(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 190 747**

(21) Número de solicitud: 200102582

(51) Int. Cl.[7]: **G06F 17/30**

(12) PATENTE DE INVENCIÓN B1

(22) Fecha de presentación: **22.11.2001**

(43) Fecha de publicación de la solicitud: **01.08.2003**

Fecha de la concesión: **14.05.2004**

(45) Fecha de anuncio de la concesión: **16.06.2004**

(45) Fecha de publicación del folleto de la patente: **16.06.2004**

(73) Titular/es: **Universidad de Málaga**
Plaza de El Ejido, s/n
29071 Málaga, ES

(72) Inventor/es: **Rodríguez Moreno, Andrés y**
Trelles Salazar, Oswaldo

(74) Agente: **No consta**

(54) Título: **Procedimiento para descubrimiento de conjuntos frecuentes en bases de datos.**

(57) Resumen:
Procedimiento para descubrimiento de conjuntos frecuentes en bases de datos.
Se presenta un método para descubrir patrones de aparición frecuentes en una base de datos. A partir de esos patrones se pueden deducir reglas de asociación en la aparición de elementos en la base de datos. La idea básica consiste en encontrar los conjuntos de elementos con frecuencia de aparición por encima de un umbral prefijado, usando como generador del espacio de búsqueda el contenido de la propia base de datos y usando la información que se desprende de la propia búsqueda para reducir al máximo la exploración de posibilidades que no formarán parte de la solución. Con este método se puede realizar la exploración de bases de datos complejas con gran nivel de detalle localizando reglas asociativas con bajo soporte (baja frecuencia), pero de gran fiabilidad y significación en entornos de aplicación como las bases de datos génicas o biológicas en general.

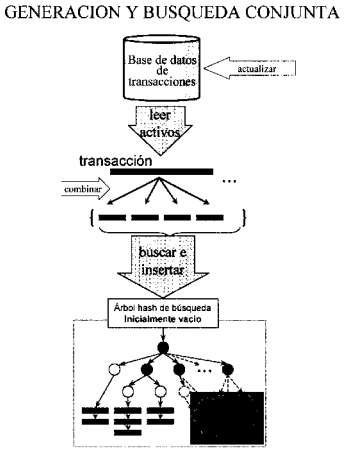

FIGURA 6

ES 2 190 747 B1

Aviso: Se puede realizar consulta prevista por el art. 37.3.8 LP.

Venta de fascículos: Oficina Española de Patentes y Marcas. C/Panamá, 1 – 28036 Madrid

DESCRIPCION

Procedimiento para descubrimiento de conjuntos frecuentes en bases de datos.

La presente invención se enmarca dentro del área conocida como extracción automática de conocimiento, que consiste en el análisis inteligente de la información residente en bases de datos para extraer el conocimiento subyacente en las relaciones y patrones de repetición que existen en la base de datos. En este caso la invención aporta un nuevo método para descubrir patrones de aparición frecuentes y las posteriores reglas de asociación que se pueden deducir de esos patrones.

Antecedentes de la invención

La tarea de descubrimiento de conocimiento tiene como meta general la extracción y abstracción de cualquier tipo de patrón, regularidad, estructuración o asociación de los datos analizados. Uno de los resultados más útiles de una de las tareas de extracción de patrones conocida como "análisis de vínculos entre datos" (*link análisis*) [Fayyad, U., Piatestky-Shapiro, G. and Smyth, P. (1996). "From data mining to knowledge discovery: An overview". *Advances in knowledge discovery and data mining*. Eds. AAAI/MIT Press, Cambridge], aparece en forma de reglas asociativas (*association rules*). Estas reglas hacen explícita la relación entre un conjunto de antecedentes y sus consecuencias asociadas, proporcionando además información adicional que nos permite valorar la importancia y trascendencia de la regla a través de su soporte y su confianza. El soporte representa el número de veces que aparece la regla en la base de datos y da una idea de la generalidad o aplicabilidad de la regla. La confianza, por su parte, es un índice de fiabilidad de la regla ($\frac{\text{apariciones de la regla}}{\text{apariciones del antecedente}}$).

El problema de la extracción de reglas asociativas ha sido estudiado prestando una atención especial desde su introducción [Agrawal, R., Imielinski, T., and Swami, A. (1993). "Mining association rules between sets of items in large databases". *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, pp.207-216] a las colecciones de datos procedentes principalmente de transacciones comerciales (conocidas como datos de "cesta de la compra"). Los resultados obtenidos describen los hábitos de compra de los consumidores y tienen una aplicación directa al diseño de catálogos, planificación de la ubicación de artículos en una superficie comercial, segmentación de consumidores, y en general, estrategias de *marketing* que pueden ser conducidas y mejoradas por el nuevo conocimiento descubierto.

Tradicionalmente, el primer paso para la extracción de reglas asociativas consiste en encontrar todos los *itemsets* frecuentes, es decir, conjuntos de elementos que aparecen en la base de datos de transacciones con una frecuencia por encima del soporte mínimo especificado por el usuario. Hemos comprobado que en la mayoría de las situaciones de "cesta de la compra" estudiadas en la literatura, el número de *itemsets* frecuentes descubiertos es limitado, lo que produce un espacio de búsqueda para las reglas asociativas bastante restringido. Sin embargo, para otros escenarios de aplicación como las bases de datos biológicas, donde los patrones a encontrar se producen con unos niveles de soporte muy bajos, nos vemos obligados a ampliar nuestra búsqueda a niveles más y más bajos de soporte, haciendo crecer de forma exponencial el espacio de búsqueda de *itemsets* frecuentes.

Algoritmos bien aceptados y establecidos [Agrawal, R. and Srikant, R. (1994). "Fast Algorithms for Mining Association Rules". *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile. Available at Almaden; Houtsma, M. and Swami, A. (1995). "Set-oriented mining of association rules". *IBM Almaden research center* RJ9567; Zaki, M.J., Partasarathy, S., Ogihara, M. and Li, W. (1996). "Parallel Data Mining for Association Rules on Shared-memory Multi-processors". *Univ. of Rochester. Computer Science Dpt.* TR 618] proceden a explorar el espacio de búsqueda de *itemsets* frecuentes de forma iterativa e incremental, localizando primero los *itemsets* frecuentes formados por un elemento, después los de dos elementos, tres, y así sucesivamente. La construcción de este espacio de búsqueda se basa en generar en la iteración k los candidatos a *itemsets* frecuentes de cardinalidad k a partir de combinaciones de los *itemsets* frecuentes de cardinalidad k-1 encontrados en la iteración anterior. Cuando el número de *itemsets* frecuentes mantiene un ritmo de crecimiento permanente en las primeras iteraciones del método, la construcción combinatoria y posterior exploración de este espacio de búsqueda llega a ser inviable en términos prácticos.

Es en este punto, precisamente, donde hemos centrado nuestros esfuerzos. La estrategia que hemos desarrollado y proponemos con objeto de esta invención resuelve el problema de forma genérica, aportando una solución muy eficiente para casos particulares donde se buscan reglas poco frecuentes que al mismo tiempo obtiene unos resultados comparables a los algoritmos actuales para los casos en los que no es necesario focalizar la búsqueda en patrones tan débiles.

Para entender mejor el objeto de esta invención daremos una definición formal de regla asociativa [Agrawal, R., Imielinski, T., and Swami, A. (1993). "Mining association rules between sets of ítems in large databases". *Proc. of the A CM SIGMOD Inf. Conference on Management of Data*, pp.207-216]:

Definiciones 1

Sea I=$\{i_1, i_2, i_3, ..., i_m\}$ un conjunto de m atributos que llamaremos elementos. Sobre los elementos de I definimos una relación de orden "<" que ordena a todos sus elementos lexicográficamente.

Cada *transacción* $T=\{x_1, x_2, x_3, ..., x_n\}$, en la base de datos de transacciones D, es un conjunto de elementos tales que todos son elementos de I, $\forall$i, $x_i \in I$, y además están ordenados lexicográficamente, $\forall$ x_i,$x_j \in T$, si i<j, entonces $x_i < x_j$

Diremos que una transacción T contiene un *itemset* (conjunto de elementos) de k elementos $A= \{x_1, x_2, x_3, ..., x_k\}$, si todos ellos son elementos de la transacción, $\forall$i, $x_i \in T$, y además están ordenados lexicográficamente, $\forall$ x_i,$x_j \in A$, si i<j, entonces $x_i < x_j$.

Una *regla asociativa* es una expresión de la forma $A \Rightarrow B$, donde A y B son *itemsets*, tales que $A,B \subseteq I$ y $A \cap B = \phi$, llamándose A antecedente y B consecuente de la regla.

El valor de la representatividad, llamado respaldo o *soporte* (support) de A será r cuando el r% de las transacciones presentes en D contengan al *itemset* A.

La *confianza* o cumplimiento (*confidence*) de la regla $A \Rightarrow B$, viene dada por la expresión *soporte*($A \cup B$)/*soporte*(A), que no es más que la probabilidad condicionada de que una transacción contenga B, sabiendo que contiene A.

El significado intuitivo de una regla como $A \Rightarrow B$, es que las transacciones en la base de datos que contienen los elementos del antecedente, A, tienden a contener también los elementos del consecuente, B. Un ejemplo podría ser: "el 89 % de clientes que compran leche, café y azúcar también compran galletas"; en este ejemplo el 89 % se conoce como confianza o cumplimiento de la regla, mientras que el soporte o respaldo de la regla $A \Rightarrow B$, sería el porcentaje de transacciones dentro de la base de datos que contienen el antecedente y el consecuente, es decir, donde se cumple la regla. La tarea de minería de datos para el descubrimiento de reglas asociativas consiste en encontrar todas las reglas que satisfagan un soporte y confianza mínimos definidos por el usuario.

Esta tarea suele ser dividida en dos pasos. El primero consiste en encontrar todos los conjuntos de elementos (*itemsets*) frecuentes, es decir, todos los *itemsets* que aparecen en la base de datos con una frecuencia por encima de la frecuencia especificada por el usuario, llamada soporte mínimo. El segundo paso consiste en formar reglas, implicaciones, entre los itemsets frecuentes. Esta segunda etapa es relativamente fácil de resolver, y consiste en generar, para todos los itemsets frecuentes A, reglas de la forma $(A - B) \Rightarrow B | \forall B \subset A$, que tengan la confianza mínima exigida por el usuario.

La primera etapa del algoritmo, el problema de encontrar los *itemsets* frecuentes de la base de datos, es la más costosa computacionalmente. Si el número de elementos diferentes de la base de datos es m, hay potencialmente 2^m-1 *itemsets* cuya frecuencia de aparición hay que comprobar, este es el espacio de soluciones que forman todos los posibles subconjuntos de I. Dado que esta primera etapa dedicada a encontrar los *itemsets* frecuentes es la más costosa computacionalmente, es el objetivo de las mejoras que introduce nuestra invención.

Para la mejor comprensión del estado de la técnica describiremos el algoritmo de obligada referencia en este sector: el algoritmo *Apriori* [Agrawal, R. and Srikant, R. (1994). "Fast Algorithms for Mining Association Rules". *Proc. or the 20th Int'l Conference on Very Large Databases*, Santiago, Chile; Agrawal, R. and Shafer, J. (1996). "Parallel Mining of Association Rules". IEEE *Transactions on Knowledge and Data Engineering*, Vol. 8, No. 6 : 962-969; US-Patent 5794209 (08/11/1998): System and method for quickly mining association rules in databases]

El algoritmo *Apriori* localiza todos los *itemsets* frecuentes en una base de datos enumerando los posibles candidatos, contabilizando sus apariciones y seleccionando los que superan el umbral de soporte mínimo. La enumeración de los candidatos se realiza de forma iterativa empezando por los de cardinalidad 1 y aumentado la cardinalidad sucesivamente (2, 3, 4, ...) hasta que no es posible encontrar elementos

frecuentes de cardinalidad mayor.

A partir de la segunda iteración, *Apriori* usa la propiedad de que todos los subconjuntos de un *itemset* frecuente deben ser igualmente frecuentes para limitar el número de *itemsets* candidatos a explorar. De esta forma, tan sólo los *itemsets* frecuentes de k elementos son usados para construir los itemsets candidatos de k+1 elementos, reduciendo así enormemente los posibles candidatos con cardinal k+1. Si no nos basásemos en esta propiedad, habría que comprobar todas las posibles combinaciones que existen de candidatos con k+1 elementos: $C_{\#(I)}{}^{k+1}$.

Podemos ver un esbozo del algoritmo a continuación:

1. L[1]={elementos frecuentes en la base de datos};

*/*Recuento inicial conjuntos con 1 elemento*/*

2. for (k=2; L[k-1] $\neq \phi$; k++)

3. {

4. C[k]= *Candidatos_Apriori(L[k-1])*; */*Generación de itemsets candidatos*/*

5. for all transacciones: $T_i \in D$ */* Recuento del soporte*/*

6. for all subconjuntos_con_cardinal_k: $s_j \subset T_i$

7. if ($s_j \in$ C[k])s_j.soporte++;

8. L[k]={c $\in$ C[k]| c.soporte $\geq$ soporte_minimo}; */* Filtro por soporte mínimo */*

9. }

En el algoritmo *Apriori* descrito:

- C[k] denota el conjunto de *candidatos* a *itemsets* frecuentes con cardinal k
- L[k] será el conjunto de *itemsets* frecuentes de k elementos.
- D es el conjunto de transacciones que componen la base de datos
- T_i es una transacción de la base de datos

En el primer paso, sin ninguna información en la que basarnos, los candidatos a priori a ser frecuentes son todos y cada uno de los elementos de la base de datos, y por lo tanto hay que contabilizar las apariciones de todos ellos. Los que superen el umbral de soporte mínimo forman L[1]. A partir de aquí, de forma iterativa y mientras haya elementos en L[k-1], se generan los candidatos a priori de la nueva generación C [k], basándonos en los *itemsets* frecuentes obtenidos en la iteración anterior (L[k-1]). Tras realizar el recuento de los candidatos en la base de datos explorando cada transacción $T_i \in$ D, aquellos que superen el umbral de soporte mínimo pasarán a formar el conjunto L[k] de *itemsets* frecuentes de cardinal k. Con ellos se iniciará una nueva iteración para buscar *itemsets* frecuentes con un elemento más.

Es importante hacer notar en este punto que la creación del conjunto de candidatos a priori (línea 4 del algoritmo *Apriori*) y el recuento del soporte de estos candidatos (líneas 5, 6 y 7 del algoritmo *Apriori*) son dos tareas que presentan una dependencia de datos clara y que se encuentran totalmente desacopladas en el algoritmo.

Pasaremos a describir la característica más importante que define al algoritmo Apriori y que lo diferencia ventajosamente de otras propuestas: su método de generación de candidatos (línea 4 del algoritmo *Apriori*). Esta generación se basa en la sencilla propiedad de que todos los subconjuntos de un itemset frecuente deben de ser frecuentes (Proposición 1), y recíprocamente, sólo las extensiones de itemsets frecuentes de k-1 elementos pueden dar lugar a itemsets frecuentes de k elementos (Proposición 2).

Proposición 1

Si A es un *itemset* frecuente, entonces cualquier *itemset* que sea subconjunto de A es también frecuente.

Demostración:

La demostración es trivial puesto que si A aparece en la base de datos un número de veces N, cualquier subconjunto de estos elementos aparece en la base de datos al menos conjuntamente con las apariciones de A, N veces, y si A era frecuente porque N≥soporte_mínimo, entonces cualquier subconjunto también supera el soporte mínimo y será frecuente.

Definición 2

Sea un *itemset* de k elementos $A= \{x_1,x_2, x_3, ..., x_k\}$, definimos a B como una extensión de A si B es un *itemset* de k+1 elementos formado por los elementos de A más uno, es decir $B= \{x_1,x_2, x_3, ..., x_k, x_{k+1}\}$ tal que $x_k < x_{k+1}$.

Proposición 2

Sólo las extensiones de *itemsets* frecuentes de k-1 elementos con otro elemento que esté dentro de los *itemsets* frecuentes de k-1 elementos pueden dar lugar a itemsets frecuentes de k elementos.

Demostración:

Si los *itemsets* frecuentes de k elementos tienen la forma $\{x_1,x_2, x_3, ..., x_{k-1}, x_k\}$, según la Proposición 2.1 todos sus subconjuntos deben de ser frecuentes, entonces su prefijo $\{x_1, x_2, x_3, ..., x_{k-1}\}$ y su sufijo $\{x_2, x_3, ..., x_{k-1}, x_k\}$ forman parte del conjunto de *itemsets* frecuentes de k-1. Por lo tanto, los *itemset* frecuentes de k elementos están formados por un prefijo de k-1 elementos, que denota en sí mismo un *itemset* frecuente, al que se le añade un último elemento que también pertenece a algún *itemset* frecuente de k-1 elementos.

Por lo tanto, podemos generar los candidatos a *itemsets* frecuentes en cada nueva iteración basándonos en los *itemsets* que ya sabemos que son frecuentes, extendiéndolos con un nuevo elemento que también se encuentre entre los *itemsets* frecuentes de la iteración anterior (Proposición 2).

De esta forma, los candidatos C[k] en la k-ésima iteración se generan combinando *itemsets* frecuentes de L[k-1]. Esto puede expresarse como:

$$C'[k]=\{X=\{x_1,x_2, ..., x_{k-2}, x_{k-1}, x_k\} | \exists\ A, B \in L[k-1], A= \{x_1,x_2, ..., x_{k-2}, x_{k-1}\} \\ B= \{x_1,x_2, ..., x_{k-2}, x_k\}, x_{k-1} < x_k, \forall i,j \text{ si } i<j \text{ } entonces x_i<x_j\}$$

Entre estos candidatos formados a partir de los *itemsets* frecuentes L[k-1], nos aseguramos de eliminar todos los candidatos que tengan al menos un subconjunto que no es *itemset* frecuente, puesto podemos asegurar que estos candidatos no serán *itemsets* frecuentes. Los candidatos son filtrados de la siguiente forma:

$$C[k]=\{X=\{x_1,x_2,..., x_k\} | X \in C'[k] \wedge \forall\ A \subset X, A= \{x_1,x_2,..., x_j\}, j=k\text{-}1, A \in L[k\text{-}1]\}$$

Los candidatos así generados se van introduciendo en un árbol *hash* para facilitar la posterior contabilización de las apariciones en las diferentes transacciones de la base de datos.

Para contabilizar la aparición de los candidatos a *itemsets* frecuentes de k elementos en cada transacción T_i de la base de datos, se forman todos los subconjuntos posibles de k elementos contenidos en T_i y se buscan en el árbol *hash* que contiene a los candidatos C[k]. Se incrementa en uno el contador de soporte de aquellos candidatos que aparezcan en la transacción.

El árbol *hash* se construye antes de empezar el recuento con todos los candidatos a explorar. La profundidad máxima del árbol en la k-ésima iteración será k. Los candidatos a contabilizar se almacenan en las hojas, mientras que los nodos internos de profundidad p poseen una tabla *hash* que apunta a nodos de profundidad p+1.

Para buscar en el árbol *hash* un subconjunto de la transacción y comprobar si es un candidato, comenzando desde la raíz del árbol, se dispersa en cada nodo interno de profundidad p usando el elemento p-ésimo del subconjunto, hasta alcanzar una hoja, dónde se comprobará si los candidatos de esta hoja se encuentran en la transacción. Para hacer esta comprobación rápidamente se usa un mapa de bits que representa a la transacción (con los bits de los elementos presentes en la transacción activos). Para evitar la comprobación repetida de una misma hoja (diferentes subconjuntos pueden colisionar en la misma hoja), se utiliza un registro para identificar las hojas visitadas [Zaki, M.J., Partasarathy, S., Ogihara, M. and Li, W. (1996). "Parallel Data Mining for Association Rules on Shared-memory Multi-processors". *Univ. of Rochester. Computer Science Dpt.* TR 618]. Al finalizar el análisis de la transacción hay que marcar todas las hojas como no visitadas antes de empezar a comprobar una nueva transacción (se mantiene una lista de hojas visitadas para hacer más fácil la desactivación de la marca).

Todos los elementos de C[k] que tengan un soporte mayor que el mínimo establecido, se introducen en la lista enlazada L[k], que representará los *itemsets* frecuentes de cardinalmente k. L[k] será usada para generar los candidatos de una nueva iteración si L[k] $\neq \phi$.

El resultado final de la generación de *itemsets* frecuentes será el conjunto de listas {L[1], L[2], L[3], ..., L[n]} con los *itemsets* de diferente cardinal que cuentan con un soporte superior al mínimo especificado. Estas listas se encuentran dispersadas en sendas tablas *hash* que aceleran el acceso para:

- Realizar con mayor eficiencia la comprobación de la existencia de un *itemset* frecuente, necesaria a la hora de realizar la poda del paso de generación de candidatos, donde se comprueba que todos los subconjuntos de cardinal inmediatamente inferior al candidato en cuestión son también frecuentes.
- Encontrar el soporte de un *itemset* frecuente a la hora de generar las reglas (hay que comprobar el soporte de los *itemsets* para calcular la confianza de las reglas).

Otros métodos que intentan encontrar nuevas y mejores estrategias para localizar patrones frecuentes para el descubrimiento de reglas asociativas han sido ya objeto de numerosas patentes. A pesar de no encontrar en estos métodos una relación directa con la invención que se propone, es interesante mencionarlos por el amplio interés que ha generado en este campo. Solo por mencionar algunos ejemplos podríamos señalar:

- US-Patent 6272478 (08/07/2001): Data mining apparatus for discovering association rules existing between attributes of data
- US-Patent 6230153 (05/08/2001): Association rule ranker for web site emulation
- US-Patent 5920855 (07/06/1999): On-line mining of association rules
- US-Patent 5983222 (11/09/1999): Method and apparatus for computing association rules for data mining in large database
- US-Patent 5794209 (08/11/1998): System and method for quickly mining association rules in databases
- US-Patent 5812997 (09/22/1998): Method and apparatus for deriving an association rule between data
- US-Patent 5615341 (03/25/1997): System and method for mining generalized association rules in databases

Esta metodología de análisis puede exportarse a una extensa gama de áreas en las se necesita procesar e interpretar conjuntos masivos de datos que se estructuran en registros que nosotros trataremos como si fuesen las transacciones del problema de la cesta de la compra. Podemos mencionar tan sólo como ejemplo dos dominios de aplicación muy interesantes, es el caso de las imágenes médicas utilizadas para diagnóstico y el de los datos biológicos generados por los proyectos sobre secuenciación de genomas.

Debido a que los conjuntos de datos que forman estas colecciones son extremadamente extensos y complejos, éstos generan espacios de búsqueda que a menudo resultan prácticamente intratables con las soluciones actuales, por lo que se hace necesario desarrollar algoritmos muy eficientes para resolver las computacionalmente caras tareas de análisis y localización de reglas asociativas.

En definitiva, con la soluciones disponibles en el estado actual de la técnica, se presentan problemas de ineficacia cuando queremos explorar soportes muy bajos para encontrar reglas que no son muy generales,

ni aplicables a una gran cantidad de transacciones, sino a sólo unas pocas, pero con una gran fiabilidad. Un ejemplo de este tipo de reglas sería, en el caso de las bases de datos biológicas, el estudio de reglas que afectan a grupos concretos de, por ejemplo, proteínas o genes pertenecientes a una misma familia funcional, donde las relaciones existen de forma muy localizada en familias concretas, de manera que la relación entre genes o proteínas de diferentes familias sería muy débil. Tanto las relaciones dentro de una pequeña familia, como las relaciones entre diferentes familias podríamos definirlas como débiles (poco frecuentes), lo que obliga a un análisis usando un soporte muy bajo.

Desde un punto de vista computacional, las características propias de los datos complejos de bases de datos como las biológicas, junto con nuestro objetivo explicitado de encontrar reglas asociativas entre patrones débiles, tienen un impacto muy importante en la carga computacional total, como en su distribución sobre las dos tareas principales en las que se divide el algoritmo *Apriori*: la generación de candidatos y el recuento del soporte de los mismos.

Por un lado, y como era de esperar, al bajar el soporte mínimo exigido para intentar encontrar reglas poco frecuentes el tiempo de cálculo crece, pero además lo hace de forma exponencial.

Por otro lado, la carga computacional, concentrada originalmente en el recuento del soporte de los candidatos a *itemsets* frecuentes (líneas 5, 6 y 7 del algoritmo *Apriori*), se desplaza de forma muy importante hacia la tarea de la generación y enumeración de esos candidatos (línea 4 del algoritmo *Apriori*). La razón fundamental para este desplazamiento de la carga a la tarea de generación de candidatos recae en el hecho de que la generación de candidatos es un proceso combinatorio, que tiene como entrada los *itemsets* frecuentes de cada iteración para producir la siguiente generación de candidatos, y el número de estos candidatos que hay que combinar crece exponencialmente al bajar el soporte.

Este desplazamiento de la carga es muy relevante porque los algoritmos actuales han optimizado al máximo el recuento de candidatos, por ejemplo, mediante el uso de árboles *hash* para comprobar si en las transacciones se encuentra alguno de los *itemsets* candidatos. La introducción de estas optimizaciones siempre han estado encaminadas a acelerar el proceso de recuento. Ahora bien, estas optimizaciones son poco importantes cuando el esfuerzo hay que realizarlo en otra tarea: la generación de candidatos.

Además de que la carga crece y se desplaza, ésta se mantiene en el tiempo sobre sucesivas iteraciones del algoritmo. Esto se debe a que al bajar el soporte aparecen *itemsets* frecuentes con un gran número de elementos (este efecto es consecuencia de las fuertes interrelaciones que existen en los datos a nivel muy localizado dentro de pequeños grupos o clusters). El tamaño máximo de *itemset* condiciona y define el número de iteraciones que se deberán realizar, así que cuando bajamos el soporte mínimo el proceso se alarga con más iteraciones.

El resultado final de esta modificación en el comportamiento del algoritmo cuando se buscan reglas poco frecuentes (de soporte bajo) en conjuntos de datos como los que hemos descrito consiste, básicamente, en que el algoritmo *Apriori*, y en concreto su método de exploración del espacio de *itemsets* candidatos a ser frecuentes, se vuelve ineficiente. Expondremos a continuación una serie de ideas y de nuevas estrategias para evitar estos problemas y hacer más eficiente la extracción de reglas asociativas en estos casos en los que el objetivo es la búsqueda de reglas de bajo soporte.

Explicación de la invención

Para afrontar estos problemas y hacer factible el uso de estas técnicas de análisis en nuevos escenarios complejos como las bases de datos biológicas hemos desarrollado un nuevo procedimiento de descubrimiento de patrones frecuentes para la localización de reglas asociativas, cuyas características son el objeto de la presente invención.

Con esta invención se presenta un sistema para la localización de conjuntos de propiedades o elementos que aparecen en una base de datos por encima de una frecuencia mínima exigida muy baja. Su principal propiedad recae en usar a la base de datos como generadora del espacio de búsqueda de los conjuntos frecuentes. Con esto se consigue explorar un espacio de búsqueda más ajustado a la solución final y sólo comprobar posibles soluciones que realmente se encuentran en la base de datos. Otros métodos sin embargo, generan espacios de búsqueda mayores que tienen en cuenta posibilidades que realmente no existen en la base de datos.

La idea consiste en ir extrayendo de la base de datos los conjuntos que pueden llegar a ser frecuentes y contabilizar sus apariciones. Cada vez que nos cercioremos de que un elemento (o transacción completa)

de la base de datos no es frecuente la eliminaremos para no tenerla más en cuenta a la hora de buscar nuevos conjuntos frecuentes.

De esta forma se acelera notablemente la búsqueda de conjuntos basada en la base de datos, puesto que la porción de base de datos que hay que tener en cuenta para la generación de candidatos a ser frecuentes decrece rápidamente con el avance de la búsqueda. Para acelerar la búsqueda además se usan estructuras hash (de indexación) complejas (árboles y listas) para realizar la contabilización de las apariciones de los conjuntos frecuentes de forma eficiente.

La eliminación de parte de la base de datos del proceso de generación del espacio de búsqueda del algoritmo, vital para el funcionamiento eficiente del método, se realiza mediante el uso de mapa de bits que indiquen el estado (activo o eliminado) de la porción de la base de datos representa, o en una realización opcional se puede ir re-escribiendo periódicamente la base de datos sólo con la parte activa de la misma con lo que se requiere mayores recursos de entrada/salida, pero menores cantidades de memoria principal.

Por lo tanto, es objetivo de la presente invención presentar un nuevo método de localización de conjuntos frecuentes orientado al descubrimiento de reglas asociativas en bases de datos de alta complejidad. El procedimiento que hemos propuesto reduce la complejidad del proceso de generación de candidatos basándonos en los elementos presentes en la base de datos, reduciendo a la vez el espacio de candidatos a explorar, y haciendo así posible una solución factible a este importante problema. Además, cuando necesitamos bajar los umbrales de soporte mínimo, el número de elementos frecuentes que aparecen en la solución crece enormemente, haciendo nuestra solución mas eficiente comparativamente en estas ocasiones.

Breve descripción de las figuras

Para la mejor compresión de cuanto queda descrito en la presente memoria, se acompañan las siguientes figuras:

Figura 1

Representación gráfica de la generación de candidatos a priori (línea 4 del algoritmo descrito en la figura 1). Imaginemos que los de *itemsets* frecuentes de cardinal 2 sean: L[2]={{A,C}, {A,T}, {C,L}, {C,P}, {C,T}}. Según el proceso de generación de candidatos expuesto, en un primer paso obtendríamos por combinación los candidatos C'[3] = { {A,C,T}, {C,L,P}, {C,L,T}, {C,P,T}}. Tras eliminar todos los candidatos de C'[3] que tienen algún subconjunto de 2 elementos no frecuente, el resultado sería un único candidato C[3]={{A,C,T}}, que tendríamos que buscar en la base de datos para comprobar si es frecuente.

Figura 2

Representación gráfica de una iteración del algoritmo a priori (pseudo código en figura 1). La parte superior de la figura representa la generación del espacio de búsqueda, los candidatos a priori, basada en los itemsets frecuentes de la iteración anterior L(k-1), y también la dispersión (indexación) de los candidatos, C(k), en el árbol para su recuento eficiente. En la parte inferior se representa la búsqueda usando el árbol de las combinaciones presentes en la base de datos. Como se puede apreciar las tareas de generación y búsqueda están muy bien delimitadas, esa separación obliga a realizar dos procesos combinatorios diferentes.

Figura 3

Representación gráfica de las tendencias de crecimiento del tiempo de proceso y del espacio de soluciones cuando el soporte disminuye. El eje vertical (tiempo y número de *itemsets* frecuentes) está en escala logarítmica.

Figura 4

Evolución del número de transacciones activas a lo largo de las iteraciones del algoritmo a priori. Se observa una disminución substancial del número de transacciones que contienen conjuntos frecuentes al avanzar el algoritmo en sucesivas iteraciones.

Figura 5

Espacio de búsqueda de candidatos *Apriori* y resultado final de *itemsets* frecuentes. Se observa como el espacio de búsqueda que se explora por algoritmos como *Apriori* es mucho mayor que la solución final.

Figura 6

Representación gráfica de una iteración del algoritmo *Generación de candidatos basada en transacciones activas*. Como puede apreciarse, si comparamos este esquema con el de la iteración del algoritmo *Apriori* (figura 2), ha desaparecido la etapa inicial de generación de candidatos que aquí se encuentra acoplada con el recuento de candidatos. Para poder realizar esta generación basada en el contenido de la base de datos es necesario actualizar la información que permita eliminar del proceso de búsqueda las secciones de la base de datos que no sean productivas.

Figura 7

Pseudocódigo del algoritmo de extracción de reglas asociativas con *Generación de Candidatos basada en Transacciones*.

Figura 8

Descomposición de la carga computacional en las dos tareas fundamentales del algoritmo de extracción de reglas asociativas: generación de candidatos y recuento del soporte. Este es un ejercicio de extracción de reglas de bajo soporte en un conjunto real de datos biológicos usando *Apriori*. Como puede observarse, la carga computacional está claramente concentrada en la tarea de generación de candidatos.

Figura 9

Reducción del espacio de búsqueda (candidatos a *itemsets* frecuentes en cada iteración). La línea inferior, marcada con aspas, representa la evolución del número de *itemsets* frecuentes (la solución) en cada iteración del algoritmo. Justo encima de ésta, y marcada con círculos, se encuentra el espacio de búsqueda explorado por nuestra propuesta que se ajusta perfectamente y de forma muy rápida a la solución. La línea superior representa el espacio explorado por el algoritmo *Apriori*, que termina ajustándose algo más tarde a la solución, explorando gran cantidad de candidats innecesarios en la primera etapa de la ejecución del algoritmo.

Figura 10

Tiempos de ejecución para el conjunto de datos real (base de datos biológicas) normalizados con respecto a *Apriori* para apreciar mejor las ganancias relativas. Los resultados mejoran en un orden de magnitud para bajos soportes. Los resultados muestran de izquierda a derecha los tiempos de ejecución del algoritmo (1) *apriori*, (2) añadiendo el bitmap de transacciones inactivas, (3) generando los candidatos en función de la transacción, (4) eliminando además los elementos no activos de la transacción.

Descripción de una realización preferida de la invención

Dividiremos la descripción de la realización de la invención en tres partes para su exposición más clara:

I.- Mapa de transacciones activas:

Si comprobamos cuántas transacciones se usan realmente para contabilizar los *itemsets* frecuentes en cada iteración (figura 4) podremos constatar que conforme realizamos nuevas iteraciones del algoritmo, el conjunto de transacciones donde encontramos algún candidatos se reduce cada vez más. En el ejemplo de la figura 4, las transacciones iniciales, unas 16.000, llegan a reducirse a menos de cien en las últimas iteraciones del algoritmo. Además, podemos asegurar que una transacción en la que no se encontró ningún candidato a *itemset* frecuente en la iteración k, no contendrá ningún *itemset* frecuente en posteriores iteraciones $i/i>k$.

De igual forma que podíamos asegurar que un *itemset* frecuente de cardinal k debe estar formado por *itemsets* frecuentes de cardinal k1, podemos deducir que el conjunto de transacciones donde encontremos

itemsets frecuentes en la iteración k, debe estar contenido en el conjunto de transacciones donde habíamos encontrado *itemsets* frecuentes en la iteración k-1. Por lo tanto, nos basta con revisar en cada iteración sólo las transacciones donde se encontró algún *itemset* frecuente en la iteración anterior y no en toda la base de datos, tal y como hace *Apriori* en todas y cada una de las iteraciones.

De esta forma, una manera directa de reducir el tiempo en el recuento sería, como hemos propuesto, buscar sólo en las transacciones donde la búsqueda tuvo éxito en la iteración anterior. Para ello usamos un mapa de bits donde están representadas todas las transacciones de la base de datos. Inicialmente todas tienen su *bit* a uno, indicando que son transacciones activas en las que hay que buscar *itemsets* frecuentes. En cada iteración del algoritmo sólo buscamos en las transacciones activas y desactivamos aquellas en las que no se encontró ningún *itemset* candidato. De esta forma en cada iteración se reduce el número de transacciones, y por lo tanto, se va reduciendo el tiempo necesario para contabilizar la aparición de los candidatos a *itemsets* frecuentes. Para bases de datos con gran número de transacciones, se puede mantener en memoria principal un *bitmap* parcial haciendo uso de estrategias *out-of-core* para grandes vectores.

Con esta sencilla estrategia se han obtenido mejoras del 10 % al 20 % en tiempo de cómputo, dependiendo de cuantas iteraciones tenga que realizar el algoritmo y cómo estén distribuidos los *itemsets* frecuentes en las transacciones. Indudablemente a esta mejora le favorece el hecho de buscar reglas de bajo soporte que generen muchas iteraciones, donde el número de transacciones que intervienen en elementos de bajo soporte se reduce considerablemente. Para situaciones donde las reglas son muy frecuentes y ocurren en gran parte de las transacciones, o se realizan pocas iteraciones, la mejora se ve limitada. No obstante la estrategia es genérica y directamente aplicable a cualquier algoritmo de descubrimiento de reglas iterativo que realiza varias lecturas de la base de datos.

II.- Generación de candidatos basada en transacciones:

La mejora realizada como se describe en el punto anterior no afecta para nada a la tarea de generación de candidatos que, hemos visto, puede llegar a ser clave en el rendimiento de la extracción de reglas. Veremos ahora cómo podemos realizar esta tarea siguiendo la nueva estrategia objeto de la presente invención: generación de candidatos guiada por los datos que residen en la base de datos de forma eficiente.

Recapitulemos. El algoritmo *Apriori* en cada paso de iteración, calcula los *itemsets* frecuentes en dos fases principales:

1. Genera todos los candidatos a *itemsets* frecuentes basándose en los *itemsets* frecuentes encontrados en la iteración anterior.

2. Realiza una lectura de las transacciones de la base de datos para contar cuantas veces aparecen cada uno de los candidatos.

Por último, se extrae de entre los candidatos todos aquellos que superan el límite de frecuencia exigido (soporte mínimo).

Como vemos, antes de buscar en la base de datos los *itemsets* frecuentes de cardinal k usa de forma inteligente la información de iteraciones anteriores para reducir el espacio de búsqueda sólo a los candidatos que de ante mano tienen alguna posibilidad de ser frecuentes. Aunque el espacio de búsqueda ciertamente se reduce, todavía existe una gran diferencia entre los candidatos y los *itemsets* realmente frecuentes, especialmente en las primeras etapas del algoritmo, tal y como podemos ver en la figura 5.

Además de este hecho, existe otro problema en la forma de generar los itemsets que usa *Apriori*. Después de podar los candidatos (usando la propiedad de que todos los subconjuntos de un *itemset* frecuente deben de ser a su vez frecuentes) el espacio de soluciones explorado se aproxima mucho a la solución final, como vemos en las últimas iteraciones de la figura 5. Pero la generación de los candidatos es un proceso combinatorio de los *itemsets* frecuentes de iteraciones anteriores (figura 1), y cuando el número de *itemsets* frecuentes se dispara exponencialmente, de forma que la combinación de todos ellos es un proceso lento y costoso (figura 3).

Este costoso proceso de generación de candidatos es el que vamos a eliminar con nuestra nueva estrategia. En realidad vamos a realizar el recuento del soporte sólo de los candidatos que estén presentes en las transacciones de la base de datos. Para ello vamos a usar las transacciones como generadoras de candidatos de forma que tendremos en cuenta sólo los *itemsets* que estén presentes en estas transacciones como posibles candidatos a ser frecuentes. Desde luego, los candidatos surgidos de las transacciones son

filtrados usando la misma poda que realiza *Apriori* sobre sus candidatos, sólo usaremos los candidatos de k elementos cuyos subconjuntos de k-1 elementos sean frecuentes, con lo que nos aseguramos no explorar ningún candidato innecesariamente. Esto resultará eficiente porque, además, en cada iteración vamos reduciendo el número de transacciones activas gracias a la estrategia introducida en el epígrafe anterior, con lo que el espacio de soluciones a explorar se acota un poco más en cada iteración.

Veamos cómo implementar la nueva estrategia. En un principio eliminamos la etapa de generación de candidatos a priori y la complejidad que esta conlleva, lo que es una ventaja inicial. Pero la mayor virtud es que no hay que introducir ninguna complejidad extra para ir generando los candidatos presentes en la transacción, porque esto lo haremos sobre la marcha, al tiempo que exploramos las transacciones para realizar el recuento del soporte.

Para contabilizar el soporte, el algoritmo *Apriori* realiza todas las combinaciones de k elementos de cada transacción y comprueba si estas combinaciones se encuentran en el conjunto de candidatos para contabilizar estas apariciones. Nosotros hacemos lo mismo, pero empezando con el conjunto de candidatos vacío. Realizaremos todas las combinaciones posibles de k elementos en cada transacción y comprobaremos si están entre los candidatos, en caso afirmativo contabilizamos la aparición, de lo contrario, lo convertimos en un nuevo candidato (siempre que todos sus subconjuntos de k-1 elementos sean frecuentes) y lo anotaremos contabilizando una aparición.

Por lo tanto, nuestro algoritmo no necesita de ningún paso extra que añada complejidad al problema. Por el contrario, ahorra el paso de generación de candidatos corazón del algoritmo *Apriori*, que tiene una complejidad exponencial (debido a la naturaleza combinatoria) respecto al número de conjuntos frecuentes encontrados.

La fase inicial de generación de candidatos que realizaba *Apriori*, en nuestra aproximación está totalmente imbricada en el proceso de recuento del soporte. A la vez que vamos contando, vamos creando el conjunto de candidatos a contar con las combinaciones que surgen de las propias transacciones. Cuando una de estas combinaciones está en el conjunto de candidatos, incrementamos la variable que registra su soporte. Si no está entre los candidatos, comprobamos si es factible (si todos sus subconjuntos son frecuentes) para añadirla al conjunto de candidatos, y contar cuántas veces más aparece en el resto de transacciones.

En la figura 6 se ha representado una iteración del algoritmo propuesto en esta invención, que puede ser comparado con el algoritmo *Apriori*, referencia en el estado de la técnica actual, en la figura 2. En estas representación esquemáticas de ambos procedimientos se pone de manifiesto la eliminación del proceso de generación combinatorio presente en el algoritmo *Apriori* (figura 2 - arriba) y que genera una gran carga computacional responsable de la mayor parte del tiempo necesario para realizar la tarea de descubrimiento de conjuntos frecuentes (figura 8).

Si vemos la figura 7 podemos observar el seudocódigo del algoritmo que implementa la estrategia presentada. En este seudocódigo podemos apreciar cómo hemos introducido el *bitmap* de transacciones transaccion_activa, que usamos para sólo explorar las transacciones dónde es posible encontrar nuevos itemsets frecuentes. Mediante el centinela transaccion usada, controlamos cuándo una transacción no se ha usado para encontrar ningún candidato, y marcarla como inactiva, de forma que la eliminamos de posteriores recuentos. Además, falta el cálculo a priori de los candidatos, que se van incorporando sobre la marcha (líneas 18 a 22 de la figura 7) conforme van surgiendo de la exploración de las propias transacciones.

III.- Reducción de elementos activos en una transacción

A continuación describimos la realización de otra mejora opcional que hemos introducido. De la misma forma que vamos eliminando transacciones que no se usan para generar nuevos *itemsets* frecuentes, podemos ir eliminando de las transacciones que quedan, los elementos individuales que no se usan para construir ningún *itemset* frecuente en dicha transacción. De esta forma, éstos se eliminan de las combinaciones que hay que realizar en cada transacción para crear los candidatos elementos que seguro no van a generar candidatos con éxito.

En el peor de los casos, en la iteración k, en una transacción formada de m elementos, con $m>k$ habría que comprobar la posibilidad de que $C_m{}^k$ *itemsets* fuesen candidatos a ser frecuentes. Si sólo tenemos j elementos activos en la transacción, con $m \geq j \geq k$, podríamos ahorrarnos $C^k{}_m$ - $C_j{}^k$ comprobaciones teniendo en cuenta sólo los j elementos activos de la transacción.

Para realizar esta eliminación de los elementos inactivos de las transacciones se puede optar por ir rescribiendo la base de datos en cada paso de la iteración, de forma que sólo contenga las transacciones y elementos que van quedando activos. No hemos usado *bitmaps*, como en el caso de las transacciones, porque mantener un *bitmap* para todos los elementos individuales en la base de datos sería demasiado costoso en términos de memoria. Como veremos en los resultados descritos a continuación, la sobrecarga de E/S introducida por esta reescritura llega a ser compensada por las ganancias en tiempo obtenidas.

Después de comprobar que los resultados obtenidos con los conjuntos de datos sintéticos usados en la literatura a modo de conjunto de pruebas de rendimiento(*bechmark*) son equiparables a los obtenidos por los algoritmos del estado actual de la técnica, pasaremos a comprobar como se comporta nuestra propuesta cuando se enfrenta a un conjunto de datos reales donde hay una gran acumulación de conjuntos frecuentes con bajo soporte como las bases de datos biológicas.

En la figura 10 se muestran los tiempos de ejecución de los algoritmos implementados a partir de nuestra invención para un conjunto de datos reales extraído de la base de datos *SwissProt* [Bairoch, A. and Apweiler, R. (2000). "The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000". *Nucleic Acids Research*, vol. 28, 1: 45-48], uno de los repositorios de información mas conocido y fiable que existe para secuencias de proteínas. Hemos usado la versión v.38 que está compuesta por 80.000 proteínas. Y nos hemos centrado en analizar los campos *keyword* y *feature* de esta base de datos.

Como puede observarse en la citada figura 10, el uso de el *bitmap* para eliminar transacciones obtiene mejoras del 10 % al 20 % en tiempo de ejecución respecto al algoritmo original, gracias tan sólo a que se le ha introducido la mejora del mapa de transacciones activas. El algoritmo de generación de candidatos basada en transacciones, *TD*, obtiene sus mejores resultados para bajos soportes donde se llega a alcanzar reducciones en un orden de magnitud. Incluso para soportes altos nuestra propuesta (*TD*) se comporta mejor que la referencia del algoritmo *Apriori*. La eliminación de elementos inactivos de las transacciones (rescribiendo el fichero de transacciones), que realiza la implementación "*TD* elementos", introduce mejoras adicionales de hasta el 10 % de tiempo de ejecución, absorbiendo bien la penalización de E/S que introduce al rescribir la base de datos de transacciones eliminando los elementos innecesarios.

El éxito de nuestra propuesta recae fundamentalmente en tres puntos:

- La eliminación de transacciones que no contribuyen a la formación de conjuntos frecuentes reduce el tiempo para el recuento del soporte, más cuantioso cuando el número de iteraciones es elevado y este recuento debe de repetirse en cada iteración.

- La *mejora fundamental* se produce en el proceso de generación de candidatos, que se acelera notablemente gracias a la eliminación del proceso combinatorio que usan los algoritmos actuales. Esta mejora es fundamental en los tipos de distribuciones de datos. Como podemos apreciar en la figura 8, cuando realizamos búsquedas usando soportes bajos, la carga computacional se encuentra concentrada en la tarea de generación de candidatos.

- Por último, el espacio de candidatos se reduce (figura 9) usando el nuevo método de generación de candidatos basado en la transacción, ya que sólo se generarán candidatos que están presentes en la base de datos en al menos una transacción. Esto permite que el recuento se agilice aún más.

REIVINDICACIONES

1. Procedimiento para el descubrimiento de conjuntos frecuentes en bases de datos consistente en:

a. la generación de conjuntos candidatos a ser frecuentes basado en la información que aparece en las transacciones de la base de datos.

b. el uso de un árbol de búsqueda hash para la comprobación de si un conjunto es frecuente y contabilizar las apariciones de dichos conjuntos frecuentes en la base de datos.

c. la comprobación de la validez a priori de los candidatos a ser frecuentes mediante la aplicación de la proposición 2 antes de su inclusión en el árbol de búsqueda.

d. la reducción de las transacciones de la base de datos basado en la imposibilidad de que una transacción contenga parte de la solución esperada (conjuntos frecuentes).

e. la eliminación de elementos individuales de la transacción que no participará ni lo hará en conjuntos frecuentes.

2. Procedimiento para el descubrimiento de conjuntos frecuentes en bases de datos, según reivindicación 1, **caracterizado** por su realización mediante una exploración del espacio de búsqueda guiada por el propio contenido de las transacciones de la base de datos.

3. Un sistema que es capaz de generar reglas asociativas basadas en los conjuntos frecuentes descubiertos gracias al procedimiento reivindicado en 1 y 2, en el que la mejora conseguida permite encontrar reglas asociativas poco frecuentes en bases de datos complejas como las biológicas.

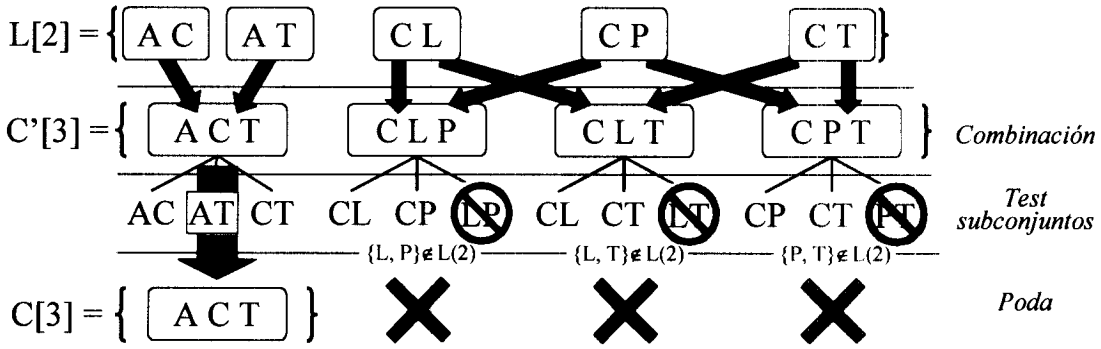
L[2] = { A C A T C L C P C T }
C'[3] = { A C T C L P C L T C P T }
Combinación
AC AT CT CL CP LP CL CT LT CP CT PT
Test subconjuntos
{L, P}∉L(2)
{L, T}∉L(2)
{P, T}∉L(2)
C[3] = { A C T }
Poda

FIGURA 1

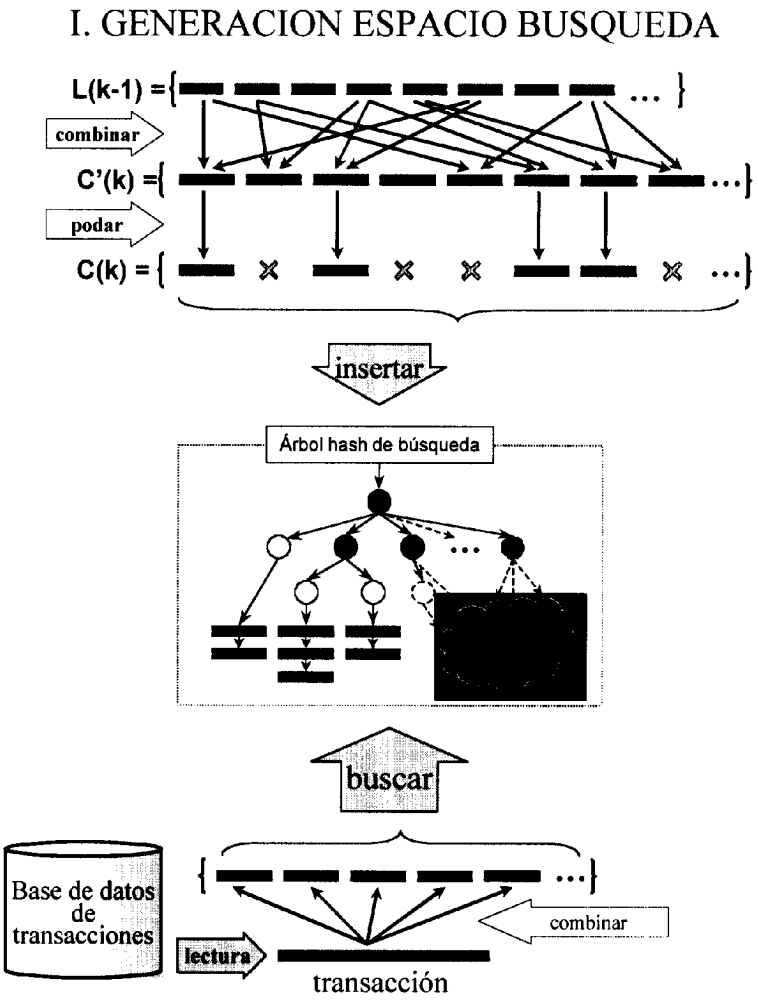
I. GENERACION ESPACIO BUSQUEDA
L(k-1) =
combinar
C'(k) =
podar
C(k) =
insertar
Árbol hash de búsqueda
buscar
Base de datos de transacciones
lectura
combinar
transacción
II. EXPLORACION Y BUSQUEDA

FIGURA 2

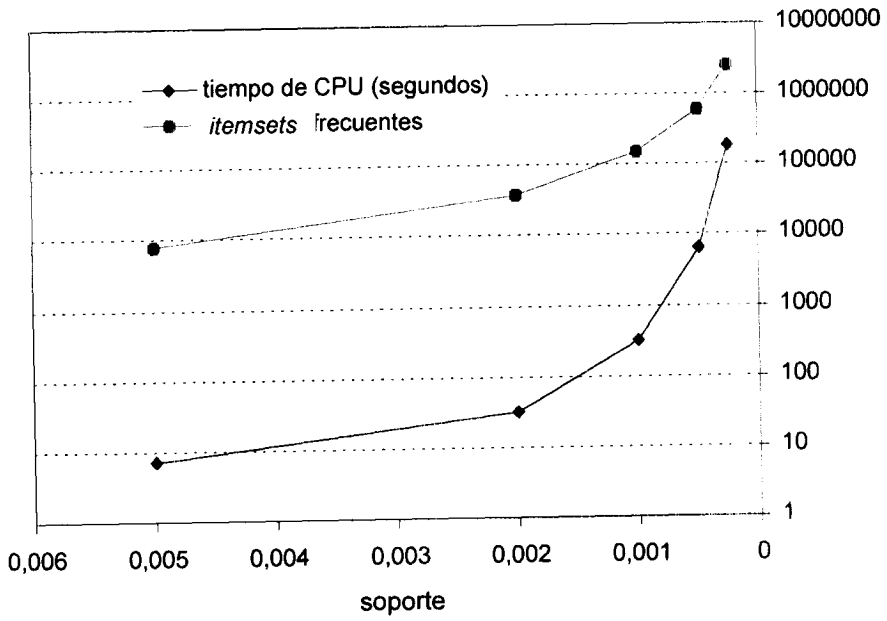
tiempo de CPU (segundos)
itemsets frecuentes
10000000
1000000
100000
10000
1000
100
10
1
0,006
0,005
0,004
0,003
0,002
0,001
0
soporte

FIGURA 3

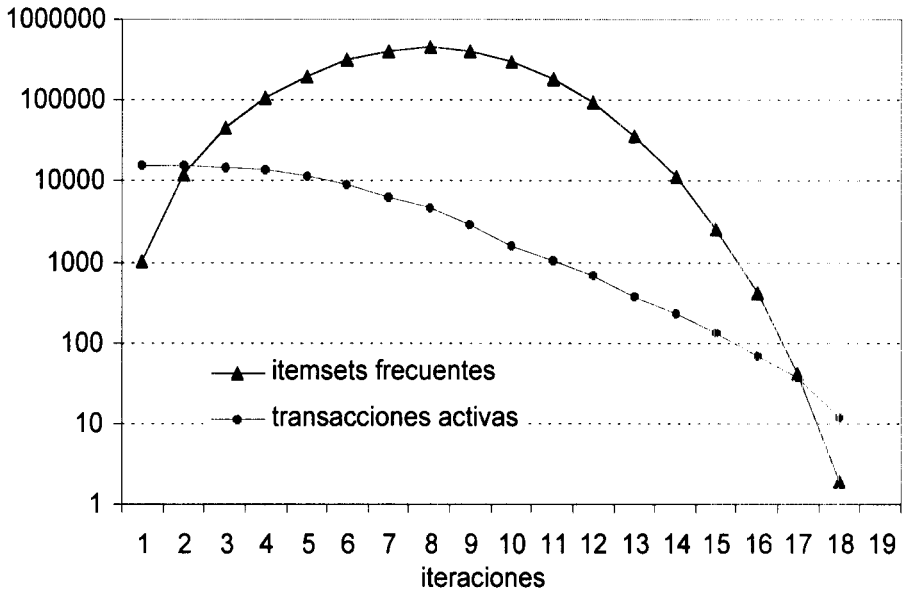
1000000
100000
10000
1000
100
10
1
itemsets frecuentes
transacciones activas
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
iteraciones

FIGURA 4

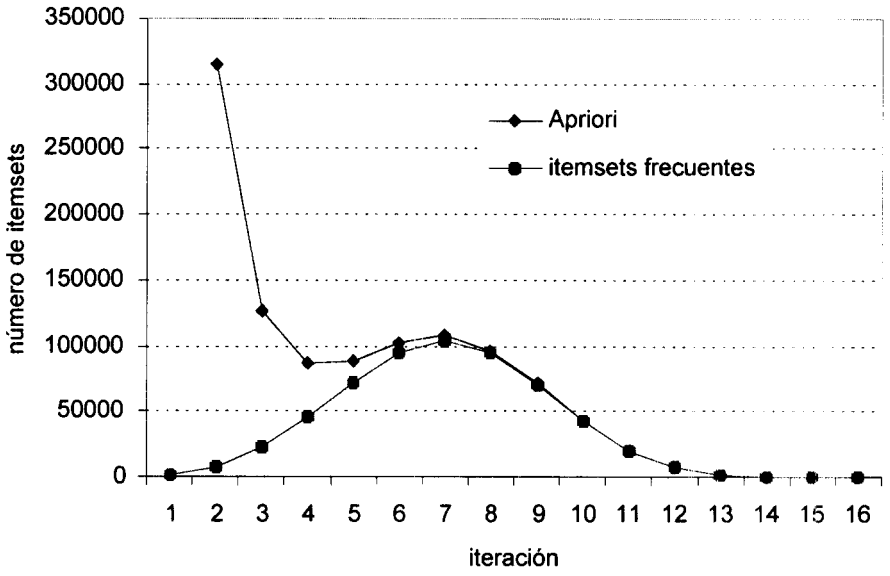
350000
300000
250000
200000
150000
100000
50000
0
número de itemsets
Apriori
itemsets frecuentes
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
iteración

FIGURA 5

GENERACION Y BUSQUEDA CONJUNTA

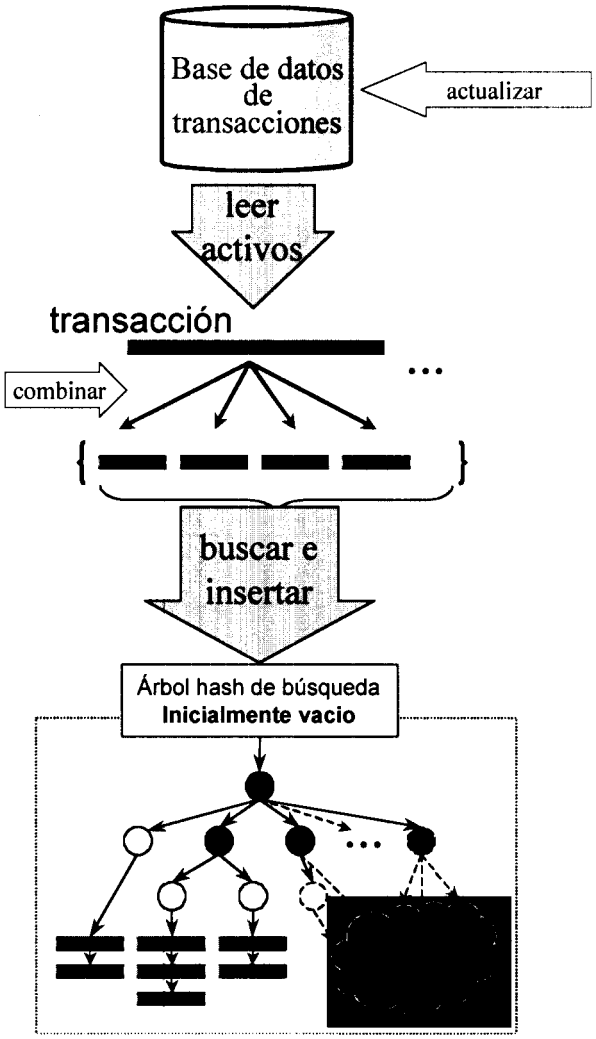

FIGURA 6

```
1. L[1]={elementos frecuentes en la base de datos};
                                        /* Recuento inicial conjuntos con 1 elemento */
2. for all transacciones: T_i ∈ D       /* Inicialización bit map de transacciones activas */
3.   transaccion_activa[i]=1;
4. for ( k=2; L[k-1]≠∅; k++)
5. {
6.   C[k] = ∅;                                  /* Inicialmente no hay candidatos */
7.   for all transacciones: T_i ∈ D && transaccion_activas[i]==1
8.   {
       /* Exploracion sólo de las transacciones activas */
9.     transaccion_usada=0;
10.      for all subconjuntos_con_cardinal_k: s_j ⊂ T_i
11.        if (s_j ∈ C[k])                  /* si el itemset es candidato lo cuenta */
12.        {
13.          s_j.soporte++;
14.          transaccion_usada=1;
15.        }
16.        else
       /* si no es candidato, pero es factible lo añade a C[k] */
17.          if ( all_subconjuntos(s_j)∈ L[k-1] )
18.          {
19.          s_j.soporte = 1;
20.          insertar(s_j , C[k]);
21.          transaccion_usada=1;
22.          }
23.      transaccion_activa[i]=transaccion_usada;

       /* si la transacción no se usa se marca como inactiva */
24.  }
25.  L[k]={c ∈ C[k]| c.soporte ≥ soporte_minimo};   /* Filtro por soporte mínimo */
26. }
```

FIGURA 7

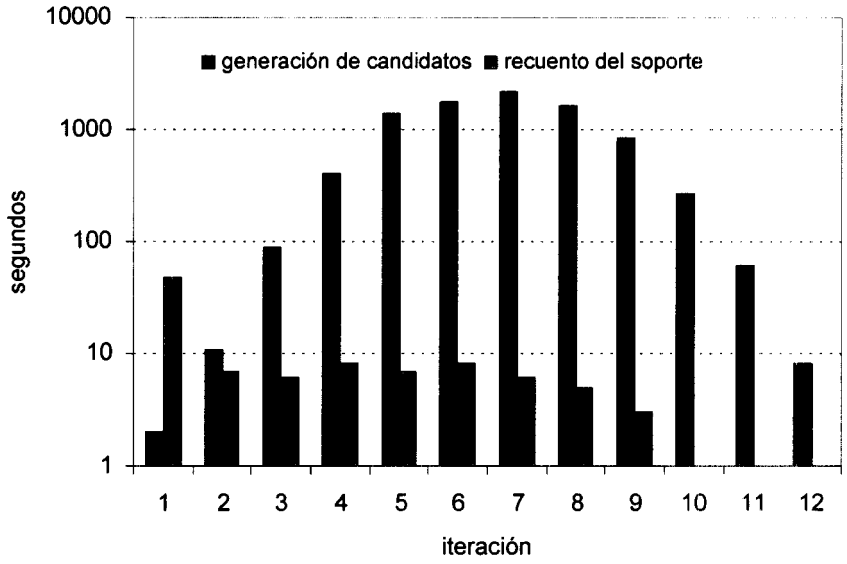
10000
1000
100
10
1
segundos
generación de candidatos
recuento del soporte
1
2
3
4
5
6
7
8
9
10
11
12
iteración

FIGURA 8

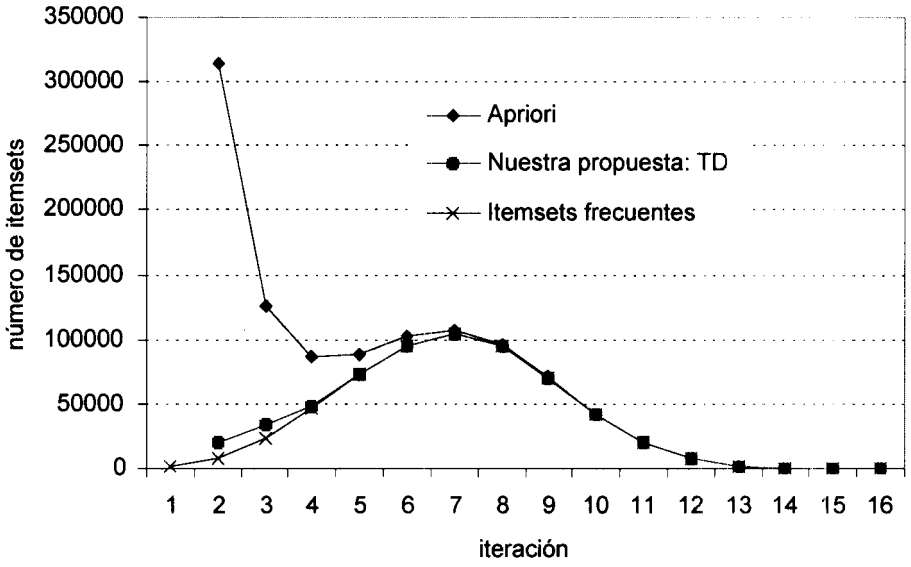
350000
300000
250000
200000
150000
100000
50000
0
número de itemsets
Apriori
Nuestra propuesta: TD
Itemsets frecuentes
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
iteración

FIGURA 9

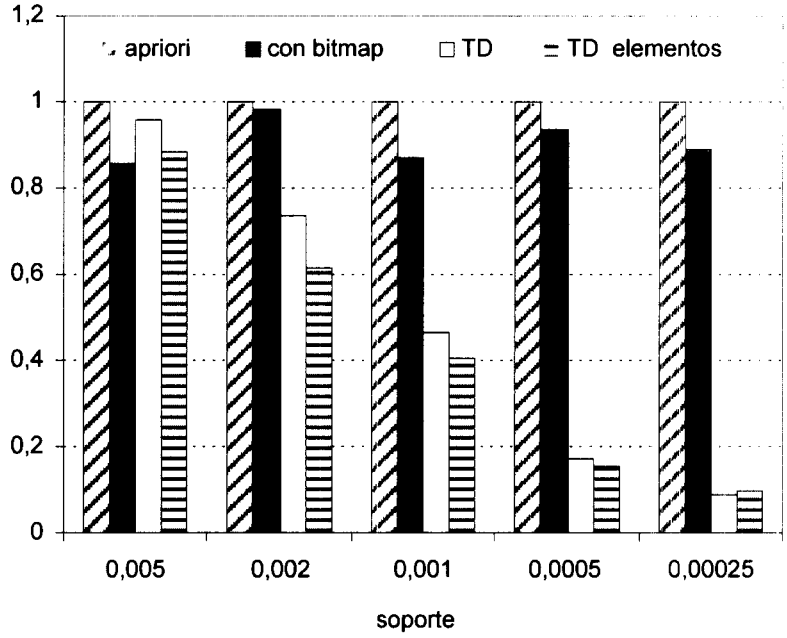
1,2
1
0,8
0,6
0,4
0,2
0
apriori
con bitmap
TD
TD elementos
0,005
0,002
0,001
0,0005
0,00025
soporte

FIGURA 10

OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) ES 2 190 747

(21) Nº de solicitud: 200102582

(22) Fecha de presentación de la solicitud: **22.11.2001**

(32) Fecha de prioridad:

INFORME SOBRE EL ESTADO DE LA TÉCNICA

(51) **Int. Cl.[7]:** G06F 17/30

DOCUMENTOS RELEVANTES

Categoría	Documentos citados	Reivindicaciones afectadas
E	US 6389416 B1 (AGARWAL et al.) 14.05.2002, todo el documento.	1-3
E	US 2003009456 A1 (SHINTANI et al.) 09.01.2003, todo el documento.	1-3
X	US 6311179 B1 (AGARWAL et al.) 30.10.2001, todo el documento.	1-3
A	US 6278998 B1 (OZDEN et al.) 21.08.2001, todo el documento.	1-3
Y	US 6301575 B1 (CHADHA et al.) 09.10.2001, todo el documento.	1-3
Y	US 5742811 A (AGARWAL et al.) 21.04.1998, todo el documento.	1-3

Categoría de los documentos citados

X: de particular relevancia
Y: de particular relevancia combinado con otro/s de la misma categoría
A: refleja el estado de la técnica
O: referido a divulgación no escrita
P: publicado entre la fecha de prioridad y la de presentación de la solicitud
E: documento anterior, pero publicado después de la fecha de presentación de la solicitud

El presente informe ha sido realizado

☒ para todas las reivindicaciones ☐ para las reivindicaciones nº:

Fecha de realización del informe	Examinador	Página
01.07.2003	M. Fluvià Rodríguez	1/1