm, LCM y FPMax. Los tres primeros extraen itemsets cerrados, mientras que FPMax extrae itemsets maximales. Se decidió comparar con estos tipos de algoritmos y no con los que extraen todos los itemsets frecuentes porque las técnicas de poda que emplean se asemejan más al procedimiento FP-MAXFLOW. Los tiempos de ejecución se registran en segundos de CPU y han sido truncados a un valor máximo de 500 segundos. Los algoritmos que han demorado más de este tiempo se representan en los gráficos mediante una saeta hacia arriba. Las pruebas se realizaron sobre un Microprocesador Intel Core I3, 2.30 GHz, para una memoria RAM disponible de 2GB. Los resultados aquí publicados podrían no repetirse.

BD mushroom

Transacciones (trans.)	8124
Clases	119
Longitud trans. mínima	23
Longitud trans. máxima	23
Longitud trans. promedio	23
Frecuencia mínima	4
Frecuencia máxima	8124

BD chess

Transacciones (trans.)	3196
Clases	75
Longitud trans. mínima	37
Longitud trans. máxima	37
Longitud trans. promedio	37
Frecuencia mínima	1
Frecuencia máxima	3195

BD accidents

Transacciones (trans.)	340183
Clases	468
Longitud trans. mínima	18
Longitud trans. máxima	51
Longitud trans. promedio	33
Frecuencia mínima	1
Frecuencia máxima	340151

BD T10I4D100K

Transacciones (trans.)	100000
Clases	1000
Longitud trans. mínima	1
Longitud trans. máxima	29
Longitud trans. promedio	10
Frecuencia mínima	1
Frecuencia máxima	7828

BD T40I10D100K

Transacciones (trans.)	100000
Clases	1000
Longitud trans. mínima	4
Longitud trans. máxima	77
Longitud trans. promedio	39
Frecuencia mínima	5
Frecuencia máxima	28738

El algoritmo FPClose [21] fue propuesto en la competición FIMI 2004, siendo clasificado como uno de los mejores. En la implementación de este trabajo no se consideraron las optimizaciones propuestas por FP-Growth*. El algoritmo Charm [22] [26] es uno de los algoritmos más eficientes para la minería de itemsets cerrados. Se empleó la variación dCharm, que es más eficiente y aplica la técnica "diffsets", mientras que el original aplica la técnica "tidsets". El algoritmo LCM [23] fue el ganador de la competición FIMI 2004. Es rápido en BD esparcidas, pero en BD densas su rendimiento

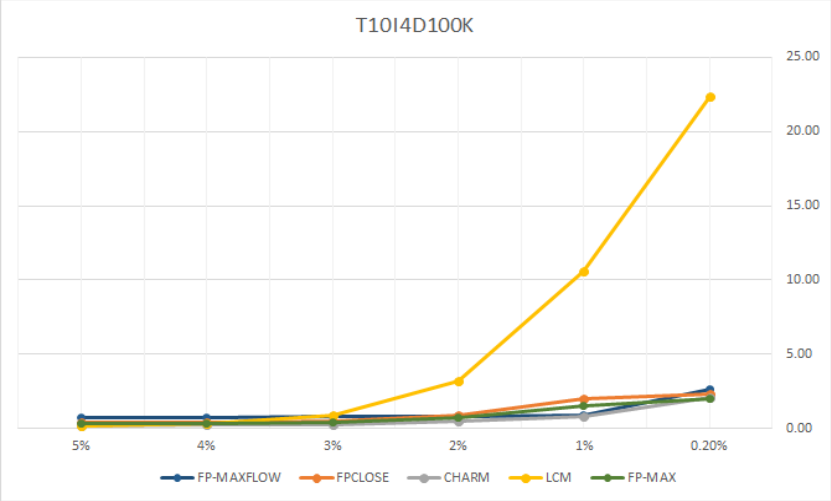

Fig. 16. Comparación de algoritmos para la BD T10I4D100K

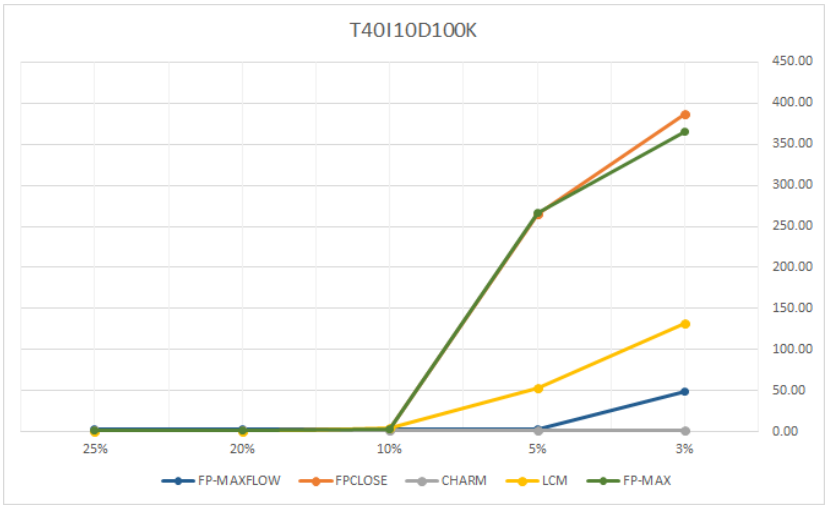

Fig. 17. Comparación de algoritmos para la BD T40I10D100K

decae considerablemente. El algoritmo FP-Max [16] es uno de los más eficientes en la extracción de itemsets maximales. Su rendimiento no es bueno en BD con transacciones largas e itemsets frecuentes largos. Es notable como la densidad de las BDs puede afectar o mejorar significativamente el rendimiento de los algoritmos. Por esta razón, este estudio comparativo se ha orientado al procesamiento de BDs extraidas de repositorios reconocidos por la comunidad de reconocimiento de patrones y no en análisis de costo y espacio.

Ofrecer una métrica de costo no ha sido prioridad en este trabajo debido a que el tiempo de ejecución no dependerá en ningún caso de la cantidad N de registros, sino de las relaciones entre los items. El comportamiento de estas relaciones es muy complicado de cuantificar debido a que en general se desconocen y cualquier variación podría tener una influencia significativa. Los niveles de consumo de memoria también dependen del tipo de implementación y del lenguaje de programación utilizado.

Tabla 4. Estudio comparativo entre FP-MAXFLOW, FPCLOSE, Charm, LCM y FP-MAX

mushroom	**0.02%**	**1%**	**5%**	**15%**	**30%**		
FP-MAXFLOW	19.634	7.677	1.983	0.219	0.069		
FPCLOSE	24.370	4.533	0.538	0.175	0.150		
CHARM	3.142	1.058	0.323	0.110	0.068		
LCM	29.147	26.441	12.168	3.500	0.723		
FP-MAX	289.179	10.708	0.479	0.126	0.120		
chess	**30%**	**35%**	**40%**	**45%**	**50%**	**55%**	**60%**
FP-MAXFLOW	7.776	4.961	3.321	2.127	1.366	0.705	0.442
FPCLOSE				> 500.000	224.486	52.691	13.682
CHARM			> 500.000	143.277	42.741	13.315	4.067
LCM				> 500.000	302.800	163.261	89.621
FP-MAX	358.645	65.372	15.309	4.138	1.371	0.531	0.281
accidents	**40%**	**45%**	**50%**	**55%**	**60%**		
FP-MAXFLOW	164.408	28.010	10.185	5.683	4.890		
FPCLOSE	6.616	5.060	4.128	3.840	3.683		
CHARM	6.420	4.440	3.178	3.061	2.844		
LCM				> 500.000	343.989		
FP-MAX	5.425	4.827	4.081	3.667	3.648		
T10I4D100K	**0.02%**	**1%**	**2%**	**3%**	**4%**	**5%**	
FP-MAXFLOW	2.593	0.898	0.829	0.768	0.723	0.687	
FPCLOSE	2.322	1.982	0.916	0.479	0.381	0.375	
CHARM	2.044	0.814	0.467	0.272	0.212	0.191	
LCM	22.406	10.599	3.193	0.850	0.334	0.164	
FP-MAX	2.022	1.517	0.742	0.423	0.331	0.318	
T40I10D100K	**3%**	**5%**	**10%**	**20%**	**25%**		
FP-MAXFLOW	48.484	3.510	2.805	2.630	2.614		
FPCLOSE	387.473	265.362	2.942	0.998	0.981		
CHARM	1.641	1.529	1.483	1.462	1.455		
LCM	131.950	53.384	4.366	0.434	0.180		
FP-MAX	365.341	266.487	3.220	1.002	0.983		

El consumo de memoria de FP-MAXFLOW podría ser diferente si se ejecuta en plataformas Windows o Linux, o si se implementa en lenguajes como C++ o Java. La experiencia del programador también influye en este elemento, pues los algoritmos estudiados en el estado del arte, y el propio FP-MAXFLOW, proponen el empleo de estructuras de datos complejas, pero no advierten sobre un momento adecuado para la liberación de memoria. En lenguajes como C o C++ hay que ser más específico para liberar memoria, mientras que en lenguajes como Java o Python este concepto podría no ser determinante por la existencia de recursos para liberar automáticamente la memoria no utilizada.

Existen en los repositorios mencionados muchas más BDs para extraer itemsets frecuentes. Sin embargo, las BDs seleccionadas están entre las más populares en el estado del arte para comprobar algoritmos de extracción de itemsets frecuentes, ya sea todo el conjunto, maximales o cerrados. Algunas publicaciones importantes que también utilizan estas BDs para evaluar algoritmos son [16], [21], [24], [25], [26] y [27]. Otras dos BDs muy utilizadas, disponibles en los repositorios FIMI y UCI, y que podrían haberse utilizado en este estudio comparativo son: “connect” y “pumsb”.

FP-MAXFLOW demuestra ser un algoritmo competitivo. El algoritmo es eficiente en BDs donde los itemsets frecuentes son de tamaño similar al de las transacciones. En BDs como “accidents”, donde las transacciones son largas y los itemsets frecuentes son cortos, el rendimiento decae por las operaciones de intersección. No obstante, aún en estos casos no hubo diferencias significativas.

5. Conclusiones

El algoritmo FP-MAXFLOW extrae los itemsets relevantes que mejor podrían explicar las correlaciones entre las variables de una BD. Demuestra ser un algoritmo competitivo con respecto a otros algoritmos del estado del arte.

Obtener solo el conjunto de itemsets relevantes agiliza las tareas de análisis y optimiza el espacio de memoria y de disco necesarios. En este sentido FP-MAXFLOW aporta conceptos y estructuras de datos que pueden tener un impacto significativo en muchos dominios.

FP-MAXFLOW ha sido aplicado en el análisis del proceso de producción de alcohol en destilerías, como parte de un sistema para la gestión de la contabilidad alcoholera y la extracción de conocimientos sobre el proceso productivo, denominado SISALCO [28]. Sus contribuciones más relevantes fueron la validación de datos y la determinación de relaciones entre grupos de variables de causa y efecto que han constituido nuevos conocimientos, patrones no deseados, o se han utilizado para comprobar que los indicadores calculados son consistentes con los conocimientos de los expertos. FP-MAXFLOW ha significado para este dominio un complemento importante para controlar la producción de alcohol mediante métodos matemático-computacionales.
Como parte de la experiencia obtenida en las aplicaciones reales de FP-MAXFLOW, se ha comprobado que la mayor utilidad de los patrones relevantes propuestos está en los dominios que necesitan analizar masas de datos con relaciones complejas entre variables, y donde se requieren respuestas con información concreta para el apoyo a la toma de decisiones rápidas, que podrían ser inmediatas. Para realizar estudios más detallados, donde se dispone de más tiempo, otros algoritmos para extraer todo el conjunto de itemsets frecuentes podrían ser más adecuados.

FP-MAXFLOW es un algoritmo fácil de implementar y tiene una estructura que permite paralelizar algunas de sus etapas. Sus estrategias de búsqueda y poda pueden variarse para obtener itemsets maximales o cerrados, con lo cual aumenta su valor práctico.

Referencias

1. **Fernandez, G. (2010).** *Statistical Data Mining Using SAS Applications*. Taylor and Francis Group, LLC.
2. **Rajaraman, A. & Jeffrey, D. (2011).** *Mining of Massive Datasets,* Vol. 184, Stanford University.
3. **Han, J. & Kamber, K. (2006).** *Data Mining Concepts and Techniques.* University of Illinois at Urbana-Champaign, Basic Concepts and a Road Map.
4. **Han, J. & Kamber, K. (2006).** The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation. In: *Data Mining. Concepts and*

Techniques, University of Illinois at Urbana-Champaign.

5. **Rajaraman, A. & Jeffrey, D. (2011).** The Algorithm of Park, Chen, and Yu. In: *Mining of Massive Datasets,* Stanford University.
6. **Al-Maolegi, M. & Arkok, B. (2014).** An improved Apriori Algorithm for Association Rules. *International Journal on Natural Language Computing*, Vol. 3, No. 1.
7. **Rajaraman, A. & Jeffrey, D. (2011).** The SON Algorithm and Map-Reduce. In: *Mining of Massive Datasets.* Stanford University.
8. **Rajaraman, A. & Jeffrey, D. (2011).** Torvonin's Algorithm. In: *Mining of Massive Datasets.* Stanford University.
9. **Han, J., Jian, P., & Yin, Y. (2000).** *School of Computing Science Simon Fraser University*.
10. **Nageswara, R.G. & Kumar, G.S. (2011).** Mining Frequent Itemsets without Candidate Generation using FP-Trees. *International Journal of Computer Science and Information Technologies*, Vol. 2, No. 6, pp. 2677–2685.
11. **Saraee, M. (2000).** *One Scan is Enough: Optimizing Association Rules Mining.* School of Computing, Science and Eng., University of Sanford.
12. **Huang, H., Wu, X., & Relue, R. (2002).** *Association Analysis with One Scan of Databases.* Dept. of Math and Computer Science, Colorado School of Mines Golden, Department of Computer Science, University of Vermont Burlington, pp. 629–632. DOI: 10.1109/ICDM.2002.1184015.
13. **Bhatnagar, D., Adlakha, N., & Saxena, A.S. (2011).** Mining Frequent Itemsets without Candidate Generation using Optical Neural Network. *IJCA, Special Issue on "Artificial Intelligence Techniques - Novel Approaches & Practical Applications"*, pp. 14–18.
14. **Bayardo, R.J. (2000).** Efficiently Mining Long Patterns from Databases. *Proceeding of Special Interest Group on Management of Data*, pp. 85–93.
15. **Burdick, D., Calimlim, M., & Gehrke, J. (2001).** Mafia: A Maximal Frequent Itemset Algorithm for Transactional Databases. *Proceedings of the 17th International Conference on Data Engineering*, pp. 443–452. DOI: 10.1109/ICDE.2001.914857.
16. **Grahne, G. & Zhu, J. (2003).** High performance mining of maximal frequent itemsets. *6th International Workshop on High Performance Data Mining*.
17. **Wang, S., Zhan, Y., & Wang, L. (2009).** An Algorithm for Mining Maximum Frequent Itemsets Using Data-sets Condensing and Intersection Pruning. *Proceedings of the Second Symposium International Computer Science and Computational Technology*, pp.12-15.
18. **Skiena, S. (2009).** *The Algorithm Design Manual.* Springer Science & Business Media, pp. 77.
19. **Han, J. & Kamber, K. (2006).** *Data Mining Concepts and Techniques.* University of Illinois at Urbana-Champaign, Mining Closed Frequent Itemsets, pp. 248.
20. **Frequent Itemset Mining Dataset Repository (2016).** http://fimi.ua.ac.be/data/
21. **Grahne, G. & Zhu, J. (2005).** Fast Algorithms for Frequent Itemset Mining Using Fp-Trees. *IEEE Knowledge and Data Engineering, Transactions*, Vol. 17, No. 10, pp. 1347–1362. DOI: 10.1109/TKDE.2005.166.
22. **Zaki, M.J. & Gouda, K. (2001).** *Fast Vertical Mining Using Diffsets.* Technical Report 01-1, Computer Science Dept., Rensselaer Polytechnic Institute, pp. 326–335. DOI: 10.1145/956750.956788.
23. **Uno, T., Kiyomi, M., & Arimura, H. (2004).** LCM v2: Efficient Mining Algorithms for Frequent Closed Maximal Itemsets. *Proc. IEEE (ICDM) Workshop on Frequent Itemset Mining Implementations*, pp. 1–8.
24. **Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999).** *Discovering Frequent Closed Itemsets for Association Rules*. Laboratoire d'Informatique (LIMOS), pp. 398–416. DOI: 10.1007/3-540-49257-7_25.
25. **Zaki, M. (2003).** Mining Closed & Maximal Frequent Itemsets. *Computer Science Department. Rensselaer Polytechnic Institute*.
26. **Zaki, M.J. & Hsiao, C.J. (1999).** CHARM: An Efficient Algorithm for Closed Association Rule Mining. *Computer Science Department. Rensselaer Polytechnic Institute*, pp. 1–20.
27. **Moonesinghe, H.D.K., Fodeh, S., & Tan, P.N. (2006).** Frequent Closed Itemset Mining Using Prefix Graphs with an Efficient Flow-Based Pruning Strategy. *Department of Computer Science & Engineering.* DOI: 10.1109/ICDM.2006.74.
28. **Díaz, A. & Ribas, M. (2015).** *Sisalco: nueva solución para la contabilidad alcoholera en destilerías cubanas ICIDCA sobre los derivados de la caña de azúcar.* Vol. 49, No. 1, pp. 32–39.

Article received on 09/12/2016; accepted on 14/11/2017.
Corresponding author Arnaldo Díaz-Molina.