RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Estado del arte para el minado de patrones secuenciales

José Kadir Febrer Hernández y
José Hernández Palancar

RT_016 **junio 2011**

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

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

RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Estado del arte para el minado de patrones secuenciales

José Kadir Febrer Hernández y
José Hernández Palancar

RT_016 **junio 2011**

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

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

Tabla de contenido

1 Introducción 1
2 Marco teórico 2
2.1 Conceptos preliminares 2
2.2 Estrategia basada en la generación de candidatos 4
2.3 Estrategia basada en el crecimiento de patrones 5
2.4 Estrategia seguida por los algoritmos para bases dinámicas 6
2.5 Métodos para la proyección de las bases de datos 7
2.5.1 Método bi-level projection 8
2.5.2 Método pseudo proyección (*pseudo-projection*) 8
2.6 Generalizando el proceso de minado de secuencias frecuentes 9
2.6.1 Límite de tiempo o GAP 9
2.6.2 Ventana de tiempo deslizante 11
2.6.3 Taxonomía 11
3 Algoritmos de detección de secuencias en bases de datos estáticas 12
3.1 Algoritmos basados en apriori 13
3.1.1 Algoritmos GSP y MSPX 13
3.1.2 Algoritmo CAMLS 14
3.1.3 SPIRIT, uso de expresiones regulares para la detección de secuencias frecuentes 16
3.2 Algoritmos basados en crecimiento de patrones 18
3.2.1 Algoritmo PrefixSpan 18
3.2.2 Algoritmo PRISM 20
3.2.3 Algoritmo SPADE 24
3.2.4 Algoritmo MEMISP 26
3.2.5 Algoritmo LAPIN 28
3.3 Algoritmos de detección de secuencias frecuentes cerradas 29
3.3.1 Algoritmo BIDE 29
3.3.2 Algoritmo CloSpan 31
3.4 Síntesis y conclusiones 34
4 Algoritmos de detección de secuencias en bases de datos dinámicas 34
4.1 Algoritmo IncSpan 35
4.2 Algoritmo IMCS 36
4.3 Algoritmo CISpan 38
4.4 Síntesis y conclusiones 39
5 Conclusiones 39

Lista de figuras

1 Secuencia frecuente sin límite de tiempo. 9
2 Secuencia frecuente con límite de tiempo de 3 meses. 10
3 Significado en el uso de GAP. 10
4 Secuencia de ejemplo donde cada transacción ocurre en un día distinto. 11
5 Secuencia original a la que se le aplica una ventana de tiempo de un tamaño de tres días. 11
6 Parte de una taxonomía hecha a los libros de un club de lectores. 12

7 Secuencia frecuente sin uso de taxonomías. 12
8 Secuencias frecuentes usando la taxonomía de la figura 6. 12
9 Taxonomía de los algoritmos que usan bases de datos fijas y la estrategia apriori. 13
10 Autómata para la expresión regular: a* (bc — bf*d — e). 16
11 Taxonomía de los algoritmos que usan bases de datos fijas y crecimiento de patrones. 18
12 Ejemplo de proyección de una base. 20
13 Retículo formado por el conjunto de números primos (C = 2, 3, 5, 7). 22
14 Extensión del último ítemset de la secuencia (itemset extension). 23
15 Extensión por secuencia (sequence extension). 23
16 Ejemplo de *id_list*, listas de identificadores de los ítems frecuentes. 24
17 Ejemplo de clases de equivalencia inducidas por la relación de equivalencia θ_1. 25
18 Ejemplo de intersección de *id_list*. 26
19 Indexado para el patrón secuencial $\langle d \rangle$. 27
20 Indexado para el patrón secuencial $\langle d, k \rangle$. 27
21 BD proyectada del prefijo j y soporte de los ítems locales. 30
22 Árbol lexicográfico de secuencias. 32
23 Sub-patrón y super-patrón hacia atrás. 33
24 Tabla hash usada en el algoritmo CloSpan. 34
25 Taxonomía de algoritmos que usan bases de datos dinámicas. 35
26 a) CSTree antes de actualizar; b) CSTree después de actualizar. 37
27 CSTree después de actualizado la base de datos. 37
28 Mezcla de los retículos de las bases de datos. 38

Lista de tablas

1 Base de datos de secuencias. 3
2 Base de datos proyectada. 4
3 Casos que pueden ocurrir en el análisis entre el nuevo conjunto de secuencias frecuentes y el que se tiene almacenado. 7
4 Matriz correspondiente a una base proyectada. 8
5 Matriz correspondiente a la base proyectada correspondiente al prefijo $\langle (b)\ (k) \rangle$. 8
6 Base de datos de secuencias. 9
7 Base de datos proyectada $\langle b \rangle$. 9
8 Proceso de obtención de secuencias candidatas y de poda. 14
9 Base de datos de secuencias. 19
10 Base de datos proyectada con relación al prefijo $\langle (j) \rangle$ (patrón secuencial). 19
11 Base de datos de secuencias. 20
12 Base de datos proyectada con respecta al ítem d. 20
13 Base de datos de secuencias. 21
14 *Bit-encoded pos* para el ítem k. 21
15 *Bit-encoded sid* para el ítem k. 21
16 *Bit-encoded sid* para los ítems k, h. 22
17 Base de datos de secuencias. 27
18 Base de datos de secuencias. 28
19 Listado de las últimas posiciones de los ítems por secuencia. 28

20 Tabla comparativa de los algoritmos que descubren patrones secuenciales en bases de datos estáticas. 34
21 Base de datos de secuencias. 36
22 Tabla comparativa de los algoritmos que descubren patrones secuenciales en bases de datos estáticas. 39

Estado del arte para el minado de patrones secuenciales

J. Kadir Febrer Hernández y José Hernández Palancar

Dpto. Minería de Datos, Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV),
Ciudad de la Habana, Cuba
{jfebrer, jpalancar}@cenatav.co.cu

RT_016, Serie Gris, CENATAV
Aceptado: 30 de diciembre de 2010

Resumen. En el presente trabajo se hace un estudio sobre el estado del arte de los principales algoritmos que existen para determinar secuencias frecuentes. En este estudio se analizan las metodologías de generación de candidatos (Apriori) y de crecimiento de patrones; determinando para cada caso sus características, rasgos que la distinguen y algoritmos que la aplican. Además, se analizan las diferentes restricciones que pueden presentar los algoritmos como pueden ser gaps, ventanas de tiempo o taxonomías.

Palabras clave: secuencias candidatas, secuencias frecuentes, soporte mínimo.

Abstract. In this work, we carry out a study of the state of the art of the algorithms for discovering frequency sequence set. This study analyzes the methodologies apriori-based and pattern-growth, and presents, for each case, its characteristics, distinctive features and the algorithms that implement them. Moreover, we analyze the different constraints that may show the algorithms such as gaps, sliding time windows or taxonomies.

Keywords: candidate sequence, frequent sequence, minimum support.

1 Introducción

Los avances tecnológicos de los últimos años han permitido que se puedan almacenar grandes volúmenes de datos de forma rápida y eficiente, lo que ha traído aparejado la necesidad de crear formas eficientes de procesar estos datos y transformarlos en información útil. De esto se ha encargado la minería de datos como un conjunto de técnicas que permiten explorar estos grandes volúmenes de datos con el objetivo de encontrar patrones frecuentes, tendencias o reglas que expliquen el comportamiento de los datos en determinados contextos.

En 1995, Agrawal y Srikant introducen, con la presentación del algoritmo AprioriAll [1] [2], un nuevo tema en la minería de datos, el minado de patrones secuenciales. Desde su surgimiento la detección de secuencias frecuentes se convirtió en un tema importante de investigación con gran aplicación en los campos de la medicina [3], las telecomunicaciones, el análisis de las transacciones de compras de los clientes, el acceso a las páginas Web [4], entre otros.

El minado de patrones secuenciales consiste en encontrar todas las secuencias frecuentes que aparecen en una base de datos. La tarea de detectar todas las secuencias frecuentes en una base de datos es un problema difícil ya que el espacio de búsqueda es extremadamente grande. Por ejemplo, si se tienen n atributos, existirán 2^{nk} secuencias frecuentes potenciales de longitud k.

Durante el transcurso del tiempo los algoritmos se agruparon en dos grandes grupos según la estrategia en que se basan, la generación de candidatos o apriori ([5]) y el crecimiento de patrones ([6] [7] [8] [9]). También se desarrollaron algoritmos que minan las secuencias cerradas ([10] [11] [12]) o maximales ([13]) frecuentes, algo muy importante cuando se tiene en cuenta la cantidad de secuencias que se obtienen cuando se minan todas las secuencias frecuentes.

El mismo desarrollo en el minado de patrones secuenciales ([14] [15] [16] [17] [18]) trajo consigo que se fuera ampliando el campo de aplicaciones y se crearan algoritmos para minar secuencias frecuentes en bases de datos dinámicas [19] (bases de datos que son modificadas frecuentemente).

En 1996, Srikant y Agrawal introducen junto con el algoritmo GSP [20] la generalización de las secuencias con el uso de GAPs ([21] [22]), ventanas de tiempo ([23]) y taxonomías lo que amplio el minado de patrones secuenciales, permitiendo incluir diferentes restricciones a las secuencias. Más tarde, Garofalakis introduce con SPIRIT [24] un nuevo método para minar secuencias, el uso de autómatas y expresiones regulares. Esto brinda la posibilidad de obtener secuencias que pertenezcan al lenguaje especificado por el autómata.

En el presente trabajo se expondrá el estado del arte sobre el minado de patrones secuenciales; algoritmos, estrategias en que se basan, formas de almacenamiento de las secuencias ([25]) y otras características relacionadas con el minado de secuencias frecuentes.

El documento está organizado de la siguiente forma: En el capítulo 2 se exponen los conceptos básicos fundamentales para definir el problema del minado de patrones secuenciales, las características generales de las estrategias usadas por los algoritmos, diferentes métodos de proyección de bases de datos y lo que se define como generalización del minado de patrones secuenciales. En el capítulo 3 se describen los algoritmos analizados, exponiendo por cada uno sus características, ventajas y desventajas.

Finalmente se exponen las conclusiones del trabajo y las referencias bibliográficas usadas en el estudio y análisis del minado de patrones secuenciales.

2 Marco teórico

En este capítulo se expondrán los conceptos básicos fundamentales relacionados con el minado de patrones secuenciales; se explicarán brevemente las estrategias más usadas en los algoritmos de detección de secuencias frecuentes y las características de los algoritmos que trabajan sobre bases de datos dinámicas (bases de datos donde la información almacenada se puede modificar en el transcurso del tiempo) y; se explicará la generalización de secuencias con el uso de GAP, ventanas de tiempo y taxonomías.

2.1 Conceptos preliminares

Desde que Agrawal y Srikant [1] introducen el minado de secuencias frecuentes, muchos han sido los algoritmos que se han propuesto con el objetivo de darle, de una manera eficiente, solución a este problema. De todos estos estudios han surgido una serie de definiciones y conceptos generales los cuales se relacionan a continuación, lo que además brindara un apoyo para la lectura del documento.

Las definiciones relacionadas con el problema de minado de secuencias frecuente parten de, un conjunto ($I = \{i_1, i_2, ..., i_n\}$) distinto de literales llamados ítems.

Definición 1 (*ítemset*): Un *elemento* o *ítemset* es un conjunto no vacio de ítems. Se denota un elemento e por (e_1, e_2, ..., e_m) donde cada e_j es un ítem.

El itemset es la base para la definición del concepto de secuencia.

Definición 2 (*secuencia*):Una *secuencia* es una lista ordenada de elementos o ítemsets. Se denota una secuencia S por $\langle S_1\ S_2\ ...\ S_n\rangle$ donde cada S_j es un elemento de la secuencia.

Un elemento también se puede considerar como una secuencia (ej. $\langle(e_{11}, e_{12}, ..., e_{1m})\rangle$).

Según la definición y para una mejor comprensión del tema de minado de patrones secuenciales, se pueden definir dos tipos de secuencias: la que está formada por itemsets de un solo ítem cada uno (ej. $\langle e_1\ e_2\ ...\ e_n\rangle$) y las formadas por itemsets que contienen varios ítems (ej. $\langle(e_{11}, e_{12}\ ...\ e_{1m})\ (e_{21}, e_{22}, ..., e_{2m})...(e_{n1}, e_{n2}, ..., e_{nm})\rangle$).

Definición 3: Una secuencia $\langle\alpha_1\ \alpha_2...\alpha_n\rangle$ está contenida en otra secuencia $\langle\beta_1\ \beta_2...\beta_m\rangle$ si existe $i_1 < i_2 < ... < i_n$ tal que $\alpha_1 \leq \beta_{i_1}$, $\alpha_2 \leq \beta_{i_2}$, ..., $\alpha_n \leq \beta_{i_n}$.

Por ejemplo, si se tiene $S_1 = \langle(k)(j, i)\rangle$, $S_2 = \langle(k, h)(j, i)\rangle$ se puede decir que S_1 está contenida en la secuencia S_2.

Esta definición permite determinar cuándo una secuencia está contenida en otra; algo muy importante para el conteo de secuencias (cálculo del soporte).

Definición 4: Si una secuencia S_1 está contenida en una secuencia S_2, S_1 es llamada una sub-secuencia de S_2 y S_2 una super-secuencia de S_1, denotada como $S_1 \subseteq S_2$.

Definición 5 (*secuencia maximal*): Una secuencia es *maximal* si no es sub-secuencia de otra secuencia.

Por ejemplo, en la base de datos de la Tabla 1, la secuencia $\langle(k)(j)(h)\rangle$ es maximal, porque suponiendo que el soporte mínimo sea 2, no existe super-secuencia que la contenga.

Tabla 1. Base de datos de secuencias.

SID	Secuencia
1	$\langle(d, k)(k)(j, h)(k)(m)\rangle$
2	$\langle(h)(m, j)(h)\rangle$
3	$\langle(k, h)(j)(k)(h)\rangle$
4	$\langle(k)(k)\rangle$
5	$\langle(j, h)(k, m)(m, j)(d)(h)\rangle$

Esta definición es usada en los algoritmos que obtienen las secuencias maximales frecuentes para implementar métodos que determinen cuando una secuencia es maximal.

Definición 6: A la cantidad de elementos o *ítemsets* en la secuencia se le denomina tamaño de la secuencia y se denota por σ.

Por ejemplo, $\sigma(\langle(a)(b, k)\rangle)$ es una secuencia de tamaño 2.

Definición 7: La cantidad de ítems en la secuencia se le denomina longitud de la secuencia y se denota por $k = \Sigma_j|S_j|$.

Una secuencia de longitud k es llamada k-secuencia.

Por ejemplo, la longitud de la secuencia $\langle(a)(b, k)\rangle$ es 3 y se denomina como: 3-secuencias.

Definición 8: El soporte de una secuencia S es la cantidad de secuencias en la base de datos que contienen esa secuencia y se representa como, $min\_sup(S)$.

Por ejemplo, el soporte de la secuencia $S = \langle(k)(k)\rangle$ en la base de datos de la Tabla 1 es $min\_sup(S) = 3$ dado que la secuencia está contenida en las secuencias 1, 3 y 4.

Esta definición es muy importe, ya que el soporte de una secuencia es lo que determina si una secuencia es frecuente o no.

Definición 9: Una secuencia es frecuente si el soporte de esta es mayor o igual que uno especificado por el usuario.

Por ejemplo, dada la base datos anterior. Si se especifica como soporte el 2 (significa que debe existir al menos dos secuencias que la contengan), se puede decir que la secuencia $\langle(k)(k)\rangle$ es frecuente porque su soporte es 3.

Definición 10 (*secuencia cerrada*): Una secuencia se dice que es cerrada si no existe una super-secuencia S con el mismo soporte.

Por ejemplo, en la base de datos de la Tabla 1 la secuencia $\langle(h)(m)\rangle$ es cerrada porque a pesar de existir dos super-secuencias $\langle(h)(m)(h)\rangle$ y $\langle(h)(m,\ j)\rangle$ su soporte es distinto.

Soporte de la secuencia $\langle(h)(m)\rangle$ es igual a 3, aparece en las secuencias 1, 2 y 5.

Soporte de las secuencias $\langle(h)(m)(h)\rangle$ y $\langle(h)(m,\ j)\rangle$ es 2, ambas aparecen en la secuencia 2 y 5.

Al igual que para la definición de secuencia maximal, la definición de secuencia cerrada es implementada en los algoritmos para determinar cuándo una secuencia es cerrada.

Definición 11 (*prefijo*): Dada una secuencia $\alpha = \langle e_1\ e_2...e_n\rangle$, una secuencia $\beta = \langle e'_1\ e'_2...e'_m\rangle$ $(m \leq n)$ se dice que es prefijo de α si y solo si (1) $e'_i = e_i$ para $(i \leq m-1)$; (2) $e'_m \subseteq e_m$; y (3) todos los ítems en $(e_m - e'_m)$ están alfabéticamente después de e'_m.

Por ejemplo, las secuencias $\langle(j)\rangle$, $\langle(j)(k)\rangle$, $\langle(j)(k,h)\rangle$ son prefijos de la secuencia $\langle(j)(k,h)(a,b)(d)\rangle$.

Definición 12 (*proyección*): Dadas las secuencias α y β tal que β es una sub-secuencia de α. Una sub-secuencia α' de una secuencia α se dice que es una proyección de α con respecto al prefijo β si y solo si (1) α' tiene prefijo β; (2) no existe una super-secuencia α'' de α', tal que α'' es una sub-secuencia de α y prefijo β.

Esta definición junto con la anterior y las dos siguientes son muy importantes para los algoritmos que realizan proyección de bases de datos. Ejemplo de esto es el algoritmo PrefixSpan que se analizará más adelante en el epígrafe 3.2.1.

Definición 13 (*postfijo*): Sea una secuencia $\alpha' = \langle e_1\ e_2...e_n\rangle$ la proyección de α con respecto al prefijo $\beta = \langle e_1\ e_2...e_{m-1}\ e'_m\rangle$ $(m \leq n)$. La secuencia $\gamma = \langle e'_m\ e_{m+1}...e_n\rangle$ es llamada postfijo de α con respecto al prefijo β y se denota por $\gamma = \alpha/\beta$, donde $e'_m = (e_m - e'_m)$. También se denota como $\alpha = \beta\Delta\gamma$.

Por ejemplo, la secuencia $\langle(k,\ h)(a,\ b)(d)\rangle$ es el postfijo de la secuencia $\langle(j)(k,\ h)(a,\ b)(d)\rangle$ con respecto al prefijo $\langle j\rangle$.

Definición 14 (*proyección de bases de datos*): Dado un patrón secuencial α en una base de datos de secuencias BD. La proyección de la base de datos α, denotada como $DB|_\alpha$, es la colección de postfijos de secuencias en BD con respecto al prefijo α.

Por ejemplo, la proyección de la base de datos del patrón secuencial $\langle j\rangle$ en la base de datos de la Tabla 1 se puede observar en la Tabla 2.

Tabla 2. Base de datos proyectada.

SID	Secuencia
1	$\langle(\_,\ h)(k)(m)\rangle$
2	$\langle(\_)(h)\rangle$
3	$\langle(\_)(k)(h)\rangle$
5	$\langle(\_,\ h)(k,\ m)(m,\ j)(d)(h)\rangle$

2.2 Estrategia basada en la generación de candidatos

La generación de candidatos o apriori, como también se le suele llamar, es una de las estrategias usadas en los algoritmos de minado de secuencias frecuentes; y se basa, de forma general, en obtener nuevas secuencias candidatas a partir de patrones secuenciales más simples.

Los algoritmos ([26] [27]) que usan la estrategia apriori generan en cada iteración un conjunto con todas las secuencias candidatas a partir de un conjunto base y, determinan cuáles de estas secuencias son frecuentes. Este proceso se realiza de forma repetitiva conociendo que las secuencias frecuentes obtenidas

en una iteración servirán de base para la generación del conjunto de secuencias candidatas de la siguiente. El proceso termina cuando el nuevo conjunto de secuencias candidatas está vacío (ver algoritmo 1). El conjunto base inicial es el que contiene todos los ítems frecuentes.

Algorithm 1: Pasos generales de la estrategia apriori.

```
L_1 = { ítems frecuentes };
K = 2;
while L_{k-1} no este vacío do
    C_k = nuevo conjunto de secuencias candidatas generado desde L_{k-1};
    foreach secuencia S en BDs do
        if secuencia en C_k es subsecuencia de S then
            incrementa el soporte de la secuencia en C_k;
        end
        L_k = secuencia en C_k que cumplen con soporte mínimo;
    end
    resultado = ⋃_k L_k;
end
```

El conjunto de secuencias frecuentes se determina al realizar un recorrido sobre la base de datos y realizar el conteo de cada una de las secuencias candidatas. Al conjunto de secuencias frecuentes pertenecerán todas aquellas secuencias candidatas cuyo soporte es mayor o igual que el especificado por el usuario.

Los procesos más costosos que presentan los algoritmos que se basan en la estrategia apriori son: la generación del conjunto de secuencias candidatas, proceso donde el número de secuencias que se generan tiende a ser bastante elevado y el conteo que se realiza de las secuencias para determinar el soporte de la misma. Por esto, los algoritmos que se desarrollan tratan de centrarse en mejorar estos aspectos. También resultan un poco ineficientes cuando existen patrones secuenciales de gran longitud. Por ejemplo, si el conjunto de ítems frecuentes está compuesto por 50 elementos, el conjunto de secuencias candidatas generadas de longitud 2 tendrá 3725 secuencias. Este número puede crecer bastante en dependencia de la cantidad de ítems frecuentes (para 500 ítems frecuentes el número de secuencias candidatas es de 374750).

2.3 Estrategia basada en el crecimiento de patrones

El crecimiento de patrones es otra de las estrategias usadas en los algoritmos de detección de secuencias frecuentes. Estos algoritmos ([28]) van proyectando, de forma recursiva, las bases de secuencias correspondientes a un prefijo o patrón. Inicialmente los patrones secuenciales son los ítems frecuentes.

Estos algoritmos, al dividir el espacio de búsqueda (bases proyectadas), se centran en sub-espacios potenciales más pequeños. Una base de datos proyectada contiene la información necesaria para el minado de patrones secuenciales que pueden crecer desde una secuencias α. A medida que crecen los patrones las bases proyectadas se hacen más pequeñas, lo que hace más fácil el minado de las secuencias frecuentes (ver algoritmo 2).

Las primeras secuencias a las que se le aplicará el crecimiento del patrón y se le proyectaran las bases son las correspondientes a los ítems frecuentes.

Hay algoritmos que después de obtener el conjunto de los ítems frecuentes no proyectan las bases de secuencias correspondientes a cada patrón, si no que hayan el conjunto de las secuencias frecuentes de longitud dos (2-secuencias).

El problema fundamental aquí, y por lo que se diferencian estos algoritmos, es relacionado con la proyección de las bases; cada algoritmo que se basa en esta estrategia desarrolla una nueva forma de

Algorithm 2: Pasos generales de los algoritmos que se basan en el crecimiento de patrones.

$L_1 = \{$ ítems frecuentes (prefijos) $\}$;
$BDP = \{$ Proyectar las bases correspondientes a cada ítems en $L_1 \}$;
foreach *baseBenBDP* **do**
    $C_k = \{$ ítems frecuentes en $B\}$;
    foreach ítem I en C_k **do**
        $prefijo =$ *ítems_correspodiente_en* $L_1 + I$;
        SubrutinaPBD(*prefijo*);
    end
    $L_k = \{$*secuencias en* C_k *que cumplen con el soporte mínimo*$\}$;
end
Resultado = $\bigcup_k L_k$

Algorithm 3: Función **SubrutinaPBD**.

$BDP = \{Proyectar la base correspondiente al prefijo\}$;
$L_1 = \{$*ítems frecuentes (BDP)*$\}$;
foreach ítem $I en L_1$ **do**
    $Prefijo_1 = prefijo +$ *ítems_correspodiente_en* L_1;
    SubrutinaPBD(*prefijo*);
end
Resultado = $\bigcup_k Prefijo\_1_k$

realizar el crecimiento del patrón y de proyectar las bases. Otro problema es que las bases proyectadas pueden llegar a ser, en cantidad de secuencias, más grandes que la base de datos original.

La mayoría de los problemas que presentan estos algoritmos son los relacionados con el crecimiento del patrón y la proyección de las bases de datos. En el caso de los dos algoritmos que calculan secuencias cerradas, estos presentan otro gran problema y es precisamente el de determinar cuándo una secuencia es cerrada.

Entre las ventajas de estos algoritmos se pueden mencionar que se centran más en el resultado ya que parten de un patrón inicial (ítems frecuentes) el cual va creciendo, lo que trae consigo que nunca se analicen secuencias que no sean frecuentes. Además, algo a destacar, es que en la medida que los prefijos van aumentando las bases de datos proyectadas van disminuyendo en tamaño; no como sucede con los algoritmos que usan la estrategia apriori que siempre que tenga una secuencia candidata vuelven a recorrer toda la base.

2.4 Estrategia seguida por los algoritmos para bases dinámicas

Los algoritmos de detección de secuencias frecuentes en datos incrementales son aquellos que trabajan sobre bases de datos que son actualizadas sistemáticamente (bases de datos dinámicas). Las actualizaciones pueden ser por inserción, modificación o eliminación de los datos.

Estos algoritmos surgen como solución al problema de ejecutar un algoritmo de detección de secuencias frecuentes cada vez que ocurra una actualización en los datos, proceso que sería muy costoso computacionalmente.

El trabajar con bases de datos dinámicas hace más difícil el proceso de minado ya que las secuencias frecuentes pueden variar con cualquier actualización que se realice. Una solución a este problema sería obtener las secuencias frecuentes cada vez que se hace una nueva actualización, proceso que sería muy costoso ya que se tendría que comenzar todo desde el inicio.

Debido a lo antes expuesto los algoritmos que trabajan sobre bases de datos dinámicas obtienen en un primer paso las secuencias frecuentes y las almacenan en una estructura. Luego, cada vez que se realizan actualizaciones a la base de datos el algoritmo determina las secuencias frecuentes que están en las secuencias que serán adicionadas a la base y hace un análisis entre este último conjunto de secuencias frecuentes y el que tiene almacenado. El conjunto resultante del análisis anterior será el nuevo conjunto de secuencias frecuentes y la nueva estructura a guardar (ver algoritmo 4).

Algorithm 4: Pasos generales que siguen los algoritmos que trabajan sobre bases de datos dinámicas.

SF = Ejecutar algoritmo para obtener las SF de la base de datos. Ejemplo: CloSpan o PrefixSpan;
foreach nueva modificación en la BD **do**
    Obtener SF_1 (*secuencias frecuentes de las nuevas secuencias*);
    SF_2 = *Analizar nuevas* SF_1 *con las SF almacenadas*;
    $SF = SF_2$;
end

Hay algoritmos que solo trabajan con las operaciones de adición o inserción de datos, debido al problema que representan para el minado de patrones secuenciales las operaciones de modificación y eliminación. Otro problema al que se enfrentan estos algoritmos es a la hora de determinar cuándo una secuencia es frecuente. Generalmente el proceso que se sigue es hacer un análisis entre las secuencias frecuentes que se tienen almacenadas y las nuevas secuencias frecuentes; en este proceso pueden ocurrir cuatro casos (ver Tabla 3).

Tabla 3. Casos que pueden ocurrir en el análisis entre el nuevo conjunto de secuencias frecuentes y el que se tiene almacenado.

Conjunto de secuencias frecuentes	BD Secuencias
Frecuente	Frecuente
No frecuente	No frecuente
Frecuente	No frecuente
No frecuente	Frecuente

Como se puede observar hay un caso muy complejo de analizar que es cuando una secuencia no es frecuente, y por la tanto no se tiene almacena en el conjunto de secuencias frecuentes pero esa misma secuencia es frecuente en el nuevo conjunto de secuencias que se agregaran a la base.

Los algoritmos que trabajan sobre bases de datos dinámicas se diferencian en el proceso de análisis del conjunto de secuencias frecuentes que se tiene almacenado y el nuevo conjunto de secuencias que modificarán la base.

2.5 Métodos para la proyección de las bases de datos

Los métodos de proyección de bases de datos son usados por los algoritmos de minado de secuencias frecuentes que usan la estrategia de crecimiento de patrones y estos se basan, precisamente, en proyectar para cada nuevo patrón que se forma, la base de datos correspondiente. El uso de estos métodos, el algoritmo PrefixSpan [29] hace uso de dos de ellos, está fuertemente relacionado con la capacidad de la memoria principal.

En el siguiente análisis se verán dos métodos de proyección de bases de datos:

- *bi-level projection*, es usado cuando no es suficiente el espacio existente en memoria para proyectar las bases de datos.

- *pseudo proyección* (*pseudo-projection*) es usado cuando existe suficiente espacio en la memoria principal para proyectar las bases de datos de los patrones secuenciales.

2.5.1 Método bi-level projection

El método *bi-level projection* reduce el número y el tamaño de las bases de datos proyectadas por lo que es utilizado cuando las bases de datos son muy grandes para la memoria disponible. En lugar de proyectar la base de datos se crea una matriz triangular donde aparecerán los soportes correspondientes a cada secuencia formada. En la Tabla 4 se muestra un ejemplo.

Tabla 4. Matriz correspondiente a una base proyectada.

b	0			
j	(2, 0, 0)	0		
k	(2, 0, 0)	(0, 0, 3)	0	
h	(0, 0, 0)	(1, 0, 0)	(1, 0, 0)	0
	b	j	k	h

Como se puede observar las celdas de la matriz están formadas por tres números que representan el soporte de tres secuencias; el primero representa el soporte de la secuencia formada por ⟨ *ítem_columna ítem_fila* ⟩ (e.g. $\langle(b)(k)\rangle$), el segundo a la secuencia ⟨ *ítem_fila ítem_columna* ⟩ (e.g. $\langle(k)(b)\rangle$) y el tercero a ⟨ (*ítem_columna ítem_fila*) ⟩ (e.g. $\langle(b, k)\rangle$). La diagonal principal representa a las secuencias formadas por el *ítem_columna* y el *ítem_fila* (e.g. $\langle(k, k)\rangle$).

Lo anteriormente expuesto es para las secuencias de longitud dos (2-*secuencias*). Ahora bien, cuando se hacen las proyecciones de las bases de datos para secuencias de longitud k, el procedimiento es diferente. Primero se deben seleccionar los ítems frecuentes existentes en la base de datos proyectada que se está analizando y ubicar estos en una matriz semejante a la anterior. Los que cumplan con el soporte se agregan al patrón que se analiza.

Por ejemplo, se tiene el patrón secuencial $\langle(b)(k)\rangle$ y su base de datos proyectada.

Tabla 5. Matriz correspondiente a la base proyectada correspondiente al prefijo $\langle(b)\ (k)\rangle$.

m	0			
j	(1, 0, 0)	2		
i	(2, 0, 0)	(0, 0, 1)	0	
(_h)	(0, 2, 0)	(1, 0, 0)	(1, 0, 0)	0
	m	j	i	(_h)

Según la figura 5 existen dos secuencias frecuentes $\langle(m)(i)\rangle$ y $\langle(\_h)(m)\rangle$ estas se concatenan con la secuencia ($\langle(b)(k)\rangle$) del patrón secuencial para formar los patrones: $\langle(b)(k)(m)(i)\rangle$ y $\langle(b)(k, h)(m)\rangle$.

*2.5.2 Método pseudo proyección (*pseudo-projection*)*

El método *pseudo-projection* propuesto por PrefixSpan [29], es utilizado para el caso que no exista espacio suficiente en la memoria principal para las bases de datos proyectadas y no se desea hacer una proyección física de los datos. Cada proyección tiene asociada dos tipos de información, un puntero a la secuencia en la base de datos y el offset del postfijo.

Por ejemplo, suponiendo que la siguiente base de datos se puede cargar completamente en la memoria principal.

La proyección de la base de datos asociada al prefijo o secuencia $\langle b\rangle$ serian las secuencias $\langle(d,\ j,\ k)\rangle$ y $\langle(j, k)(d, i, m)\rangle$; en vez de realizar esto, el método de pseudo-proyección permite representar los sufijos de las secuencias proyectadas usando punteros a las secuencias y los offset correspondientes.

Tabla 6. Base de datos de secuencias.

SID	Secuencia
1	$\langle(a, b)(d, j, k)\rangle$
2	$\langle(d, j, k)(h)\rangle$
3	$\langle(d, j, k)\rangle$
4	$\langle(b)(j, k)(d, i, m)\rangle$

Tabla 7. Base de datos proyectada $\langle b\rangle$.

id-secuencia	offset
1	3
4	2

El offset es la posición dentro de la secuencia donde comienza el sufijo. Por ejemplo, en la primera secuencia de la base de datos que aparece en la Tabla 6, para el prefijo $\langle(a, b)(d)\rangle$ el offset correspondiente sería el 4.

El método de pseudo proyección es más eficiente en tiempo y espacio ya que no realiza una copia física de los postfijos de las secuencias.

2.6 Generalizando el proceso de minado de secuencias frecuentes

El primero que aborda este tema es Agrawal [20], donde da una explicación bastante detallada del problema. Esta generalización del problema permite clasificar los algoritmos según el método que usan para la detección de las secuencias. Si en el proceso de detección trabajan con crecimiento de patrones [29] [30] [31] [32] o generando conjuntos de secuencias candidatas [1] [20] [24]. En estos tipos de algoritmos existen otras características que los diferencian entre si, como son el uso de ventanas de tiempo deslizante, límites de tiempo o GAP [33] y taxonomías.

2.6.1 Límite de tiempo o GAP

Es una condición de que deben cumplir dos elementos de un patrón secuencial, restringe el tiempo entre dos elementos adyacentes. Es un intervalo de tiempo Esta condición, como se verá más adelante, puede representar además de un tiempo, una cantidad de ocurrencias determinada.

Por ejemplo, en un club de lectores se registran diariamente todas las ventas de libros realizadas a sus clientes. Esto permite a la administración poder realizar análisis detallados de sus ventas logrando resultados de gran interés. Asimismo, si un cliente compra el *Código Da Vinci* y tres años después *Ángeles y Demonios*, esto puede ser una secuencia frecuente en la venta de estos libros pero a la administración puede no serle de importancia por el tiempo transcurrido entre una operación (elemento) de compra y otra (ver figura 1).

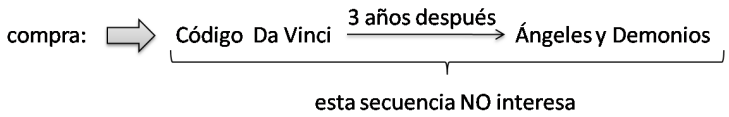

Fig. 1. Secuencia frecuente sin límite de tiempo.

Si lo que realmente le interesa a la administración son las secuencias frecuentes que tengan un límite de tiempo entre la ocurrencia de una compra y otra, entonces se establece un límite de tiempo que deben cumplir todos los elementos para que se puedan considerar parte de la secuencia frecuente (ver figura 2).

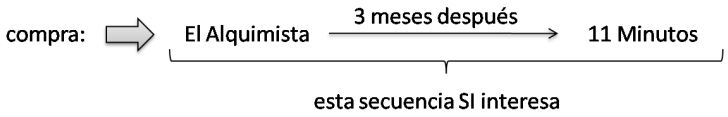

Fig. 2. Secuencia frecuente con límite de tiempo de 3 meses.

En el caso anterior se muestra la secuencia de compra de otro cliente pero donde el intervalo entre la compra del primer libro y el segundo cumple con la condición establecida.

En el ejemplo anterior se uso GAP como un límite o restricción de tiempo. Ahora se usará como una restricción en la separación entre las palabras de un texto que forman una secuencia frecuente.

Por ejemplo, se tiene un texto y se desea determinar las secuencias frecuentes que aparecen en el mismo, con un umbral igual a tres ($\beta = 3$).

En el texto que se analizará aparecen estos cinco fragmentos:

1. ...los **turistas** opinan que **Cuba** es un lindo país... $item_n$
2. ...los **turistas** visitan **Cuba** porque es un lindo país... $item_n$
3. ...Cuba es un lindo país por la su naturaleza y la belleza de su gente... $item_n$
4. ...a los **turistas** les gusta venir de visita a **Cuba**... $item_n$
5. ...los **turistas** que aman **Cuba**, dicen que es un lindo país... $item_n$

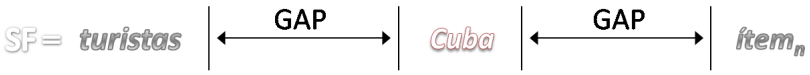

Fig. 3. Significado en el uso de GAP.

SFM sin GAP

- los turistas Cuba es un lindo país

SFM con GAP = 0

- los turistas
- es un lindo país

SFM con GAP = 3

- los turistas Cuba es un lindo país

En el ejemplo anterior se uso GAP como límite de la cantidad de palabras que pueden existir entre dos ítems (palabras) para pertenecer a una misma secuencia frecuente.

2.6.2 Ventana de tiempo deslizante

Define un tiempo que permite que dos ítems en transacciones adyacentes formen parte de un mismo elemento de un patrón secuencial, siempre que los tiempos de estas transacciones estén dentro de la ventana de tiempo.

Cada elemento del patrón puede estar contenido en la unión de los ítems pertenecientes a varias transacciones siempre que la diferencia entre los tiempos de las transacciones sea menor que el tamaño de la ventana.

Cuando se va a comparar una secuencia S asumida como patrón con las secuencias de la base de datos, la ventana de tiempo permite que los elementos en si puedan ser agrupados con respecto a la ventana de tiempo.

Por ejemplo, se tiene una base de datos donde una de las transacciones del cliente es la de la figura 4.

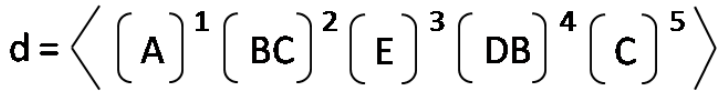

Fig. 4. Secuencia de ejemplo donde cada transacción ocurre en un día distinto.

Cada transacción de la secuencia anterior (figura 4) ocurre en un día diferente; entonces, si se toma como 3 el tamaño de la ventana de tiempo, se puede decir que la secuencia $d_1 = \langle (ABCE)(DBC) \rangle$ está incluida en la secuencia d (ver figura 5).

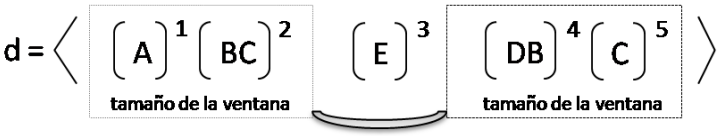

Fig. 5. Secuencia original a la que se le aplica una ventana de tiempo de un tamaño de tres días.

El tamaño de la ventana hace posible unir elementos adyacentes siempre que se cumpla con el tamaño de la ventana. Esto permite unir los elementos (A) con (BC) y (DB) con (C) para formar (ABC) y (DBC). El elemento E se puede, en este caso, unir con cualquiera de estos dos últimos elementos, ya que el tamaño de la ventana lo permite. Por lo que además de estar d incluida en la base de datos, resultado de unir (A), (BC) con (E) y (DB) con (C) para formar $\langle (ABCE)(DBC) \rangle$; se puede unir E con (DBC) para formar $\langle (ABC)(EDBC) \rangle$.

Por ejemplo, si en un mercado se toma un tamaño de ventana de 1 semana, un cliente que compre un *Pollo Congelado* el sábado, el jueves unos *Tomates* y 15 días más tarde compra de una vez *Chorizo* y *Jamón*; soporta el patrón $\langle (\textit{Pollo Congelado Tomates})(\textit{Chorizo Jamón}) \rangle$.

Como se vio la ventana de tiempo nos permite lograr otra forma de determinar secuencias frecuentes partiendo de unir elementos adyacentes siempre que el tamaño de la ventana lo permita.

2.6.3 Taxonomía

Es el uso de taxonomías para determinar patrones de secuencias frecuentes. La taxonomía permite generalizar asociaciones entre los diferentes ítems. Por ejemplo, una taxonomía nos puede decir que *un tomate es una verdura y a su vez un alimento*.

En muchas bases de datos se tiene definida una jerarquía (taxonomía) sobre los datos y esta permite encontrar patrones que incluyen a los diferentes niveles de la taxonomía.

Se dice que una transacción contiene un ítem X∈I si X está en T o X es un ancestro de algún ítem en T.

Por ejemplo, se tiene, de nuevo, el club de lectores; ahora la administración desea saber las secuencias frecuentes de las ventas de los libros pero partiendo de una taxonomía.

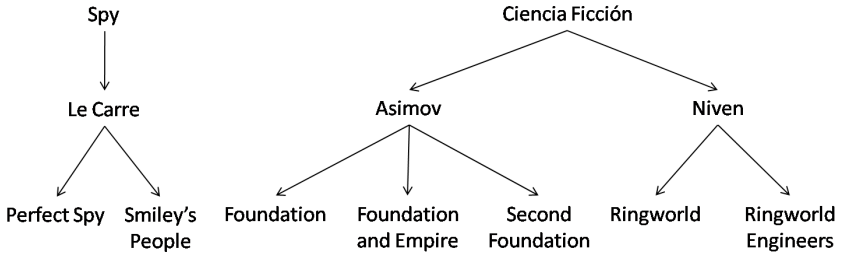

Fig. 6. Parte de una taxonomía hecha a los libros de un club de lectores.

Una secuencia frecuente puede ser la que se muestra en la figura 7.

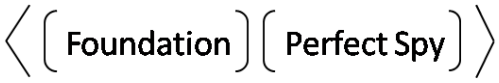

Fig. 7. Secuencia frecuente sin uso de taxonomías.

Si se usa el patrón de taxonomía de la figura 6 se pueden tener, además, los patrones de la figura .

Fig. 8. Secuencias frecuentes usando la taxonomía de la figura 6.

El análisis de las secuencias frecuentes usando un patrón de taxonomía nos permite tener un mejor conocimiento generalizado de las secuencias frecuentes.

3 Algoritmos de detección de secuencias en bases de datos estáticas

En este capítulo se expondrán las características fundamentales de varios de los algoritmos que minan patrones secuenciales.

Los algoritmos aparecerán divididos en epígrafes según la característica que más lo distingue. En los dos primeros epígrafes estarán los algoritmos agrupados según la estrategia en que se basan, en el primer epígrafe estarán los algoritmos basados en apriori y en el segundo los que se basan en el crecimiento de patrones.

En el tercer epígrafe estarán los algoritmos que determinan el conjunto de las secuencias cerradas; y por último los algoritmos que calculan el conjunto de patrones secuenciales en bases de datos dinámicas.

3.1 Algoritmos basados en apriori

En este capítulo se expondrán las características de algunos de los algoritmos que se basan en la estrategia apriori. En la figura 9 aparece una taxonomía con los algoritmos que se expondrán.

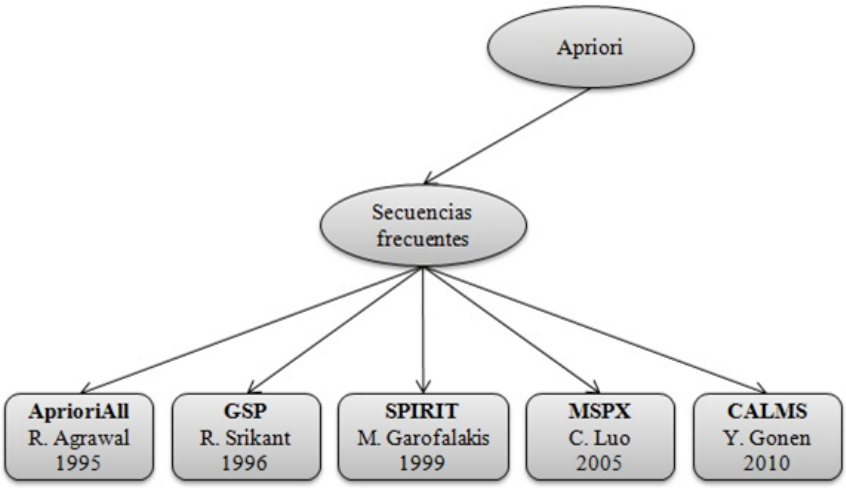

Fig. 9. Taxonomía de los algoritmos que usan bases de datos fijas y la estrategia apriori.

3.1.1 Algoritmos GSP y MSPX

Los algoritmos GSP (Generalized Sequential Patterns [20]) y MSPX [34] hacen uso de la estrategia apriori para obtener el conjunto de las secuencias frecuentes; y ambos siguen de forma general, el mismo proceso para hallar las secuencias candidatas, el cual está dividido dos fases:

Fase de enlace: es la encargada de generar el conjunto de secuencias candidatas. La generación de este conjunto es mediante la intersección del conjunto de secuencias frecuentes de la iteración anterior, con él mismo.

Fase de poda: es la encargada de determinar cuáles de las secuencias candidatas puede ser eliminada antes de realizar el recorrido de la base de datos para el conteo de las mismas.

El proceso explicado anteriormente también es usado por varios de los algoritmos de detección de secuencias frecuentes.

Los pasos usados por los algoritmos GSP [20] y MSPX [34] para la obtención del conjunto de secuencias candidatas:

Fase de enlace: las secuencias candidatas se generan al hacer una unión del conjunto L_{k-1} (conjunto de secuencias frecuentes) con él mismo. Una secuencia S_1 se une con una secuencia S_2, si luego de eliminar el primer ítem de S1 es igual a S_2 después de haberle eliminado el último ítem. La secuencia candidata es la formada por la secuencia S_1 adicionándole el último ítem de S_2. Este ítem se adiciona como un nuevo elemento si está como un elemento separado de S_2 y como parte del último elemento en caso contrario.

Fase de poda: eliminar del conjunto de secuencias candidatas aquellas secuencias que tengan una sub-secuencia continua que no aparezca entre las secuencias frecuentes del conjunto L_{k-1}.

La Tabla 8 muestra las secuencias frecuentes y las obtenidas después de la fase de enlace y de la fase de poda.

Tabla 8. Proceso de obtención de secuencias candidatas y de poda.

3-secuencias frecuentes	Secuencias, después de la fase de enlace	Secuencias, después de la fase de poda
$\langle(b)(d,e)\rangle$	$\langle(b,c)(d,e)\rangle$	$\langle(b,c)(d,e)\rangle$
$\langle(c)(d,e)\rangle$	$\langle(b,c)(d)(f)\rangle$	
$\langle(b,c)(d)\rangle$		
$\langle(b,c)(e)\rangle$		
$\langle(b)(d)(e)\rangle$		
$\langle(b)(d)(f)\rangle$		

En la Tabla 8 se puede observar que en la fase de poda se elimina la secuencia $\langle(b,c)(d)(f)\rangle$ porque no cumple con la condición que se plantea, ya que la sub-secuencia $\langle(c)(d)(f)\rangle$ no aparece dentro del conjunto de secuencias frecuentes.

A pesar de las semejanzas que presentan estos algoritmos (GSP [20] y MSPX [34]), ambos presentan diferencias cuando están hallando el conjunto de secuencias frecuentes de cada iteración.

- GSP (2) recorre la base de datos haciendo un conteo para cada secuencia en el conjunto de secuencias candidatas.
- MSPX (9) toma una base de datos aleatoria extraída de la base original y hace un conteo, sobre esta base, para todas las secuencias candidatas (C_k). Entonces, se selecciona un soporte mínimo θ como criterio para dividir C_k; C_k^- tendrá las secuencias con soporte menor que θ y C_k^+ el resto. Luego se hace una búsqueda sobre la base de datos original para realizar el conteo de las secuencias en C_k^- y al finalizar se eliminan de este todas las que su conteo es menor que el soporte mínimo. El conjunto de secuencias frecuentes (L_k) se forma de la unión de los conjuntos C_k^- y C_k^+.

3.1.2 Algoritmo CAMLS

CAMLS (Constraint-based Apriori Mining of Long Sequences [35]) es un algoritmo para el minado de secuencias frecuentes que hace uso de restricciones. Entre sus características principales presenta la separación en dos fases del proceso de obtención de las secuencias frecuentes, lo que provoca la introducción de un nuevo método de poda. Este algoritmo es una combinación de los algoritmos Apriori [1] y SPADE [31].

Las restricciones usadas por el algoritmo son:

Intra-event: para limitar los ítems en un evento; usa una variable (*MaxEventLength*) para limitar la cantidad de ítems en un evento.

Inter-event: para limitar la cantidad de eventos en una secuencia; usa una variable (*MaxSequenceength*) para limitar la cantidad de eventos por secuencias y otra (*MaxGap*) que define el tiempo máximo que debe existir entre dos secuencias para considerarlas frecuentes.

Como se mencionó anteriormente el algoritmo presenta dos fases nuevas que lo diferencian de los demás algoritmos de este tipo, la fase *event-wise*: el resultado de esta fase es un conjunto base con todos los eventos frecuentes que satisfacen la restricción *intra-event*; y la fase *sequence-wise*: la entrada de esta

fase es la salida de la fase anterior y su salida es el conjunto de todas las secuencias que cumplen con la restricción *inter-event*.

Mencionadas las características principales del algoritmo se pueden describir los pasos que sigue de la siguiente forma:

- Fase *event-wise*, encuentra todos los eventos frecuentes en la base de datos.
- Obtención de las secuencias frecuentes que cumplan con la restricción *MaxEventLength*.
- Fase *sequence-wise*.
- Para toda $2 \leq k \leq MaxSequenceLength$ y el conjunto de secuencia candidatas no es vacio.
  - Generar conjunto de secuencia candidatas.
  - Podar el conjunto de candidatas.
- Retornar el conjunto de todas las secuencias frecuentes.

Como todos los algoritmos del tipo apriori CAMLS también presenta las fases de generación del conjunto de secuencias candidatas y poda del mismo.

Generación de candidatos

Esta fase es una mejora de la generación de candidatos del algoritmo SPADE [31], la diferencia está en que SPADE [31] crece en un ítem a la vez en cada iteración o nivel de la generación de secuencias candidatas y en CAMLS se crece por eventos. Hay que señalar que por cada dos secuencias frecuentes se generarán dos secuencias candidatas.

Para generar las secuencias candidatas se toman dos secuencias frecuentes (S_1 y S_2) de la iteración anterior que tengan iguales prefijos de tamaño-1, entonces las dos secuencias candidatas generadas serian, S_1 concatena con el último evento de S_2 y S_2 concatenada con el último evento de S_1.

Por ejemplo, si se tienen dos secuencias frecuentes de tamaño 3 (3-*sequences*): S_1: $\langle(i,h)(j)(h,k)\rangle$ y S_2: $\langle(i,h)(j)(m)\rangle$. Las secuencias candidatas derivadas de las dos secuencias anteriores son: $\langle(i,h)(j)(h,k)(m)\rangle$ y $\langle(i,h)(j)(m)(h,k)\rangle$.

Fase de Poda

La fase de poda usada es parecida a las vistas anteriormente, es decir, una secuencia candidata es eliminada si alguna de sus sub-secuencias de *tamaño*-1 no pertenece al conjunto de frecuentes. Dado que el crecimiento de las secuencias en la generación de candidatas se hace por eventos, el algoritmo incluye otro chequeo y es que la secuencia candidata no puede tener otra sub-secuencia del mismo tamaño (generada en la misma iteración) que no sea frecuente. Por ejemplo, si se tienen dos secuencias candidatas S_1: $\langle(i,h)(j)(k)\rangle$, S_2: $\langle(h)(j)(k)\rangle$ y S_2 no es frecuente, entonces S_1 no puede ser frecuente porque S_2 es sub-secuencia de S_1.

Ventajas

Entre las ventajas se tienen, el uso de las restricciones para la obtención de secuencias candidatas y el método de generación de secuencias candidatas haciendo el crecimiento por eventos y no por ítem como la mayoría de los algoritmos del tipo apriori.

Desventajas

La desventaja fundamental del algoritmo CAMLS es la misma que presentan todos los algoritmos del tipo apriori, la generación de secuencias candidatas.

El uso de restricciones se puede considerar como desventaja si se tienen en cuenta que el algoritmo solamente puede ser usado para bases de datos donde se almacenen los tiempos de ocurrencia y duración de las secuencias. Vale aclarar que CAMLS se hizo con un objetivo específico.

3.1.3 SPIRIT, uso de expresiones regulares para la detección de secuencias frecuentes

Básicamente los algoritmos que usan la estrategia de generación de candidatos en su esqueleto son similares a la estrategia Apriori de Agrawal y Srikant [1]. SPIRIT [24] no deja de ser uno de estos y a pesar de usar expresiones regulares para determinar las secuencias frecuentes mantiene el uso de esta estrategia.

SPIRIT (Sequential Pattern mIning with Regular expressIon consTraints [24]) es una familia de algoritmos que minan las secuencias frecuentes maximales. Estos a diferencia de los algoritmos de minado de secuencias donde tradicionalmente se usa como restricción un soporte mínimo, proponen el uso de expresiones regulares (RE en inglés) como una herramienta más para especificar limitaciones.

El objetivo de los algoritmos SPIRIT es minar los patrones que no solo son frecuentes, sino también los que pertenecen al lenguaje de las secuencias generadas por una expresión regular especificada. Para esto se apoyan en la equivalencia que tienen las expresiones regulares con los autómatas finitos deterministas, que permite construir un autómata finito determinista equivalente a partir de cualquier expresión regular (ver figura 10).

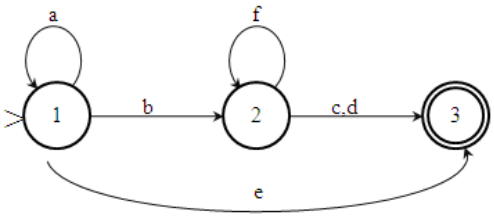

Fig. 10. Autómata para la expresión regular: a* (bc — bf*d — e).

En el autómata de la figura 10 se tiene que una secuencia S es aceptada por un autómata A_R, si siguiendo la secuencia de transición para los elementos de S desde el estado inicial se llega a un estado final o de aceptación. Por ejemplo, dada la secuencia $\langle(b,c)\rangle$ y el autómata de la figura 10, se dice que es aceptada porque partiendo del estado inicial (estado 1) con el ítem b y siguiendo las transiciones (estado 2, intermedio) se llega a uno de los estados finales (estado 3) con el último ítem de la secuencia.

A continuación se expondrán algunas definiciones que surgen del proceso que realiza el autómata para determinar si una secuencia es aceptada.

Definición 15 (*legal*): Una secuencia S es legal con respecto a un estado t de un autómata A_R, si cada estado de transición en A_R está definido cuando se sigue la secuencia de transición para los elementos de S desde t.

Por ejemplo, la secuencia $\langle(a,b,f)\rangle$ se dice que es legal con respecto al estado 1 porque desde este estado y siguiendo las transacciones se llega a un estado definido (estado 2 en este caso) con el último elemento (ítem f).

Definición 16 (*válida*): Una secuencia S es válida con respecto a un estado t de un autómata A_R, si S es legal con respecto a t y el estado final de la transiciones desde t con la entrada S es un estado de aceptación del autómata A_R. Es equivalente decir que, S es válida si es aceptada por A_R.

Por ejemplo, la secuencia $\langle(f,d)\rangle$ es válida con respecto al estado 2 porque la secuencia es legal con respecto a este estado y el estado final de las transiciones desde 2 coincide con uno de los estados de aceptación del autómata.

El esqueleto o base de los algoritmos de la familia SPIRIT recibe un parámetro de entrada C que especifica la restricción a usar en el minado de los patrones. Partiendo de esta restricción se induce a una más débil denominada C'. La selección de C' es la que diferencia a los miembros de la familia SPIRIT y su *fuerza*, el rigor con que se minan los patrones secuenciales. También, C' es utilizada en la generación de candidatos y en la poda de los mismos.

A continuación aparecerán, de forma sintetizada, los pasos que siguen los algoritmos de la familia SPIRIT:

- Encontrar, partiendo de C, una restricción C' más débil. La selección está en dependencia del algoritmo de la familia que se utilice.
- Hallar el conjunto de los ítems frecuentes y que además satisfagan C'.
- Repetir
  - Generar el conjunto (C_k) de las secuencias candidatas que satisfagan C'.
  - Podar el conjunto de secuencias candidatas.
  - Realizar el conteo de las secuencias candidatas.
- Hasta *Condición de Terminación*.
- Retornar el conjunto de todas secuencias candidatas que satisfacen C.

Los pasos entre los puntos 3 y 5 se repiten hasta que el conjunto de secuencias candidatas sea un conjunto vacio.

Algoritmos de la familia SPIRIT

SPIRIT (***Naive***)

Este algoritmo requiere que todos los elementos de una secuencia candidata aparezcan en la expresión regular. Esta restricción es la más débil de todos los algoritmos de la familia.

La generación de candidatos y la poda usadas es la misma que usan los algoritmos GSP [20] y MSPX [34] y que fue explicada en el sub-epígrafe correspondiente a esos algoritmos. El algoritmo termina si el conjunto de secuencias candidatas generadas es vacio.

SPIRIT (***Legal***)

El algoritmo requiere que cada secuencia sea legal con respecto a algún estado del autómata A_R, esta restricción es más fuerte que la del algoritmo SPIRIT(N).

El conjunto de secuencias candidatas (k-secuencias) está compuesto por todas aquellas secuencias de $longitud$-1 (deben ser prefijos o sufijos de la k-secuencia) que sean legales (ver definición) con respecto a cualquier estado y que sean potencialmente frecuentes. La poda de una secuencia se realiza si se encuentra una sub-secuencia que sea legal con respecto a algún estado y no sea frecuente. El algoritmo termina si el conjunto de secuencias candidatas que son legales con respecto a algún estado es vacio.

SPIRIT (***Valid***)

El algoritmo con este tipo de restricción solo tendrá en cuenta aquellas secuencias que sean válidas con respecto a cualquier estado de A_R. La restricción usada es más fuerte que la utilizada en SPIRIT(L).

El conjunto de secuencias candidatas (k-secuencias) está dado por todas aquellas secuencias de $longitud$-1 (deben ser sufijos de la k-secuencia) que sean válidas (ver definición) con respecto a cualquier estado del autómata y que sean potencialmente frecuentes. La poda es similar a la que realiza el algoritmo anterior con la diferencia de que en este caso se analiza si son legales. El algoritmo termina si el conjunto de secuencias candidatas es vacio.

SPIRIT (***Regular***)

Este algoritmo de la familia SPIRIT es el que pone la mayor restricción a las secuencias contando para el soporte solo aquellas secuencias válidas que sean aceptadas por A_R.

Cada algoritmo de la familia SPIRIT va aumentando su nivel de restricción sobre el conjunto de secuencias candidatas lo que implica una relación de la siguiente forma:

$$F_k^{SPIRIT(R)} \subseteq F_k^{SPIRIT(V)} \subseteq F_k^{SPIRIT(L)} \subseteq F_k^{SPIRIT(N)}$$

Ventajas

La oportunidad de especificar a través de expresiones regulares el lenguaje de las secuencias que se desean obtener, ayuda a regular el conjunto resultante, minando solo las secuencias frecuentes que sean de interés.

Desventajas

Se puede decir que la familia de los algoritmos SPIRIT presenta varias desventajas:

- La obligación en el uso de expresiones regulares para determinar el lenguaje de las secuencias frecuentes que se desean obtener, da a conocer de antemano las posibles soluciones. Algo que iría en contra de la obtención de resultados inesperados.
- La complejidad que trae aparejado el uso de expresiones regulares.

3.2 Algoritmos basados en crecimiento de patrones

En este capítulo se expondrán las características de algunos de los algoritmos que se basan en la estrategia de crecimiento de patrones. A continuación aparece una taxonomía con los algoritmos que se expondrán.

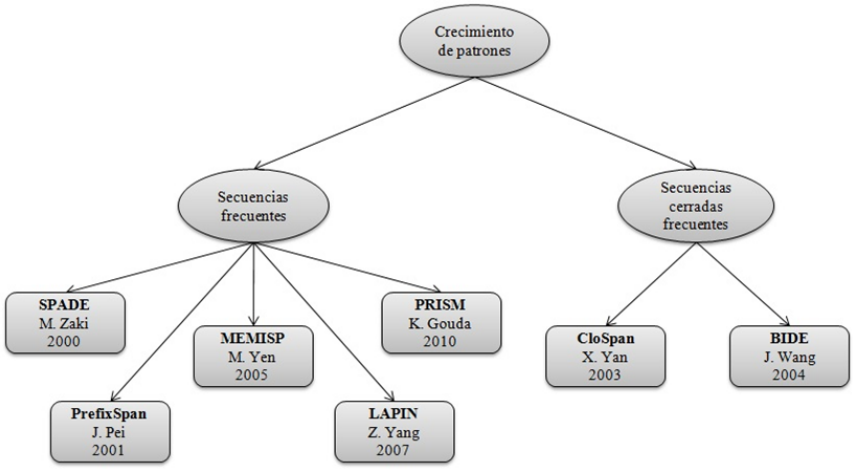

Fig. 11. Taxonomía de los algoritmos que usan bases de datos fijas y crecimiento de patrones.

3.2.1 Algoritmo PrefixSpan

PrefixSpan (Prefix-project Sequential pattern mining [29]) es un algoritmo que mina las secuencias frecuentes existentes en una base de datos y como su nombre (*Prefix-project*) lo indica, se basa en el prefijo de las secuencias para ir realizando el crecimiento del patrón mientras proyecta las bases de datos de los postfijos correspondientes. Además, crea una nueva forma de proyectar las bases de datos, el método de pseudo-proyección.

Una base de datos proyectada $S|_\alpha$ está formada por los postfijos de las secuencias en S con respecto al prefijo α.

De forma general los pasos que sigue PrefixSpan para la detección de las secuencias frecuentes son los siguientes:

- Hallar el conjunto de ítems frecuentes.
- Dividir el espacio de búsqueda en subconjuntos donde cada uno representa un patrón secuencial. En la primera iteración los patrones secuenciales son el conjunto formado por los ítems frecuentes.
- Minar los subconjuntos de patrones secuenciales construyendo las correspondientes proyecciones de las bases de datos y minando cada uno recursivamente, es decir, llamar nuevamente al mismo algoritmo pero usando como base de entrada el patrón secuencial que se analiza.

Por ejemplo, dada la base de datos de la Tabla 9.

Tabla 9. Base de datos de secuencias.

SID	Secuencia
1	$\langle(a, b)(d, j, k)\rangle$
2	$\langle(d, j, k)(h)\rangle$
3	$\langle(d, j, k)\rangle$
4	$\langle(b)(j, k)(d, i, m)\rangle$

El primer paso es determinar los ítems frecuentes para un umbral o soporte dado por el cliente. Tomando como soporte mínimo el valor 2, los ítems frecuentes serían: $\langle b\rangle$: 2, $\langle d\rangle$: 4, $\langle j\rangle$: 4, $\langle k\rangle$: 4 (aparecen de la forma, $\langle$ ítems frecuente $\rangle$: soporte). Los ítems frecuentes representan al conjunto de patrones secuenciales de longitud 1. Este conjunto (patrones secuenciales) se divide en tantos subconjuntos como patrones existan. Para el ejemplo que se está analizando el conjunto de patrones secuenciales se dividiría en 4 subconjuntos.

Luego de hallar los patrones secuenciales se proyectará una base de datos por cada patrón. En las bases de datos proyectadas aparecerán todos aquellos postfijos (sub-secuencias) que tienen como prefijo al patrón secuencial que se quiere proyectar. A continuación se muestra como quedaría la base de datos proyectada para el patrón secuencial $\langle(j)\rangle$.

Tabla 10. Base de datos proyectada con relación al prefijo $\langle(j)\rangle$ (patrón secuencial).

Prefijo	Postfijos
$\langle(j)\rangle$	$\langle(\_k)\rangle$
	$\langle(\_k)(h)\rangle$
	$\langle(\_k)\rangle$
	$\langle(\_k)(d, i, m)\rangle$

El siguiente paso sería, partiendo de los nuevos patrones secuenciales formados y de forma recursiva, llamar al algoritmo para cada uno de estos patrones (prefijos).

Como se puede observar en la figura 12, el mayor problema que presenta el algoritmo es el costo en la proyección de las bases de datos. El tamaño requerido para el almacenamiento de las bases de datos que se van proyectando puede crecer considerablemente ya que las secuencias en estas bases pueden repetirse para varios prefijos. En el peor de los casos el algoritmo puede llegar a multiplicar la cantidad de transacciones de la base de datos original por la cantidad de ítems frecuentes.

Debido al problema planteado anteriormente PrefixSpan propone el uso de dos métodos de proyecciones, *bi-level projection* y *pseudo-proyección*.

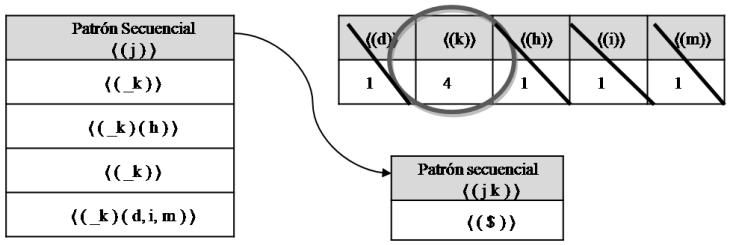

Fig. 12. Ejemplo de proyección de una base.

Ventajas

El algoritmo PrefixSpan [29] presenta varias ventajas como pueden ser: el uso de la estrategia de divide y vencerás, la forma en que realiza el crecimiento del patrón y el uso del método de pseudo-proyección para las bases de datos proyectadas.

Desventajas

El principal problema que presenta este algoritmo es la proyección de las bases de datos, de ahí los dos métodos que propone. Además, según el modo en que busca los prefijos pueden llegar a existir secuencias que no las de como frecuentes. Por ejemplo, en la base de datos de la Tabla 9 se tiene como ítem frecuente d, al tomarlo (ítem d) como prefijo, se pasa a proyectar la base de los postfijos correspondientes.

Tabla 11. Base de datos de secuencias.

SID	Secuencia
1	$\langle(a, b)(d, j, k)\rangle$
2	$\langle(j, k, d)(h)\rangle$
3	$\langle(d, j, k)\rangle$
4	$\langle(d)(j, k(d, i, m)\rangle$

En la Tabla 12 aparece la base de datos proyectada relacionada al prefijo d; si se analiza bien se puede ver que la segunda aparición del ítem d en la cuarta secuencia no se analiza, lo que trae el problema de no analizar la posibilidad de formar otras secuencias como puede ser $\langle(d, i)\rangle$.

Tabla 12. Base de datos proyectada con respecta al ítem d.

Patrón secuencial $\langle(d)\rangle$
$\langle(\_j, i)\rangle$
$\langle(h)\rangle$
$\langle(\_j, k)\rangle$
$\langle(j, k)(d, i, m)\rangle$

3.2.2 Algoritmo PRISM

PRISM (PRIme-Encoding Based Sequence Mining [36]) es un algoritmo para el minado de secuencias frecuentes que hace uso de un nuevo enfoque, el *primal block encoding*; esta metodología está basada en la factorización en números primos. El algoritmo, también hace uso de la teoría de retículo para representar todas las posibles combinaciones del conjunto base de números primos; conjunto que es usado para la obtención del *primal encoded.*

El algoritmo utiliza una codificación binaria para las posiciones de los ítems dentro de las secuencias (*bit-encoded pos*) y de las apariciones de los ítems en las secuencias (*bit-enconded sid*). Estas codificaciones son el punto de partida para el proceso de detección de secuencias frecuentes.

Bit-encoded pos

La codificación de las posiciones de los ítems dentro de las secuencias o *bit-encoded pos* se construye poniendo, de izquierda a derecha, un 1 cuando el ítem aparece en la transacción y 0 si no se encuentra. Por ejemplo, dada la base de datos de secuencias de la Tabla 13.

Tabla 13. Base de datos de secuencias.

Número	Secuencia
1	$\langle(b,\,k)(k)(j,\,h)(k)(m)\rangle$
2	$\langle(h)(m,\,j)(h)\rangle$
3	$\langle(k,\,h)(j)(k)(h)\rangle$
4	$\langle(k)(k)\rangle$
5	$\langle(j,\,h)(k,\,m)(m,\,j)(d)(h)\rangle$

El *bit-encoded pos* del ítem *k* se puede observar en la Tabla 14.

Tabla 14. *Bit-encoded pos* para el ítem *k*.

SID	Bit-encoded pos	Primal-encoded pos
1	1101,0000	{42, 1}
2	0000	{1}
3	1010	{10}
4	1100	{6}
5	0100,0000	{3, 1}

En la Tabla 14, cada *bit-encoded pos* corresponde a una codificación binaria del ítem *k* en cada secuencia de la base de datos. Por ejemplo, el *bit-encoded pos* de la secuencia $\langle(k,h)(j)(k)(h)\rangle$ (secuencia 3 en la base de datos) es *1010* debido a que el ítem *k* aparece en la primera y tercera transacción y no aparece en la segunda y cuarta.

Bit-encoded sid

El *bit-encoded sid* es la codificación de las apariciones de los ítems en las secuencias, en este caso el 1 significa la presencia del ítem en la secuencia y el 0 el caso contrario. La Tabla 15 muestra la codificación del ítem *k*.

Tabla 15. *Bit-encoded sid* para el ítem *k*.

Bit-encoded sid	Primal-encoded sid
1011,1000	{70, 2}

El *bit-encoded sid* del ítem k es *1011, 1000* dado que el 1 aparece en las posiciones 1, 3, 4 y 5, lo que indica que el ítem está presente en estas secuencias en la base de datos (ver Tabla 13).

Primal block encoding

El *primal block encoding* es el conjunto de valores que representan la codificación de cada uno de los bloques binarios de un *bit-encoded*. Por ejemplo, en la Tabla 15 el valor numérico 70 corresponde a la codificación del primer bloque binario *1011* y el número 2 al segundo bloque binario *1000*.

La codificación parte de un conjunto base de números primos ordenados ascendentemente, dígase $C = \langle 2, 3, 5, 7 \rangle$. Ahora bien, un bloque binario viene siendo un subconjunto del conjunto C. Por ejemplo, el bloque binario *1011* es el subconjunto $S = \langle 2, 5, 7 \rangle$ que representa al valor numérico 70 dado que 2 * 5 * 7 = 70. En el caso que la codificación binaria sea *0000* el valor numérico asociado es 1. Todas las posibles combinaciones de los elementos del conjunto C inducen un retículo (ver figura 13) donde la operación de intersección está dada por el máximo común divisor y la operación de unión por el mínimo común múltiplo.

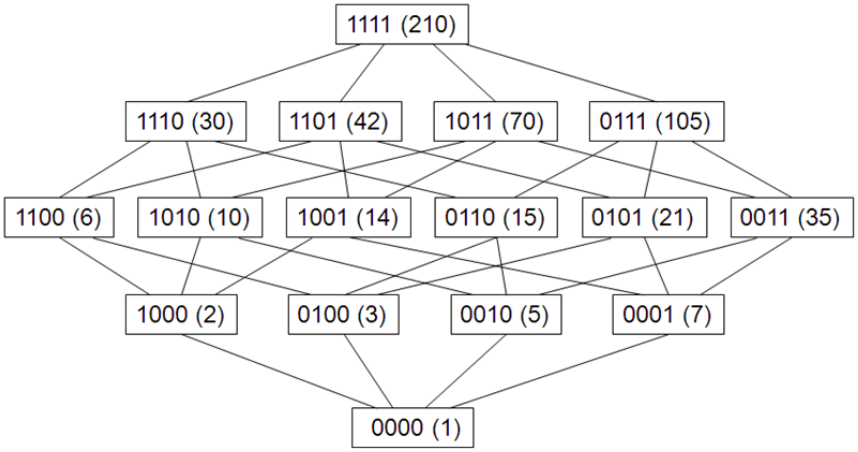

Fig. 13. Retículo formado por el conjunto de números primos ($C = \langle 2, 3, 5, 7 \rangle$).

Espacio de búsqueda

PRISM [30] usa un árbol de búsqueda donde cada nodo, que es un patrón secuencial, puede ser extendido de dos formas distintas (ver epígrafe sobre definiciones formales básicas), extendiendo el último ítemset de la secuencia (*ítem extensión*) o agregando una nueva transacción al final de la secuencia (*sequence extensión*). Debido al nuevo enfoque usado en este algoritmo el proceso de extensión de la secuencia se hace de forma diferente.

En el algoritmo cada secuencia tiene asociado dos valores, el *bit-encoded* y el *primal-encoded* por lo que cada nueva secuencia formada debe tener ambos. Estos valores se hallan de forma distinta en dependencia del tipo de extensión que se realice. Si la extensión es extendiendo el último ítemset, el *primal-encoded* se determina hallando el máximo común divisor entre los bloques del patrón secuencial y el ítem a expandir. Por ejemplo, se quiere expandir por extensión del último ítemset, la secuencia $\langle k \rangle$ (1-secuencia) con el ítem frecuente h.

Tabla 16. *Bit-encoded sid* para los ítems k, h.

Secuencia	Bit-encoded sid	Primal-encoded sid
k	1011, 1000	{70, 2}
h	1110, 1000	{30, 2}

Se toman los *primal-encoded* (ver Tabla 16) correspondientes a cada una de las secuencias y se determina el máximo común divisor.

$mcd(70, 30) = 10$	$mcd(2, 2) = 2$
1 0 1 0	1 0 0 0

Luego de calculado el máximo común divisor se obtiene el *primal-encoded* 10, 2 de la nueva secuencia $\langle(k,h)\rangle$, este resultado es necesario para determinar el *bit-encoded sid*, que se halla con el máximo común divisor entre los *primal-encoded* de los *bit-encoded pos* (ver figura 14).

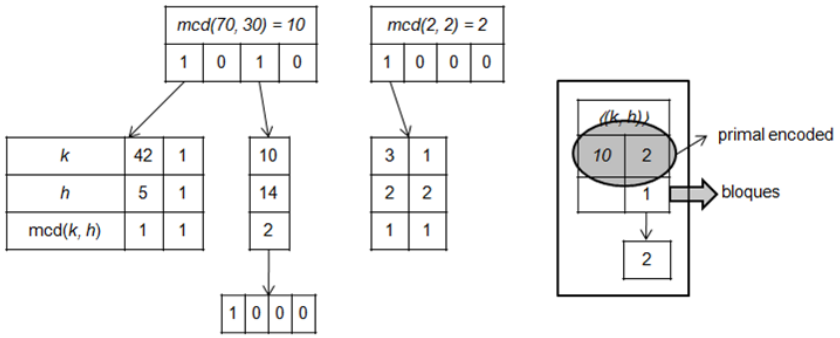

Fig. 14. Extensión del último ítemset de la secuencia (itemset extension).

En el caso de la extensión por secuencia, es decir, agregando una nueva transacción al final de la secuencia, el valor del *primal-encoded* se halla igual que la extensión del último ítemset de la secuencia (ejemplo anterior), pero aplicándole primeramente el operador de enmascaramiento (*masking operator*) a la secuencia que se extenderá. Para esto se toman los bloques binarios de la secuencia a la que se le aplicara el operador y se busca la aparición del primer 1. Luego de tener este valor, el resultado de aplicar el operador es colocar ceros mientras la posición sea menor o igual que la del primer uno y uno para el resto de las posiciones. Por ejemplo, los bloques binarios de la secuencia k son *10111000*, el resultado de aplicar el operador seria *01111111*.

En la figura 15 se muestra el proceso para el caso de la extensión por secuencia (*sequence extension*). En este caso la secuencia que se forma es $\langle(k)(h)\rangle$.

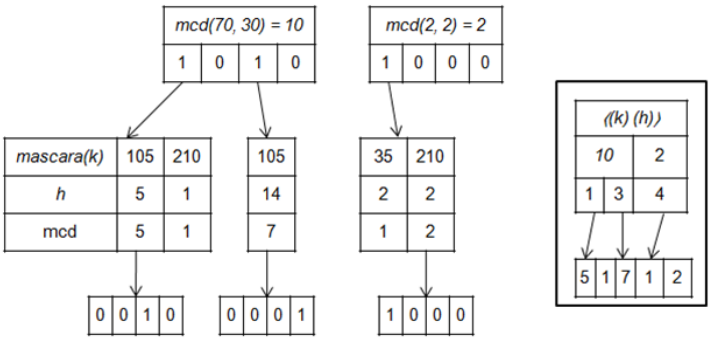

Fig. 15. Extensión por secuencia (sequence extension).

De una forma bastante breve se puede decir que el soporte de una secuencia, en PRISM, es igual a la suma de los 1 que aparecen en los diferentes bloques de las secuencias. Por ejemplo, en el caso anterior la

secuencia $\langle(k)(h)\rangle$ tiene soporte igual a 2. Además, esto nos permite saber en cuales secuencias ocurrió, en este caso en las secuencias 1 y 5 (ver Tabla 14).

Ventajas

La forma binaria de representar las posiciones dentro de las secuencias y la existencia o no de una secuencia dentro de otra es una idea muy buena; además de que debe reducirse bastante el uso de la memoria.

Desventajas

Alguno de los problemas que puede presentar el algoritmo debido a la codificación binaria con que trabaja son los siguientes:

- Si la cantidad de secuencias en la base de datos es grande pueden existir problemas a la hora de codificar las existencias de los patrones en las secuencias. Por ejemplo, el tipo de dato extended es de 10 bytes, si se multiplica este número por 8, que es la cantidad de bit que tiene un byte, da un valor de 80, y una base con 80 secuencias es una base muy pequeña.
- El tamaño de las secuencias también podría ser un problema a la hora de codificar las posiciones dentro de las mismas.
- El tiempo de procesamiento del algoritmo, a pesar de lo que dice la publicación, puede ser otro problema debido a todo el procesamiento que hay que realizar.

3.2.3 Algoritmo SPADE

SPADE (Secuential PAttern Discovery using Equivalence classes [31] [37]) es un algoritmo que mina el conjunto de todas las secuencias frecuentes, donde se incluyen las cerradas y las maximales. Entre sus características están el uso de listas de identificadores (*id_list*) con formato vertical como base de datos y el uso de la teoría de retículo para descomponer el espacio de búsqueda original en pequeños sub-espacios (sub-retículos), los cuales puedan ser procesados en memoria de forma independiente. También propone dos estrategias de búsqueda para enumerar las secuencias frecuentes dentro de cada sub-retículo: búsqueda primero en profundidad y búsqueda primero a lo ancho.

H		J		K		M	
CID	TID	CID	TID	CID	TID	CID	TID
1	20	1	15	1	15	2	15
2	20	2	10	1	10	2	30
2	30	2	15	2	15	3	40
4	30	3	40	3	40	4	35
		4	30				

Fig. 16. Ejemplo de *id_list*, listas de identificadores de los ítems frecuentes.

Si la memoria no fuera una limitante el algoritmo pudiera recorrer todo el retículo e intersecando las *id_list* de las secuencias correspondientes, determinar el soporte de cada secuencia. Como lo expuesto anteriormente no sucede con frecuencia, el algoritmo hace uso de la técnica de divide y vencerás para dividir el retículo original en pequeños sub-retículos según una relación de equivalencia. Esta relación de equivalencia divide el conjunto (retículo) en varias clases de equivalencia o sub-retículos. La relación de equivalencia está dada por: dos secuencias pertenecen a una misma clase de equivalencia si comparten un mismo prefijo de longitud k.

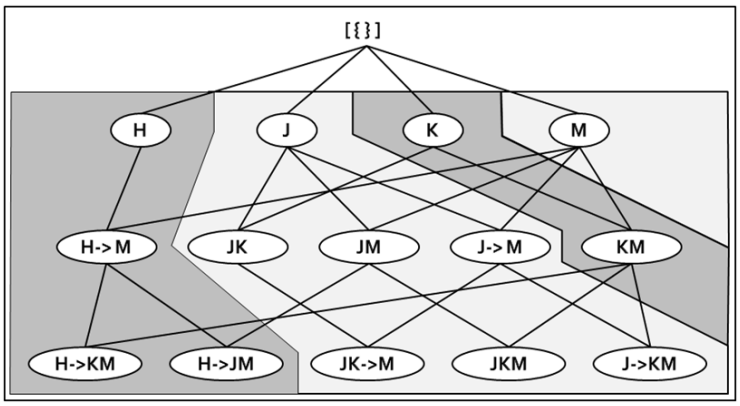

Fig. 17. Ejemplo de clases de equivalencia inducidas por la relación de equivalencia θ_1.

Las secuencias se forman por la intersección de las *id_list* de dos secuencias y su soporte está dado por la cardinalidad de la *id_list* resultante. Inicialmente se parte de las *id_list* pertenecientes al conjunto de ítems frecuentes (ver figura 16). Las *id_list* están formadas por el par CID (identificador del cliente) y TID (tiempo de transacción).

Intersecando *id_list*

Si se recuerda, la relación de equivalencia dice: dos secuencias pertenecen a la misma clase de equivalencia si ambas tienen un prefijo de longitud k.

- *Átomo ítemset vs átomo ítemset*: Si son intersecadas PK con PM, se formaría la nueva secuencia PKM.
- *Átomo ítemset vs átomo secuencia*: Si son intersecadas PK con $P \rightarrow M$, el resultado es la nueva secuencia $PK \rightarrow M$.
- *Átomo secuencia vs átomo secuencia*: Si son intersecadas $P \rightarrow K$ con $P \rightarrow M$, pueden haber tres posibles resultados: un nuevo ítemset $P \rightarrow KM$, y dos nuevas secuencias $P \rightarrow K \rightarrow M$ y $P \rightarrow M \rightarrow K$. Un caso especial es cuando se interseca $P \rightarrow K$ con ella misma, lo cual solo puede formar la secuencia $P \rightarrow K \rightarrow K$.

Luego de analizar las características principales del algoritmo se pueden presentar, de forma generalizada, los pasos que sigue:

- Hallar el conjunto de los ítems frecuentes.
- Hallar el conjunto de secuencias frecuentes de longitud 2 (2-secuencias).
- Determinar el conjunto de clases de equivalencia correspondiente a la relación de equivalencia.
- Para cada clase de equivalencia, enumerar las secuencias frecuentes.

El primer paso es obtener los ítems frecuentes para un umbral o soporte dado por el cliente y construir para cada uno su *id_list*. Luego se halla el conjunto de las secuencias frecuentes de longitud dos, para esto el algoritmo propone dos posibles soluciones ya que una simple intersección de las *id_list* de los ítems puede ser un problema muy ineficiente.

Las posibles soluciones serian:

- Hacer un conteo de las secuencias de longitud 2.
- Hacer una transformación de las listas (*id_list*) que están en forma vertical a una forma horizontal.

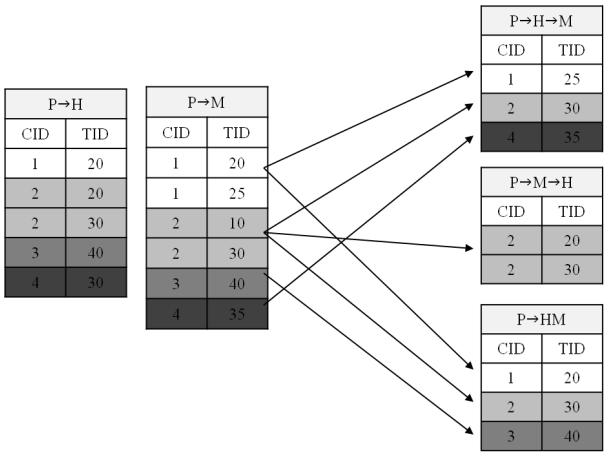

Fig. 18. Ejemplo de intersección de *id_list*.

Para la transformación horizontal se recorren los *id_list* de cada ítem y para cada par (cliente, tiempo de la transacción) se inserta en una lista el par (ítem, tiempo de la transacción), se forma una lista con todas las secuencias de longitud 2 de cada secuencia del cliente y se actualiza la cantidad en un arreglo de dos dimensiones indexado por los ítems frecuentes.

En SPADE [38] una secuencia es podada si su clase de equivalencia ha sido procesada y no pertenece al conjunto de las secuencias de longitud k-1.

Ventajas

El algoritmo SPADE limita solamente tres los recorridos a realizar en la base de datos, lo que reduce considerablemente los costos de tiempo ejecución.

Desventajas

El algoritmo no tiene en cuenta el procesamiento de secuencias donde el ítem se repita. Por ejemplo, la secuencia ($H \rightarrow KMH$) no se analiza porque en el procedimiento que sigue el algoritmo no puede estar el caso de la segunda aparición de un ítem (para nuestro ejemplo sería la segunda H). También se pierde tiempo convirtiendo la base de datos original a una vertical para luego en algunos casos hacer una conversión a una forma horizontal.

3.2.4 Algoritmo MEMISP

MEMISP (MEMory Indexing for Sequential Pattern mining [39]) es un algoritmo para minar el conjunto de secuencias frecuentes el cual usa una estrategia de *encuentra entonces indexa*.

La forma general del algoritmo es la siguiente:

- Recorre la base de datos buscando los ítems frecuentes, en caso de que la base sea muy grande el algoritmo divide esta y busca por cada una los ítems frecuentes.
- Para cada patrón secuencial actual y un ítem de extensión se construye el nuevo patrón y el conjunto de índices correspondientes.
- Usando los índices se determinan los ítems locales de una base proyectada correspondiente a un prefijo (patrón secuencial al que corresponden los índices).

Los puntos 2 y 3 se repiten para cada uno de los patrones secuenciales. Por ejemplo, dada la base de datos de la Tabla 17.

Tabla 17. Base de datos de secuencias.

SID	Secuencia
1	$\langle(d, i, k)\rangle$
2	$\langle(d, n, k)(h)\rangle$
3	$\langle(d, k)\rangle$
4	$\langle(b)(j, k)(d, h, m)\rangle$

Primero se determinan los ítems frecuentes ($\langle d\rangle$: 4, $\langle k\rangle$: 4, $\langle h\rangle$: 2) y se construyen los índices para cada patrón secuencial actual (inicialmente se parten de los ítems frecuentes, patrones secuenciales de longitud 1).

Para el patrón secuencial $\langle d\rangle$ la tabla de índices quedaría como se muestra en la figura 19.

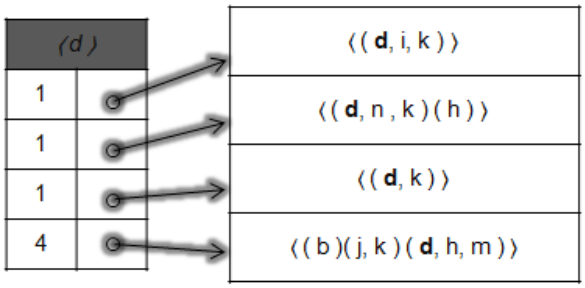

Fig. 19. Indexado para el patrón secuencial $\langle d\rangle$.

Luego se pasa a determinar los ítems locales que son frecuentes para seguir extendiendo el prefijo. En nuestro ejemplo se tendría como ítem frecuente k ($\langle krangle$: 3). El soporte de k aquí es distinto al soporte inicial.

Al nuevo patrón secuencial formado $\langle d,k\rangle$ se le determinarían los índices como aparecen en la figura 20.

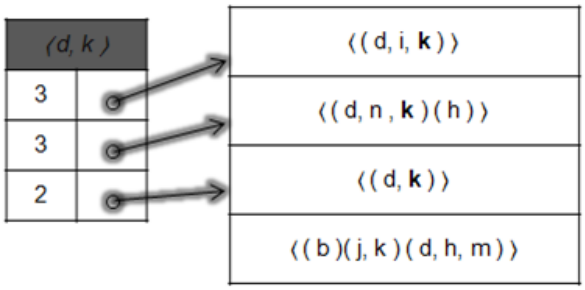

Fig. 20. Indexado para el patrón secuencial $\langle d, k\rangle$.

El proceso anterior se sigue mientras existan ítems locales frecuentes.

Ventajas

El proceso de indexado que realiza el algoritmo para el minado de secuencias frecuentes es muy bueno, además de que realiza una sola pasada sobre la base de datos.

Desventajas

En el proceso de buscar los ítems frecuentes dentro de cada segmento en los que se dividió la base de datos puede darse el caso que no encuentre un ítem como frecuente dada la distribución del mismo (ítem). Esto no ocurriría si se tomara la base como una sola, o en otro caso, sumar los resultados de cada uno de los conteos.

3.2.5 Algoritmo LAPIN

LAPIN (LAst Position INduction [40]) es un algoritmo para el minado de secuencias frecuentes que usa en su procesamiento un listado con las últimas apariciones de los ítems por secuencias. Este listado es el que le permite al algoritmo realizar el crecimiento de los patrones secuenciales.

La forma general de trabajar del algoritmo es la siguiente:

- Obtener los ítems frecuentes y crear la lista de posiciones de los ítems.
- Para cada ítems frecuente
  - Llamar al método *Generar_Patrón*.

Algoritmo Generar_Patrón:

- Encontrar el conjunto de ítems cuya posición es mayor que la del último ítem de la secuencia dada por parámetro.
- Para cada ítem del conjunto anterior
  - Combinar la secuencia dada con el ítem correspondiente.
  - Llamar al método *Generar_Patrón*.

Por ejemplo, dada un soporte mínimo igual a 2 y la base de datos de la figura 18.

Tabla 18. Base de datos de secuencias.

SID	Secuencia
10	$\langle(d)(k)(i,k)(h)(d,i,k)(d)(h)\rangle$
20	$\langle(i)(k,h)(d)(k)(i,h)\rangle$
30	$\langle(h)(i,k)(d,k)(k,h)\rangle$

Se hace una búsqueda sobre la base de datos para obtener una lista con las últimas posiciones de los ítems en las secuencias (ver Tabla 18). El listado obtenido se ordena ascendentemente.

Tabla 19. Listado de las últimas posiciones de los ítems por secuencia.

SID	Última posición del ítem en la secuencia			
10	$\rightarrow i_{last}=5$	$k_{last}=5$	$d_{last}=6$	$h_{last}=7$
20	$d_{last}=3$	$\rightarrow k_{last}=4$	$i_{last}=5$	$h_{last}=5$
30	$i_{last}=2$	$d_{last}=3$	$\rightarrow k_{last}=4$	$h_{last}=4$

Ahora bien, si se tiene como prefijo de secuencias frecuentes a $\langle d\rangle$ y sus posiciones en la base de datos (Tabla 19): 10:1, 20:3, 30:3, donde sid:eid representa la secuencia sid y el elemento eid. Se pasa a chequear la lista de posiciones para obtener el primer índice cuya posición es mayor que $\langle d\rangle$ resultando

10:1, 20:3 y 30:3 (10: i_{last}= 5, 20: k_{last}= 4, 30: k_{last}= 4). Comenzando por estos índices y hasta el final de la secuencia e incrementando el soporte de cada ítem analizado el resultado es: $\langle d \rangle$: 1, $\langle i \rangle$: 2, $\langle k \rangle$: 3, $\langle h \rangle$: 3, para lo cual se puede determinar que $\langle di \rangle$, $\langle dk \rangle$, $\langle dh \rangle$ son patrones frecuentes.

En el ejemplo anterior se uso el incremento por extensión del último ítemset de la secuencia, el proceso para la extensión por secuencia se realiza de la misma forma.

Ventajas

La ventaja más notable que tiene el algoritmo es la sencillez de la idea aplicada para la detección de secuencias frecuentes.

Desventajas

Puede llegar a ser poco ineficiente para bases de datos medianas y grandes.

3.3 Algoritmos de detección de secuencias frecuentes cerradas

Dentro del conjunto de algoritmos que minan secuencias frecuentes están los que obtienen el conjunto de las secuencias cerradas. La obtención de secuencias cerradas permite reducir el uso de la memoria ya que la cantidad de estas secuencias con respecto a las frecuentes es menor. Además, se pueden obtener todas las secuencias frecuentes a partir de las secuencias cerradas.

Una característica importante en estos algoritmos es el uso de un método que permite determinar cuándo una secuencia frecuente es cerrada.

Los algoritmos que minan las secuencias cerradas frecuentes se pueden describir de la siguiente forma:

- Hallar los ítems frecuentes (patrones secuenciales de longitud 1).
- Proyectar las bases de datos sobre los patrones secuenciales.
- Realizar el conteo de las secuencias dentro de las bases proyectadas.
- Determinar del grupo de secuencias frecuentes cuáles son cerradas.

Los algoritmos que minan este tipo de secuencias se diferencian en la forma que cada uno determina cuando una secuencia es cerrada, proceso que cada algoritmo trata de mejorar y hacer más eficiente.

3.3.1 Algoritmo BIDE

BIDE (BI-Directional Extension [41]) es un algoritmo para el minado de secuencias frecuentes cerradas que hace uso del método pseudo projection para proyectar las bases de datos. Además, utiliza un nuevo método, el *BI-Directional Extension closure checking* para determinar si una secuencia es cerrada y el método *BackScan* junto con la técnica de optimización *ScanSkip* para podar el espacio de búsqueda.

Algo muy importante a señalar, es que el algoritmo solamente analiza secuencias que estén compuestas por una sola transacción (single-items). Por ejemplo: $\langle (b,j,k,d,i,m) \rangle$, secuencia compuesta por una sola transacción. $\langle (b)(j,k)(d,i,m) \rangle$, secuencia compuesta por más de una transacción.

BI-Directional Extension closure checking

El chequeo de la extensión en dos direcciones (*BI-Directional Extension checking*) es un método creado y usado por BIDE para determinar si una secuencia es cerrada; el cual dice que una secuencia es cerrada, cuando no existe una extensión hacia delante (*forward-extension event*) o hacia atrás (*backward-extensión event*).

Extensión hacia delante (*forward-extension event*)

Existe una extensión hacia delante (*forward-extension event*) si hay un ítem frecuente local (ítem frecuente en una base de datos proyectada) que tenga el mismo soporte que el prefijo que se está analizando.

Por ejemplo, si el soporte de j (prefijo de la base de datos proyectada) es igual a 4 y existe algún ítem k local frecuente con el mismo soporte (ver la figura 21), entonces se puede decir que existe una extensión hacia delante.

BD proyectada ⟨(j)⟩
⟨(_k)⟩
⟨(_k)(h)⟩
⟨(_k)⟩
⟨(_k)(d,i,m)⟩

Ítems locales				
⟨(d)⟩	⟨(k)⟩	⟨(h)⟩	⟨(i)⟩	⟨(m)⟩
1	4	1	1	1

Fig. 21. BD proyectada del prefijo j y soporte de los ítems locales.

Extensión hacia atrás (*backward-extension event*)

Existe una extensión hacia atrás si dado un prefijo S_p ($e_1e_2...e_n$) y un valor i ($1 \leq i \leq n$) se puede encontrar un ítem e' el cual aparece en cada *i-ésimo máximo periodo* del prefijo S_p en la base de datos. A continuación aparecerán algunas de las definiciones relacionadas con el análisis del chequeo de la extensión hacia atrás.

Definición 17 (*The i-th maximum period of a prefix sequence*): Para una secuencia S con un prefijo $S_p = e_1e_2...e_n$, el *i-th* máximo periodo del prefijo Sp en S se define como: (1) si $1 < i \leq n$, este es la parte de secuencia entre el final de la primera instancia del prefijo $e_1e_2...e_{i-1}$ en S y el *i-th last-in-last* aparición con respecto al prefijo S_p; (2) si $i = 1$, este es la parte de la secuencia S localizada antes de la *1st last-in-last* aparición con respecto al prefijo S_p.

Definición 18 (*The i-th last-in-last appearance w.r.t a prefix sequence*): Para una secuencia S con un prefijo $S_p = e_1e_2...e_n$, la *i-th last-in-last* aparición con respecto al prefijo S_p en S es denota como LL_i y definida recursivamente como: (1) si $i = n$, esta es la última aparición de c en la última instancia de el prefijo S_p en S; (2) si $1 \leq i < n$, es la última aparición de e_i en la última instancia del prefijo S_p en S mientras LL_i puede aparecer antes de LL_{i+1}.

Definición 19 (*Last instance of a prefix sequence*): Dada una secuencia S que contiene un prefijo $e_1e_2...e_i$, la última instancia del prefijo en S es la sub-secuencia desde el comienzo de S hasta la última aparición del ítem e_i en S.

Poda del espacio de búsqueda

Todos los algoritmos de detección de secuencias frecuentes podan el espacio de búsqueda para eliminar las secuencias que no son necesarias analizar. BIDE [41] crea un nuevo método y una técnica de optimización denominados *BackScan* y *ScanSkip* respectivamente.

El método *BackScan* poda el espacio de búsqueda si dado un prefijo S_p ($e_1e_2...e_n$) y un valor i ($1 \leq i \leq n$) se puede encontrar un ítem e' el cual aparece en cada *i-ésimo* semi-máximo periodo del prefijo S_p en la base de datos.

A continuación aparecerán algunas de las definiciones relacionadas con el análisis del método de poda *BackScan*.

Definición 20 (*The i-th semi-maximum period of a prefix sequence*): Para una secuencia S con un prefijo $S_p = e_1e_2...e_n$, el *i-th* semi-máximo periodo del prefijo S_p en S se define como: (1) si $1 < i \leq n$, este es la parte de secuencia entre el final de la primera instancia del prefijo $e_1e_2...e_{i-1}$ en S y el *i-th last-in-first* aparición con respecto al prefijo S_p; (2) si $i = 1$, este es la parte de la secuencia S localizada antes de la 1st *last-in-first* aparición con respecto al prefijo S_p.

Definición 21 (*The i-th last-in-first appearance w.r.t. a prefix sequence*): Para una secuencia S con un prefijo $Sp = e_1e_2...e_n$, la *i-th last-in-first* aparición con respecto al prefijo S_p en S se denota como LF_i y definida recursivamente como: (1) si $i = n$, esta es la última aparición de e_i en la primera instancia de el prefijo S_p en S; (2) si $1 \leq i < n$, es la última aparición de e_i en la primera instancia del prefijo S_p en S mientras LF_i puede aparecer antes de LF_{i+1}.

Visto lo anterior entonces se pueden mostrar, de forma general, los pasos que sigue el algoritmo:

- Determinar el conjunto de ítems frecuentes.
- Proyectar las bases de datos para cada ítem del conjunto de frecuentes.
- repetir para cada ítem del conjunto de frecuentes
  - si no se puede podar (a través del método *BackScan*), realizar el chequeo de la extensión hacia atrás y llamar al método *bide* para ese ítem frecuente.

El conjunto de los ítems frecuentes se determinan, inicialmente, de la base de datos original; luego, los que se analizan, son los ítems frecuentes locales de las bases de datos proyectadas.

Hay que tener en cuenta que cada vez que se hace un crecimiento del patrón hay que proyectar la base de datos correspondiente.

El algoritmo usado por BIDE [41] para enumerar el conjunto de secuencias frecuentes es similar al pseudo-proyección usado por PrefixSpan [29].

Ventajas

Entre las ventajas que presenta el algoritmo están:

- El tratamiento que realiza para eliminar, en las secuencias, el análisis (en el caso de BIDE son de una sola transacción) donde se repita de forma continua el mismo ítem. Por ejemplo, la secuencia $\langle(h, k, k, j, l)\rangle$.
- El uso del método *bi-direccional* para chequear cuándo una secuencia es cerrada, resuelve un gran problema para los algoritmos de este tipo.
- Poda el espacio de búsqueda con mayor profundidad que algoritmos anteriores.

Desventajas

La principal desventaja que presenta el algoritmo BIDE es que solo permite analizar secuencias de una sola transacción. También está el método *bi-direccional* que es bastante costoso, sobre todo, por el chequeo que realiza de la extensión hacia atrás.

3.3.2 Algoritmo CloSpan

CloSpan (Closed Sequencial pattern mining [32]) es un algoritmo para minar secuencias cerradas frecuentes que hace uso de un nuevo método para determinar cuándo una secuencia es cerrada y para la poda del espacio de búsqueda. Este algoritmo puede ser aplicado para bases de datos de cualquier tamaño. Además, introduce el concepto de árbol lexicográfico de secuencias.

Árbol lexicográfico de secuencias

En la construcción del árbol lexicográfico de secuencias lo primero es definir un orden para el conjunto de los ítems dentro de la base de datos. Luego se define un orden para las secuencias con respecto al orden establecido de los ítems. Este orden se define como sigue:

Dadas las secuencias $\alpha = \langle e_1 e_2 ... e_n \rangle$ y $\beta = \langle e'_1 e'_2 ... e'_m \rangle$, donde $e_1 \leq e_2 \leq ... \leq e_n$ y $e'_1 \leq e'_2 \leq ... \leq e'_m$ entonces $\alpha < \beta$ si y solo si se cumple cualquiera de los siguientes puntos:

- dado un valor k, $0 \leq k \leq min(n,m)$, se tiene $e_r = e'_r$ para $r < k$, y $e_k < e'_k$
- $n < m$, y $e_1 = e'_1$ P, $e_2 = e'_2$,... , $e_n = e'_n$

Por ejemplo, dado los siguientes ítems ordenados $I = \{k,h,m,n\}$, se tienen que $\langle (k,m) \rangle < \langle (h,m) \rangle$, $\langle (k,m) \rangle < \langle (k,m,n) \rangle$.

Ahora se puede construir un árbol lexicográfico de secuencias donde cada nodo es un patrón secuencial que se puede extender por extensión del último ítemset de la secuencia (itemset extension) o agregando una secuencia al final (sequence extension) pero siempre manteniendo el orden definido (ver la figura 22).

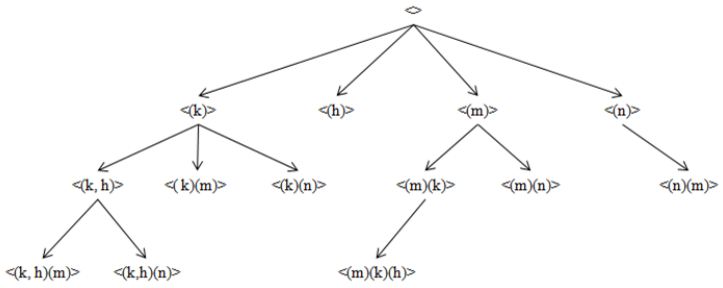

Fig. 22. Árbol lexicográfico de secuencias.

Poda del espacio de búsqueda

En el proceso de poda del espacio de búsqueda se definen varios conceptos dentro de los que se encuentra un teorema de equivalencia entre bases proyectadas.

Teorema (*equivalencia de bases proyectadas*): Dadas dos secuencias S, S' tal que $S \subseteq S$', entonces $D_s = D'_s \Longleftrightarrow I(D_s) = I(D'_s)$

En el teorema anterior $I(D_s)$ significa la cantidad de ítems de la base de datos proyectada D_s por la secuencia S; en esta cantidad se incluyen también los ítems no frecuentes. Es bueno señalar que esto es suficiente para que dos bases de datos sean iguales.

Basado en el lema de terminación temprana por equivalencia (*early termination by equivalence*) CloSpan desarrolla dos métodos de poda, el sub-patrón hacia atrás y el super-patrón hacia atrás.

Ambos métodos tratan de que secuencias que ya han sido crecidas absorban otras secuencias, ya que no es necesario crecer una secuencia si el crecimiento coincide con el de otra secuencia ya desarrollada. Por ejemplo, si dadas dos secuencias $\alpha = \langle (h) \rangle$, $\beta = \langle (k),(h) \rangle$ donde α es menor que β, α es sub-secuencia de β y $I(D_\alpha) = I(D_\beta)$ no es necesario crecer el patrón h para todos los hijos de h ya que son los mismos que los del patrón $\langle (k)(h) \rangle$.

El algoritmo cuenta de dos pasos importantes, el primero hace un pre procesamiento ordenando los ítemsets, calculando los ítems frecuentes y luego borrando los no frecuentes y las secuencias vacías. El segundo paso llama de forma recursiva al algoritmo principal o CloSpan.

Los pasos del primer algoritmo son los siguientes:

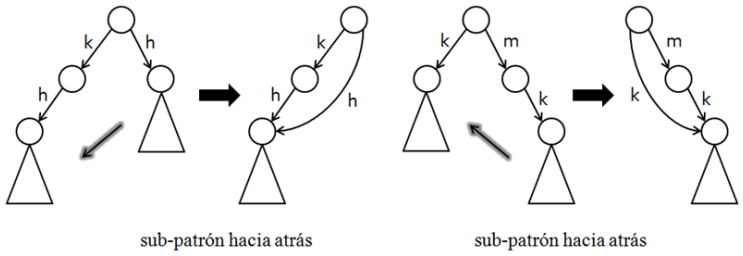

Fig. 23. Sub-patrón y super-patrón hacia atrás.

- Determinar el conjunto de los ítems frecuentes.
- repetir para cada ítem frecuente
  - CloSpan(secuencia (ítem), base de datos proyectada, soporte mínimo, L)
- Eliminar de L las secuencias que no son cerradas.

El algoritmo anterior es por donde comienza el proceso de detección de secuencias frecuentes. Los pasos que siguen el algoritmo CloSpan son:

- Chequea que la secuencia que se va a analizar no esté desarrollada.
- Si la secuencia tiene un super-patrón o un sub-patrón entonces se modifican los enlaces en la estructura sino se inserta un nuevo patrón.
- Se buscan los ítems locales frecuentes a la base proyectada por la secuencia que se analiza.
- repetir para cada ítem local frecuente
  - Extender la secuencia por extensión del último ítemset (itemset extension).
  - Llamar *CloSpan* con la nueva secuencia formada.
- repetir para cada ítem local frecuente
  - Extender la secuencia agregando una secuencia (sequence extension).
  - Llamar *CloSpan* con la nueva secuencia formada.

El proceso usado por CloSpan para la poda puede hacerse un poco engorroso por lo que el algoritmo define una forma de trabajar bastante sencilla. Se define una tabla hash la cual almacenará el par $\langle I(D_s), s\rangle$, y tendrá como llave $I(D_s)$ (cantidad de ítems dentro de la base de datos proyectada) . Por ejemplo, cuando una nueva secuencia es analizada se calcula el $I(D_s)$ que representa el índice en la tabla hash. Luego se busca esta entrada en la tabla y se analiza si la nueva secuencia es sub-secuencia de alguna de las secuencias ya desarrolladas y que tienen la misma entrada en la tabla. En la tabla se almacena un puntero al nodo correspondiente en la secuencia prefijo (ver figura 24).

Ventajas

El método usado para determinar cuándo una secuencia es cerrada, la manera en que poda el espacio de búsqueda y el teorema que define cuando dos bases de datos son equivalentes.

Desventajas

Dentro de las desventajas que presenta el algoritmo esta el gran costo en el consumo de la memoria que presentan los métodos utilizados para podar el espacio de búsqueda y el tamaño del espacio de búsqueda para el caso de que el número de secuencias cerradas sea grande.

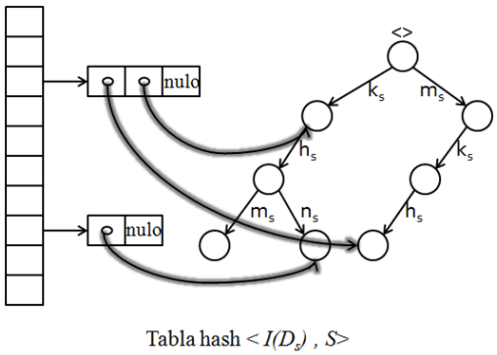

Fig. 24. Tabla hash usada en el algoritmo CloSpan.

3.4 Síntesis y conclusiones

En este capítulo se analizaron algunos de los algoritmos que permiten descubrir patrones secuenciales en bases de datos estáticas. De estos se vieron sus características fundamentales, ventajas y desventajas.

Teniendo en cuenta la estrategia usada los algoritmos se dividieron, para su análisis, en dos grupos: los que están basados en apriori y los basados en el crecimiento de patrones. Además, se analizaron los problemas que presentan estas estrategias.

También se analizaron algoritmos que permiten descubrir patrones secuenciales cerrados y las ventajas que estos pueden traer así como los problemas que implican el descubrimiento de este tipo de secuencia.

En la Tabla 20 se muestra un estudio comparativo de los algoritmos vistos anteriormente.

Tabla 20. Tabla comparativa de los algoritmos que descubren patrones secuenciales en bases de datos estáticas.

Tipo de algoritmo	Algoritmo	Secuencias frecuentes	Secuencias cerradas	Uso de restricciones
Apriori	GSP	X		X
	MSPX	X		
	CAMLS	X		X
	SPIRIT	X		X
Crecimiento de patrones	PrefixSpan	X		
	PRISM	X		
	SPADE	X		
	MEMISP	X		
	LAPIN	X		
	BIDE		X	
	CloSpan		X	

4 Algoritmos de detección de secuencias en bases de datos dinámicas

En este capítulo se expondrán las características de algunos de los algoritmos que minan patrones secuenciales en bases de datos dinámicas. En la figura 25 aparece una taxonomía con los algoritmos que se expondrán.

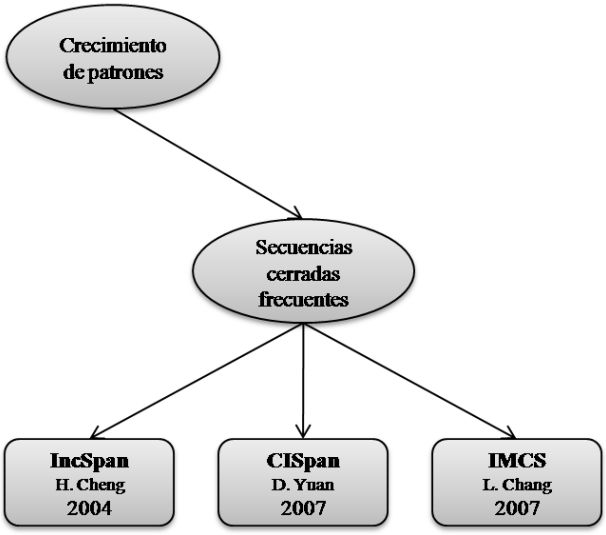

Fig. 25. Taxonomía de algoritmos que usan bases de datos dinámicas.

4.1 Algoritmo IncSpan

IncSpan (Incremental Mining of Sequential Patterns [42]) es un algoritmo para minar secuencias cerradas frecuentes sobre datos incrementales. Solo trabaja sobre inserciones o eliminaciones de datos. El algoritmo usa un árbol de secuencias frecuentes para mantener las secuencias frecuentes y las semi-frecuentes. Las secuencias semi-frecuentes son aquellas que cumplen con la siguiente definición.

Definición 22: Una secuencia S es semi-frecuente si dado un soporte p y un número α se cumple que: $Soporte(S) \leq p + \alpha$

De forma general los pasos que sigue el algoritmo para el análisis de las secuencias frecuentes son:

- Buscar los ítems frecuentes de la base de datos a adicionar.
- Adicionar los nuevos ítems frecuentes al conjunto FS (secuencias frecuentes).
- Adicionar los nuevos ítems semi-frecuentes al conjunto SFS (secuencias semi-frecuentes).
- Para cada ítem en FS
  - Llamar al PrefixSpan para determinar todas las secuencias frecuentes y semi-frecuentes que se forman a partir de ítems correspondientes.
- Para cada patrón secuencial en FS o SFS
  - Si el soporte de la secuencia en la base de datos original mas la variación del soporte en la base de datos a adicionar es igual al soporte, entonces
    - Inserta la secuencia en el conjunto de secuencias frecuentes (FS).
  - Sino
    - Inserta la secuencia en el conjunto de semi-secuencias frecuentes (SFS).

Hay que destacar que el algoritmo hace uso de un nuevo método de macheo, el macheo reverso de patrones que se basa en el análisis de la secuencia desde el final de la misma hacia el inicio. Este método de macheo parte de que los ítems son adicionados siempre por el final de la secuencia, lo que hace al método más eficiente que los anteriores.

Ventajas
Dentro de las ventajas que presenta el algoritmo IncSpan está el mantener además de las secuencias frecuentes las semi-frecuente, esto hace más rápido el proceso de actualización de las secuencias frecuentes, ya que una secuencia semi-frecuente en la base de datos original puede ser frecuente cuando se adicione la nueva base.

Desventajas
Entre las desventajas del algoritmo está la limitante de analizar las modificaciones sobre los datos, es decir, el algoritmo solamente trabaja cuando se realizan actualizaciones basadas en inserciones o eliminaciones de datos, no para datos modificados.

4.2 Algoritmo IMCS

IMCS (Imcremental Mining of Closed Sequential Patterns [43]) es un algoritmo para minar secuencias cerradas frecuentes sobre base de datos incrementales, su característica principal es el uso de de una nueva estructura (árbol CSTree) para el almacenamiento de las secuencias frecuentes.

El funcionamiento del algoritmo se basa en construir a partir de la base de datos el CSTree y mantenerlo actualizado cada vez que se hagan inserciones, modificaciones o eliminaciones a la base.

CSTree
La estructura CSTree (ver figura 26) es un árbol diseñado para mantener las secuencias cerradas frecuentes y otras informaciones auxiliares necesarias para el trabajo del algoritmo. Cada nodo representa un ítem de una secuencia, que comienza en la raíz y termina en el nodo.

Los nodos del árbol CSTree pueden clasificarse en:

- ***Closed node***: Si S_n (secuencia con respecto al nodo n) es una secuencia cerrada, n es un *closed node*.
- ***Stub node***: n es un *stub node* si existe un número entero $i(1 \leq i \leq l)$ y un ítem e el cual aparece en cada uno de los *i-eisimo* semi máximo periodo (ver sub-epígrafe 4.2.3.1 Algoritmo BIDE (16)) de S_n en todas las secuencias en la base de datos (D) que contienen S_n.
- ***Bridge node***: Si S_n es frecuente y n no es ni *closed node* ni *stub node*, entonces n es un *bridge node*.
- ***Non-0 infrequent node***: n es un *non-0 infrequent node* si satisface cualquiera de las siguientes condiciones:
  1. $l = 1 \wedge 0 < sup_D(S_n) < \sigma$;
  2. $l \geq 2 \wedge 0 < sup_D(S_n) < \sigma \wedge sup_D(\langle \alpha_1 ... \alpha l - 1 \rangle) \geq \sigma \wedge supD(\langle \alpha_1 ... \alpha_{l-2} \alpha_l \rangle) \geq \sigma$.

Durante el proceso de actualización de las secuencias frecuentes los nodos pueden cambiar de tipo (*closed node*, *stub node*, *bridge node* y *non-0 infrequent node*).

Un ejemplo de un CSTree se puede observar en la figura 26 donde aparece el antes y después de realizada la inserción de las secuencias $\langle d \rangle$ y $\langle a \rangle$ en la base de datos.

Tabla 21. Base de datos de secuencias.

ID	Secuencia	inc secuencia
1	$\langle (a, b, d) \rangle$	
2	$\langle (a, c, b) \rangle$	d
3	$\langle (c) \rangle$	a

De forma general se pueden definir los pasos del algoritmo IMCS:

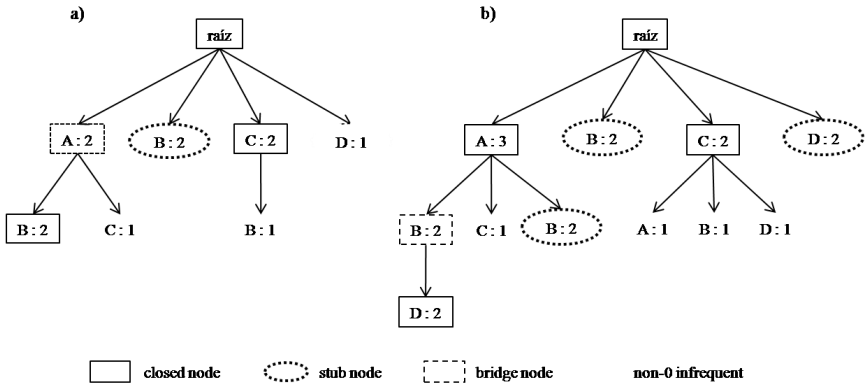

Fig. 26. a) CSTree antes de actualizar; b) CSTree después de actualizar.

- Llamar al método *UpdateCSTree*.
- Llamar al método *ChangeCSTreeNodeState*.

El método *UpdateCSTree* es llamado después de cada modificación para actualizar los soportes de los nodos del árbol y luego se llama al método *ChangeCSTreeNodeState* para actualizar los estados (tipos) de los nodos. En la figura 27 se observan los cambios de los estados en varios nodos, las actualizaciones de sus soportes y algunos nodos nuevos que aparecen durante el proceso de actualización.

Por ejemplo, durante el proceso *UpdateCSTree* luego de haber hecho una actualización a la base de datos los nodos A:2, D:1 cambian de soporte a A:3 y D:2 respectivamente; además de crear nuevos nodos como son: B:2 como hijo de A:3; A:1 y D:1 como hijos de C:2; y D:2 como hijo de B:2. En la figura 27 se puede observar las modificaciones hechas por el método *UpdateCSTree* al árbol *CSTree*.

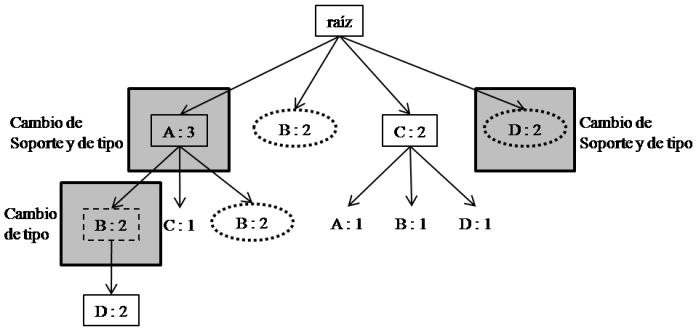

Fig. 27. CSTree después de actualizado la base de datos.

En la figura 27 también se observan los nodos que cambiaron de tipo después ejecutado el método ChangeCSTreeNodeState. Los nodos que cambiaron son los sombreados en la figura anterior, A:3 que cambio de bridge node a closed node; B:2 que cambio de closed node a bridge node; y D:2 que cambio de non-0 infrequent a stub node.

Ventajas

La nueva estructura CSTree desarrollada que permite mantener actualizada las secuencias cerradas frecuentes y otras informaciones adicionales para el trabajo con las secuencias frecuentes.

Desventajas

El método usado para determinar cuándo una secuencia frecuente es cerrada es el mismo creado y usado por el algoritmo BIDE [41], método que es bastante costoso en tiempo.

4.3 Algoritmo CISpan

CISpan (Comprehensive Incremental Sequential Pattern mining [44]) es un algoritmo para minar secuencias cerradas frecuentes en base de datos incrementales. Este algoritmo a diferencia de otros del mismo tipo, permite analizar, además de las inserciones y eliminaciones, modificaciones a la base de datos.

El algoritmo presenta dos estructuras para realizar el procesamiento, una es un retículo de prefijos de secuencias (*prefix sequence lattice*) donde aparecen todas las secuencias cerradas frecuentes de la base de datos original y un retículo incremental (*incremental lattice*) para las secuencias que se agregaran a la base de datos. La estructura *prefix sequence lattice* se crea al inicio de la ejecución del algoritmo después de haber calculado el conjunto de todas las secuencias cerradas frecuentes y la segunda estructura *incremental lattice* cada vez que se actualiza la base de datos.

La forma general de trabajar que sigue el algoritmo es la siguiente:

- Inicialmente se llama al algoritmo CloSpan para calcular las secuencias cerradas frecuentes de la base de datos y crear el retículo de prefijos de secuencias con este resultado.
- Cada vez que se adicionen secuencias a la base de datos
  - Se crea el retículo incremental con esas secuencias que se agregan.
  - Se toma el retículo de prefijos actualizado (se actualiza en caso de que se tengan que hacer eliminaciones).
  - Se mezclan estos dos retículos para formar un retículo resultante.

El retículo resultante de este análisis es el mismo que si se analizara la base de datos desde el inicio con el algoritmo ClosSpan.

En el momento que se tienen las dos estructuras estas se mezclan para formar la base de datos actualizada (ver la figura 28).

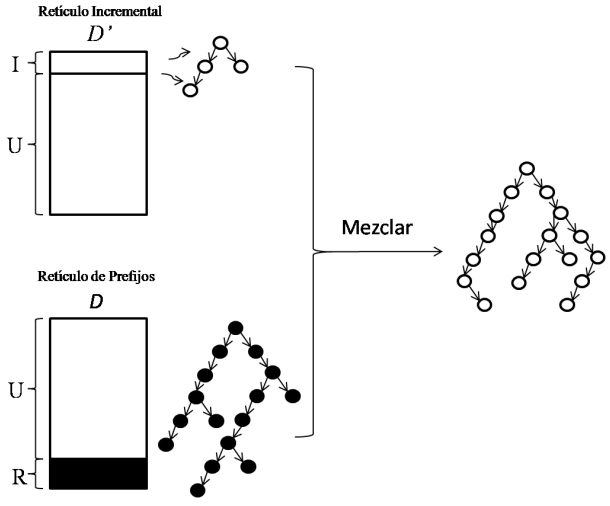

Fig. 28. Mezcla de los retículos de las bases de datos.

Ventajas

A diferencia de otros algoritmos sobre bases de datos dinámicas que solo trabajan con eliminaciones y adiciones, CISpan trabaja, además, con modificaciones.

Desventajas

La manera en que CISpan analiza las modificaciones a la base de datos trae que se trabaje como si se hubieran realizados dos operaciones distintas, inserción y eliminación, lo que provoca un procesamiento mayor para este caso.

4.4 Síntesis y conclusiones

En este capítulo se analizaron algunos de los algoritmos que permiten descubrir patrones secuenciales en bases de datos dinámicas. De estos se vieron sus características fundamentales, ventajas y desventajas. También se analizo, para cada uno, el problema fundamental que presentan estos algoritmos que está relacionado con el descubrimiento de las nuevas secuencias frecuentes que pueden aparecer cada vez que se hace una actualización en la base de datos (dígase inserción, modificación o eliminación de secuencias en la base de datos).

En la Tabla 22 se presenta un estudio comparativo entre algunas de las características de los algoritmos expuestos.

Tabla 22. Tabla comparativa de los algoritmos que descubren patrones secuenciales en bases de datos estáticas.

Algoritmo	Secuencias cerradas	Analiza modificaciones	Mantiene secuencias semi-frecuentes
IncSpan	X		X
IMCS	X		
CISpan	X	X	

5 Conclusiones

En el presente trabajo se realizó un análisis sobre los principales algoritmos que minan el conjunto de secuencias frecuentes. Los cuales se dividieron en tres grupos teniendo en cuenta la técnica usada y la base de datos sobre la que trabajan. Estos grupos están compuesto por: los basados en la estrategia apriori y de crecimiento de patrones en bases de datos estáticas; y los algoritmos que minan patrones secuenciales en bases de datos dinámicas.

También se hizo un estudio de las diferentes definiciones relacionadas con el tema de minado de secuencias, así como un análisis sobre el uso de restricciones en el minado de las mismas.

Se expusieron los algoritmos basados en la estrategia apriori y varios de los problemas que estos presentan en el cálculo de los patrones secuenciales como son, la gran cantidad de secuencias candidatas que generan y que tienen que ser analizadas en cada pasada, muchas de las cuales no son frecuentes. El tiempo requerido para el conteo (determina que la secuencia sea frecuente) de cada secuencia candidata; esto último está dado por la gran cantidad de secuencias que se generan o por patrones secuenciales de gran tamaño.

Con respecto a los algoritmos basados en el crecimiento de patrones, se pudieron apreciar los problemas relacionados con el espacio requerido de memoria para proyectar las bases, las cuales pueden llegar a

aumentar considerablemente; y los casos en que las bases de datos están compuestas por secuencias muy largas o cuando los patrones secuenciales son grandes (longitud de la secuencia).

En cuanto al grupo de los algoritmos que trabajan en bases de datos dinámicas se pudieron apreciar algunos de los problemas que estos presentan, tales como: el almacenamiento de las secuencias frecuentes que van siendo encontradas cada vez que se realiza una actualización en la base de datos (dígase inserciones o eliminaciones de secuencias); proceso que esta dado con el objetivo de no tener que realizar todo el proceso de obtención de secuencias frecuentes desde el inicio. También está el problema de las nuevas secuencias que son adicionadas y que pueden ser frecuentes pero que no son tomadas como tal, dado que estas (las nuevas secuencias) son analizadas con respecto a las secuencias frecuentes que ya han sido encontradas y no con respecto a la base de datos original.

Basado en lo antes expuesto y lo expresado en el documento, se puede decir que existen diferentes áreas de trabajos futuros:

- El minado de secuencias frecuentes sobre bases de datos incrementales.
- La generalización (gap, ventana de tiempo y taxonomías) en la obtención de secuencias frecuentes.
- La creación de algoritmos para la minería de secuencias frecuentes cerradas o maximales.
- La creación de estructuras de datos que mejoren el tamaño del espacio de búsqueda y/o rapidez del análisis.
- La creación de métodos que permitan seleccionar el tipo de secuencias frecuentes que se quieren obtener. Por ejemplo, se pueden tener como secuencias frecuentes $\langle(DBC)\rangle$ y $\langle(ABE)(D)\rangle$ pero se desea poner como restricción, además de ser la secuencia frecuente, que debe tener como sufijo a la sub-secuencia $\langle(AB)\rangle$; por lo tanto, en este caso no se daría como resultado final a la secuencia frecuente $\langle(DBC)\rangle$ ya que no cumple con la restricción que se pide. Ejemplo de esto es el usado por el algoritmo SPIRIT, el cual trabaja con autómatas finitos deterministas y expresiones regulares para delimitar las secuencias que se desean obtener.

Referencias bibliográficas

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proceedings of 1995 International Conference Data Engineering (ICDE'95). (March 1995) 3–14
2. Agrawal, R., Srikant., R.: Mining sequential patterns. In: In Proceedings of the 11th International Conference on Data Engineering. (1995) 3–14
3. T. P. Exarchos, C. Papaloukas, C.L., Fotiadis., D.I.: Mining sequential pattern for protein fold recognition. Journal of Biomedical Informatics **1** (2008) 165–179
4. Ezeife C.I., L.Y.: Mining web log sequential patterns with position coded pre-order linked wap-tree. Journal of Data Mining and Knowledge Discovery (DMKD) **10** (2005) 5–38
5. R. J. Kuo, C.M.C., Liu., C.Y.: Integration of k-means algorithm and apriorisome algorithm for fuzzy sequential pattern mining. Applied Soft Computing **9** (2009) 85–92
6. J. Han, J.P., Yin., Y.: Mining frequent patterns without candidate generation. In: In Proceeding of the ACM SIGMOD International Conference on Management of Data. (2000) 1–12
7. M. Baumgarten, A.G.B., Hughes., J.G.: Tree-growth based sequential and associated pattern discovery. In: In Proceeding of the Fifteenth International Conference on Software Engineering & Knowledge Engineering. (2003) 240–244
8. M. Leleu, C. Rigotti, J.F.B., Euvrard., G.: Go_spade: Mining sequential over datasets with consecutive repetitions. In: In Proceeding of the International Conference on Machine Learning and Data Mining in Pattern Recognition. (2003) 293–306
9. M. Zhang, B. Kao, C.L.Y., Cheung., D.: A gsp-based efficient algorithm for mining frequent sequences. In: In Proceeding of the International Conference on Applied Informatics. (2001) 14–18
10. L. Chang, T. Wang, D.Y., Luan., H.: Seqstream: Mining closed sequential patterns over stream sliding windows. In: Proceedings of the IEEE International Conference on Data Mining. (October 2008) 83–92
11. P. Tzvetkov, X.Y., Han., J.: Tsp: Mining top-k closed sequential patterns. In: In Proceeding of the 3rd IEEE International Conference on Data Mining. (2003) 347–354

12. Li, C., Wang., J.: Efficiently mining closed subsequences with gap constraints. In: In Proceeding of the SIAM International Conference on Data Mining. (2008) 313–322
13. C. Raïssi, P.P., Teisseire., M.: Speed: Mining maximal sequential patterns over data streams. In: 3rd International IEEE Conference on Intelligent Systems. (2006) 546–552
14. D. Lo, S.C.K., Li., J.: Mining a raking generators of sequential patterns. In: In Proceedings of the SIAM International Conference on Data Mining. (2008) 553–564
15. H. Cheng, X.Y., Han., J.: Seqindex: Indexing sequences by sequential pattern analysis. In: In Proceeding of the Fifth SIAM International Conference on Data Mining. (2005) 601–605
16. Yun., U.: Wis: Weighted interesting sequential pattern mining with a similar level of support and/or weight. ETRI Journal **29** (2007) 336–352
17. Yang, J., Wang., W.: Cluseq: Efficient and effective sequence clustering. In: In Proceeding of the 19th IEEE International Conference on Data Engineering. (2003) 101–112
18. Ezeife, C.I., Monwar., M.: Ssm: A frequent sequential data stream patterns miner. In: In Proceeding of the IEEE Symposium on Computational Intelligence and Data Mining. (2007) 120–126
19. Wei-Hua Hao, N. P. Lin, H.J.C.C.I.C., Chueh., H.E.: Maintaining and mining sequential patterns in incremental sequence database. In: In Proceedings of the 2008 International Computer Symposium. (2009) 335–340
20. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: Proceeding in the 5th International Conference Extending Database Technology. (March 1996) 3–17
21. Sandra de Amo, W.P.J., Giacometti., A.: Milprit: A constraint-based algorithm for mining temporal relational patterns. International Journal of Data Warehousing and Mining **4** (2008) 42–61
22. Y. Tong, L. Zhao, D.Y.S.M., Xu., K.: Mining compressed repetitive gapped sequential patterns efficiently. In: In Proceedings of the Advanced Data Mining and Applications. (2009) 652–660
23. Hong X., Wei Y., W.Z.: Audit oriented fast algorithm for sequential mining with constraints. In: In proceedings of The International Conference on Computational Intelligence for Modelling. (2005) 957–962
24. M. Garofalakis, R.R., Shim, K.: Spirit: Sequential pattern mining with regular expression constraints. In: Proceedings of the 25th International Conference on Very Large Data Bases. (September 1999) 223–234
25. Mimaroglu, S., Simovici., D.A.: Binary sequences and association graphs for fast detection of sequential patterns. In: In Extraction at Gestion des Connaissances. (2009)
26. J. Han, H. Cheng, D.X., Yan., X.: Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery **15** (2007) 55–86
27. J., A.: Mining sequential patterns. Technical report (2001)
28. D. Saputra, D.R.A.R., Foong., O.M.: Mining sequential patterns using i-prefixspan. International Journal of Computer Science and Engineering **2** (2008) 549–554
29. J. Pei, J. Han, B.M.A., Pinto, H.: Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceeding of the International Conference on Data Engineering. (April 2001) 215–224
30. K. Gouda, M.H., Zaki, M.: Prism: A prime-encoding approach for frequent sequence mining. Journal of Computer and System Sciences **76** (February 2010) 88–102
31. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Machine Learning Journal **42**(1/2) (Jan/Feb 2001) 31–60
32. X. Yan, J.H., Afshar, R.: Clospan: Mining closed sequential patterns in large datasets. In: Proceeding of 2003 SIAM International Conference on Data Mining. (May 2003) 166–177
33. Masseglia, F., Poncelet, P., Teisseire, M.: Efficient mining of sequential patterns with time constraints: Reducing the combinations. Expert Syst. Appl. **36** (March 2009) 2677–2690
34. Luo, C., Chung, S.M.: Efficient mining of maximal sequential patterns using multiple samples. In: SDM. (2005) 131–138
35. Y. Gonen, N. Gal-Oz, R.Y., Gudes, E.: Camls: A constraint-based apriori algorithm for mining long sequences. In: Proceeding of the 15th International Conference of Database Systems for Advanced Applications. (April 2010)
36. K. Gouda, M.H., Zaki, M.: Prism: A prime-encoding approach for frequent sequence mining. In: Proceedings of the IEEE International Conference on Data Mining. (October 2007) 487–492
37. Zaki, M.J.: Efficient enumeration of frequent sequences. In: Proceedings of the seventh international conference on Information and knowledge management. CIKM '98, New York, NY, USA, ACM (1998) 68–75
38. Zaki, M.J.: Scalable data mining for rules. Technical report, Rochester, NY, USA (1998)
39. Lin, M., Lee, S.: Fast discovery of sequential patterns through memory indexing and database partitioning. Journal of Information Science **21** (2005) 109–128
40. Yang, Z., Wang, Y., Kitsuregawa, M.: Lapin: Effective sequential pattern mining algorithms by last position induction for dense databases. In: DASFAA. (2007) 1020–1023
41. Wang, J., Han, J.: Bide: Efficient mining of frequent closed sequences. In: Proceedings of the 20th International Conference on Data Engineering. ICDE '04, Washington, DC, USA, IEEE Computer Society (2004) 79–

42. Cheng, H., Yan, X., Han, J.: Incspan: incremental mining of sequential patterns in large database. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '04, New York, NY, USA, ACM (2004) 527–532
43. Chang, L., Yang, D., Wang, T., Tang, S.: Imcs: incremental mining of closed sequential patterns. In: Proceedings of the joint 9th Asia-Pacific web and 8th international conference on web-age information management conference on Advances in data and web management. APWeb/WAIM'07, Berlin, Heidelberg, Springer-Verlag (2007) 50–61
44. Yuan, D., Lee, K., Cheng, H., Krishna, G., Li, Z., Ma, X., Zhou, Y., Han, J.: Cispan: Comprehensive incremental mining algorithms of closed sequential patterns for multi-versional software mining. In: SDM'08. (2008) 84–95

RT_016, junio 2011
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Lucía González Bayona
Diseño de Portada: Di. Alejandro Pérez Abraham
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba