RNPS No. 2143
ISSN 2072-6260
Versión Digital

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

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

SPRITE: Un nuevo algoritmo para la minería de secuencias frecuentes

José Kadir Febrer Hernández,
Raudel Hernández León y
José Hernández Palancar

RT_026 **mayo 2014**

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

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

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

SPRITE: Un nuevo algoritmo para la minería de secuencias frecuentes

José Kadir Febrer Hernández,
Raudel Hernández León y
José Hernández Palancar

RT_026 **mayo 2014**

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

Tabla de contenido

Lista de figuras

Lista de tablas

SPRITE: Un nuevo algoritmo para la minería de secuencias frecuentes

José Kadir Febrer Hernández[1], Raudel Hernández León[1] y José Hernández Palancar[2]

[1] Equipo de Investigaciones de Minería de Datos, Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV), La Habana, Cuba
{jfebrer, rhernandez}@cenatav.co.cu
[2] Equipo de Investigaciones de Biometría, Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV), La Habana, Cuba
{jpalancar}@cenatav.co.cu

RT_026, Serie Gris, CENATAV
Aceptado: 20 de enero de 2014

Resumen. En el presente trabajo se propone un nuevo algoritmo, denominado SPRITE (**S**equential **P**atte**r**n Min**i**ng based on Frequent **T**wo-S**e**quences), para la minería de secuencias frecuentes en conjuntos de datos estáticos. SPRITE introduce una nueva estructura de datos que permite incrementar la eficiencia en la generación de las secuencias candidatas. Adicionalmente, SPRITE propone una estrategia de poda basada en la intersección de vectores binarios para reducir la cantidad de secuencias candidatas a analizar. En los experimentos realizados se comprobó que SPRITE tiene mejor desempeño que los principales algoritmos de cálculo de secuencias frecuentes, reportados en la literatura.
Palabras clave: minería de datos, secuencias frecuentes, patrones secuenciales.

Abstract. In this report, we propose a novel sequential pattern mining algorithm, called SPRITE (**S**equential **P**atte**r**n Min**i**ng based on Frequent **T**wo-S**e**quences), which works on static datasets. SPRITE introduces a new data structure that allows to increase the efficiency in the generation of candidate sequences. Moreover, SPRITE proposes a pruning strategy based on the intersection of binary vectors to reduce the number of candidate sequences to analyze. Our experimental study shows that SPRITE outperforms popular sequence mining methods.
Keywords: data mining, frequent sequences, sequential patterns.

1. Introducción

El presente trabajo tiene como precedente un reporte técnico [1] titulado: "Estado del arte para el minado de patrones secuenciales", donde se realizó un estudio de los algoritmos de minería de secuencias frecuentes existentes, analizándose sus ventajas y desventajas, así como la importancia y actualidad de la minería de secuencias.

El problema de encontrar todas la secuencias frecuentes en un conjunto de datos es exponencial. Por ejemplo, si se tiene un conjunto formado por 10 ítems diferentes, se generan 190 posibles secuencias candidatas de dos ítems, 100 secuencias formadas por dos elementos de 1 ítem cada uno ($\langle i_j\ i_k\rangle$) y 90 secuencias formadas por un solo elemento de 2 ítems ($\langle i_j i_k\rangle$, $i_j \neq i_k$). Si en vez de 100 tuviéramos 1000 ítems diferentes, las cantidad de secuencias candidatas sería 1999000. Como se puede observar, un aumento en la cantidad de ítems provoca un considerable aumento en la cantidad de secuencias candidatas y por consiguiente, un aumento en el tiempo de procesamiento.

Los algoritmos de minería de secuencias utilizan diferentes estrategias para lograr una mayor eficiencia, ya sea introduciendo nuevas estrategias de poda [2], o desarrollando nuevas estructuras de datos [3]. Por lo general, los algoritmos reportados siguen dos estrategias fundamentales: apriori (generación de candidatos y conteo de ocurrencias) [2,4,5] y crecimiento de patrones [6,7,8].

En este reporte técnico se propone un nuevo algoritmo para el cálculo de secuencias frecuentes, denominado SPRITE, que obtiene mejores resultados de eficiencia que los principales algoritmos reportados para el cálculo de secuencias frecuentes. SPRITE introduce una nueva estructura de datos para almacenar las secuencias frecuetes y para acelerar el proceso de generación de las secuencias candidatas.

El documento está organizado de la siguiente forma: en la Sección 2 se presentan los conceptos básicos de la minería de secuencias frecuentes. En la Sección 3 se describe el trabajo relacionado. En la Sección 4 se introduce la nueva estructura de datos y se detalla el algoritmo (SPRITE). Los resultados experimentales y el análisis de los mismos se presentan en la Sección 5. Finalmente, se exponen las conclusiones en la Sección 6.

2. Conceptos básicos

En el presente reporte nos limitaremos a describir los conceptos y definiciones necesarios para la comprensión del algoritmo propuesto.

En lo adelante se utilizarán los símbolos "⟨" y "⟩" para delimitar los elementos de una secuencia, así como los símbolos "(" y ")" para delimitar los conjuntos de ítems, en este último caso los paréntesis pueden omitirse para una mayor simplicidad.

La minería de secuencias frecuentes, de forma similar a la minería de conjuntos frecuentes de items, es una técnica cuyo objetivo es calcular todas las secuencias frecuentes, a partir de un conjunto de transacciones. A diferencia de un conjunto de ítems (*itemset*), donde un ítem ocurre a lo sumo una vez, en una secuencia un *itemset* puede ocurrir múltiples veces. Adicionalmente, a diferencia de la minería de *itemsets*, donde $(abc) = (cba)$, en la minería de secuencias $\langle\ (ab)\ c\ \rangle \neq \langle\ c\ (ab)\ \rangle$.

Sea $I = \{i_1, i_2, ..., i_n\}$ un conjunto de elementos o ítems y T un conjunto de transacciones, donde cada transacción es una secuencia no vacía $\langle \alpha_1\ \alpha_2\ ...\ \alpha_m \rangle$, tal que $\alpha_i \subseteq I$. El tamaño de un *itemset* se define por su cardinalidad y un *itemset* que contiene k ítems se denomina k-itemset. De forma similar, una secuencia que contiene k *itemsets* se denomina k-secuencia.

Una secuencia $\alpha = \langle \alpha_1\ \alpha_2\ ...\ \alpha_m \rangle$ es una sub-secuencia de una secuencia $\beta = \langle \beta_1\ \beta_2\ ...\ \beta_r \rangle$, si existen valores enteros $j_1 < j_2 < ... < j_m$ tal que $\alpha_1 \subseteq \beta_{j_1}$, $\alpha_2 \subseteq \beta_{j_2}$, ..., $\alpha_m \subseteq \beta_{j_m}$. Adicionalmente, se denomina posición de ocurrencia de la secuencia α en la secuencia β, posOcurrencia(α, β), a la posición que ocupa el elemento β_{j_m} en la secuencia β. Es válido aclarar que una secuencia α puede tener más de una posición de ocurrencia en una secuencia β.

El Soporte de una secuencia α es la fracción de las transacciones en T que contienen a α (ver Ec. 1). Una secuencia es frecuente si su Soporte es mayor o igual a un umbral específico.

$$Sop(\alpha) = \frac{|T_\alpha|}{|T|}\ , \tag{1}$$

donde T_α es el conjunto de transacciones en T que contienen a α y $|\cdot|$ representa la cardinalidad.

Por ejemplo, dados el conjunto de transacciones de la Tabla 1 y un umbral de Soporte igual a 0,6, la secuencia $\langle a \rangle$ es frecuente porque hay cuatro transacciones que la contienen (Soporte igual a 0,8, ver transacciones 1-4).

Tabla 1. Conjunto de transacciones.

Tid	Secuencia
1	$\langle a\ b \rangle$
2	$\langle cd\ aef\ b \rangle$
3	$\langle af \rangle$
4	$\langle aef\ b \rangle$
5	$\langle b \rangle$

3. Trabajo relacionado

Desde Apriori [9], propuesto por Agrawal y Srikant, muchos han sido los algoritmos desarrollados para la minería de secuencias frecuentes. En general, estos algoritmos se pueden dividir en dos grupos de acuerdo con la estrategia que utilizan para generar las secuencias: (1) los basados en apriori y (2) los basados en crecimiento de patrones. En el reporte técnico anterior [1] se profundiza en las diferencias entre estos dos grupos, así como en las características de los mismos. De igual forma, se detallan cada uno de los principales algoritmos, por lo tanto, en este apartado nos limitaremos solo a mencionarlos.

Entre los algoritmos más representativos del grupo 1 se encuentran GSP [10] y SPIRIT [11], el primero permite utilizar diferentes restricciones como son: la distancia entre los ítems (GAP), el uso de ventanas de tiempo y el uso de taxonomías; en cambio, el segundo utiliza expresiones regulares para filtrar el conjunto de secuencias frecuentes.

El principal exponente de los algoritmos basados en crecimiento de patrones (grupo 2), es el algoritmo PrefixSpan [6], al cual se le han implementado varias mejoras, entre las que se encuentran: GenPrefixSpan [12] , I-PrefixSpan [13], MI-PrefixSpan [14], C-PrefixSpan [15] y P-PrefixSpan [16], entre otros. Para cada ítem frecuente, el algoritmo PrefixSpan proyecta el conjunto de datos y sobre esta proyección repite el proceso recursivamente.

Además de PrefixSpan, se han desarrolado otros algoritmos que han obtenido muy buenos resultados, entre ellos figuran: ESPE [17], LAPIN [7], PRISM [8] y SPAN [18], entre otros. El algoritmo propuesto en este trabajo (SPRITE) toma ideas de los algoritmos LAPIN, ESPE y PRISM. En la siguiente sección se describe detalladamente el algoritmo SPRITE.

4. Algoritmo propuesto

En esta sección se presenta el algoritmo SPRITE y se mencionan sus ventajas con relación al estado del arte. Como parte del algoritmo SPRITE, se introduce una nueva estructura de datos que garantiza una eficiente generación de las secuencias candidatas y se propone una estrategia de poda que reduce el número de secuencias candidatas a analizar.

4.1. Estructura de Datos

La nueva estructura de datos almacena, por cada secuencia frecuente α y por cada transacción t, una lista con las posiciones de ocurrencia de α en t. Además, se almacena un vector binario con un 1 en las transacciones que contienen a α y un 0 en las transacciones que no la contienen.

Por ejemplo, en la Figura 1 se muestran cinco transacciones y el vector binario asociado a la secuencia $\langle f \rangle$. Además, en la Tabla 2 se muestran las posiciones de ocurrencia de la secuencia $\langle f \rangle$ en el conjunto de transacciones de la Figura 1.

El algoritmo LAPIN también utiliza una estructura de datos para almacenar las posiciones de los ítems. Por tanto, es importante marcar las diferencias: LAPIN almacena por cada ítem y por cada transacción, la última posición de cada ítem (ver Tabla 3) mientras que SPRITE almacena por cada secuencia frecuente α todas las posiciones de ocurrencia de α en las transacciones que la contengan.

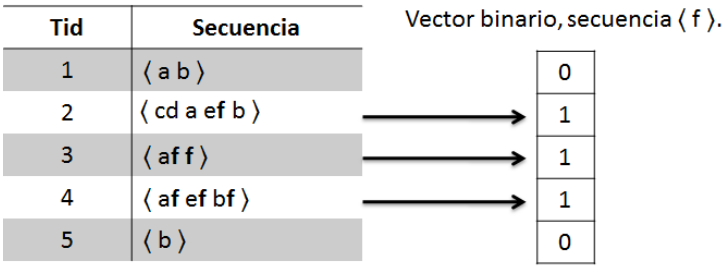

Fig. 1. Vector binario asociado a la secuencia $\langle f \rangle$.

Tabla 2. Posiciones de ocurrencia de la secuencia $\langle f \rangle$ en el conjunto de transacciones de la Figura 1.

Tid	Posiciones de ocurrencia
2	3
3	1, 2
4	1, 2, 3

Tabla 3. Posiciones almacenadas por el algoritmo LAPIN.

Tid	Última posición en la secuencia
1	a = 1, b = 2
2	c = 1, d = 1, a = 2, e = 3, f = 3, b = 4
3	a = 1, f = 2
4	a = 1, f = 3 , e = 2, b = 3
5	b = 1

4.2. Algoritmo SPRITE

En una primera etapa, el algoritmo SPRITE calcula todas las secuencias frecuentes de tamaño 1 y de cualquier longitud y realiza un mapeo de las mismas a números enteros positivos, de forma similar a los principales algoritmos reportados. Como se mencionó en la sección anterior, SPRITE almacena por cada secuencia frecuente t, calculada en la primera etapa (1-secuencias), una lista de ocurrencias (con las posiciones de ocurrencia) por cada transacción que contenga a t y un vector binario con la presencia o no de t en las transacciones.

En una segunda etapa, SPRITE genera las posibles $2-$secuencias a partir de las 1-secuencias y calcula sus Soportes. Dadas las 1-secuencias frecuentes $\alpha = \langle i \rangle$ y $\beta = \langle j \rangle$, antes de calcular el Soporte de la secuencia candidata $\langle i\ j \rangle$, SPRITE aplica una estrategia de poda que consiste en intersectar los vectores binarios de α y β, en caso de que la cantidad de bits activos del vector resultante ($\mathrm{Sop}(\langle i\ j \rangle)$) sea menor que el umbral de Soporte establecido entonces no se realiza el conteo real del Soporte de $\langle i\ j \rangle$. En caso de que la cantidad de bits activo satisfaga el umbral de Soporte entonces se recorren las listas de ocurrencia de α y β se determina el Soporte real ya que no basta con que α y β estén en la misma transacción, además

se necesita que la posición de ocurrencia (ver Sección 2) de β sea mayor que la posición de ocurrencia de α. El Algoritmo 1 muestra el pseudo código para calcular las 2-secuencias frecuentes.

Algorithm 1: Cálculo de las 2-secuencias frecuentes.

Input: Conjunto de transacciones T y umbral de Soporte *minSup*.
Output: Todas las 2-secuencias frecuentes.

```
L1 = {unoSecuenciasFrecuentes(T)}
L2 = ∅
foreach ⟨I⟩ ∈ L1 do
    foreach ⟨J⟩ ∈ L1 do
        \\ se intersectan los vectores binarios de ⟨I⟩ y ⟨J⟩ para calcular
        \\ el máximo Soporte posible de ⟨I J⟩, denominado maxSop, si
        \\ maxSop ≤ minSup entonces no se analiza la secuencia ⟨I J⟩.
        if maxSop > minSup then
            Sop = 0
            foreach t ∈ T do
                if posOcurrencia(⟨J⟩,t) > posOcurrencia(⟨I⟩,t) then
                    Sop = Sop + 1
                end
            end
            if Sop > minSup then
                L2 = L2 ∪ {⟨I J⟩}
            end
        end
    end
end
return L2
```

De igual forma que se hace para las 1-secuencias, SPRITE almacena para cada 2-secuencia frecuente t una lista de ocurrencias por cada transacción que la contenga y un vector binario con la presencia o no de t en las transacciones.

Antes de continuar presentaremos la siguiente definición:

Definición 1 *Dadas las secuencias* $\alpha = \langle \alpha_1\ \alpha_2\ ...\ \alpha_m \rangle$ *y* $\beta = \langle \alpha_m\ \beta_1 \rangle$*, se define como la unión entre* α *y* β *a la secuencia* $\langle \alpha_1\ \alpha_2\ ...\ \alpha_m\ \beta_1 \rangle$ *y se utilizará el operador* $\bigoplus$ *para indicar la unión de dos secuencias.*

Finalmente, en una tercera etapa, dada una k-secuencia $\alpha = \langle \alpha_1\ \alpha_2\ ...\ \alpha_k \rangle$, las secuencias candidatas de tamaño $k+1$ que se obtienen a partir de α se generan mediante la unión de esta con todas las 2-secuencias de la forma $\beta = \langle \alpha_k\ \beta_1 \rangle$. El Algoritmo 2 muestra el pseudo código para calcular las K-secuencias frecuentes, con $k \geq 2$.

5. Resultados experimentales

En esta sección se presentan los resultados obtenidos en los experimentos realizados, en los cuales se comparó el algoritmo SPRITE con los principales algoritmos de cálculo de secuencias frecuentes, reportados en la literatura, como son: GSP [10], PrefixSpan [6], LAPIN [7] y PRISM [8]. Los códigos fuentes de los algoritmos fueron brindados por los propios autores. Los experimentos se realizaron sobre cinco conjuntos de datos, construidos con un generador (ver parámetros en la Tabla 4), desarrollado por el grupo

Algorithm 2: Cálculo de las $(k+1)$-secuencias frecuentes a partir de las k-secuencias frecuentes $(k \geq 2)$.

Input: Conjunto de transacciones T, conjunto de las k-secuencias frecuentes $kFrec$, conjunto de las 2-secuencias frecuentes $dosFrec$ y umbral de Soporte $minSup$.
Output: Todas las $(k+1)$-secuencias frecuentes.

$L_1 = \emptyset$
foreach $\alpha = \langle \alpha_1\ \alpha_2\ ...\ \alpha_k \rangle \in kFrec$ **do**
  foreach $\beta = \langle \alpha_k\ \beta_1 \rangle \in dosFrec$ **do**
    \\ se intersectan los vectores binarios de α y β para calcular
    \\ el máximo Soporte posible de $\alpha \bigoplus \beta$, denominado $maxSop$, si
    \\ $maxSop \leq minSup$ entonces no se cuenta el Soporte real.
    if $maxSop > minSup$ **then**
      $Sop = 0$
      foreach $t \in T$ **do**
        if $posOcurrencia(\beta, t) > posOcurrencia(\alpha, t)$ **then**
          $Sop = Sop + 1$
        end
      end
      if $Sop > minSup$ **then**
        $L_2 = L_2 \cup \{\alpha \bigoplus \beta\}$
      end
    end
  end
end
return L_2

de investigación de minería de datos del Departamento de Ciencias de la Computación de la Universidad de Illinois (http://illimine.cs.uiuc.edu).

Tabla 4. Parámetros de entrada para el generador de conjuntos de datos sintéticos.

Parámetro	Descripción
C	Promedio de *itemsets* por secuencia (transacción).
T	Promedio de ítems por *itemset*.
S	Longitud promedio de las secuencias maximales.
I	Longitud promedio de los *itemsets* en las secuencias maximales.
N	Cantidad de ítems diferentes.
D	Cantidad de transacciones del conjunto de datos.

En los experimentos se consideró el tiempo de ejecución como métrica de comparación de igual forma que en la literatura reportada. Los experimentos se realizaron en una computadora con procesador Intel Core 2 Quad a 2,5 GHz y 4 Gb DDR3 RAM, corriendo sobre el sistema operativo Windows 7. El algoritmo fue implementado en el lenguaje ANSI C standard. Se consideró el tiempo de procesamiento, sin incluir los tiempos de entrada/salida, como el tiempo de comparación para todos los algoritmos incluidos en este reporte.

Se generaron cinco conjuntos de datos sintéticos, ver los valores de los parámetros en la Tabla 5. Como se puede observar, se modificaron pricipalmente los valores de los parámetros que más influyen en la eficiencia de estos algoritmos: promedio de *itemsets* por transacciones, número de ítems y número de transacciones.

Tabla 5. Valores de los parámetros de los conjuntos de datos utilizados en los experimentos.

Conjunto de datos	C	T	S	I	N	D
DS_1	20	3	4	4	20	10 000
DS_2	40	3	4	4	20	20 000
DS_3	40	3	4	4	50	500
DS_4	10	20	4	4	100	1 000
DS_5	10	20	4	4	100	10 000

Para evaluar el comportamiento del algoritmo SPRITE se realizaron cinco experimentos, uno por cada conjunto de datos de la Tabla 5. En el primer experimento (ver Figura 2) se emplearon los valores de parámetros más utilizados en la literatura, y como se puede observar, SPRITE obtiene los mejores resultados, seguido por el algoritmo PRISM.

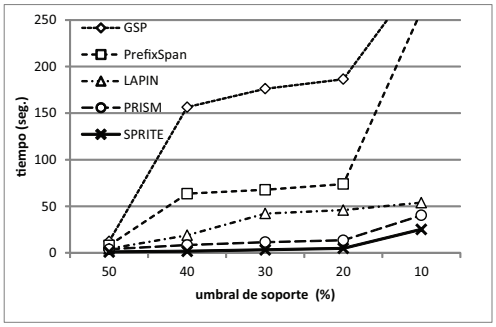

Fig. 2. Tiempo de ejecución utilizando el conjunto de datos DS_1.

En el segundo experimento se incrementaron simultáneamente el número de secuencias, de 10000 a 20000, y el promedio de *itemsets* por secuencias de 20 a 40. Como resultado se obtuvo que los algoritmos más afectados fueron GSP y PrefixSpan, en ambos casos el tiempo de ejecución superó los 250 segundos (ver Figura 3).

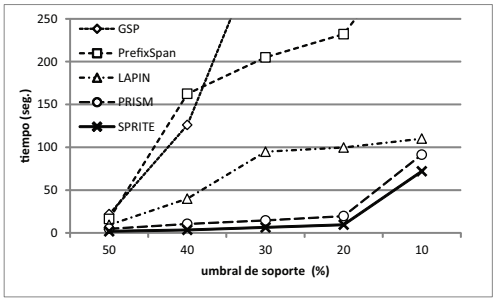

Fig. 3. Tiempo de ejecución utilizando el conjunto de datos DS_2.

En el tercer experimento se incrementó el número de ítems de 20 a 50, se mantuvo en 40 el promedio de *itemsets* por secuencia y se redujo considerablemente la cantidad de transacciones. En la Figura 4 se pueden ver los tiempos de ejecución de todos los algoritmos con la excepción del algoritmo LAPIN, el cual no ejecutó correctamente con umbrales de Soporte por debajo del 10 %. Para mostrar el rendimiento del algoritmo SPRITE, se utilizaron en este experimento bajos umbrales de Soporte para obtener una mayor cantidad de secuencias frecuentes.

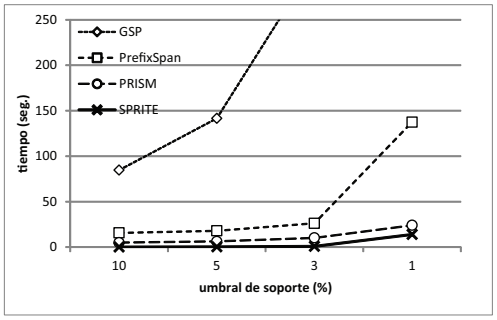

Fig. 4. Tiempo de ejecución utilizando el conjunto de datos DS_3.

En la Figura 5, se muestran los resultados despúes de incrementar el número de ítems (de 50 a 100) y el promedio de ítems por *itemsets* (de 3 a 20). En general, todos los algoritmos evaluados muestran buen rendimiento con excepción del algoritmo GSP, el cual no trabaja para umbrales de Soportes menores que el 80 %.

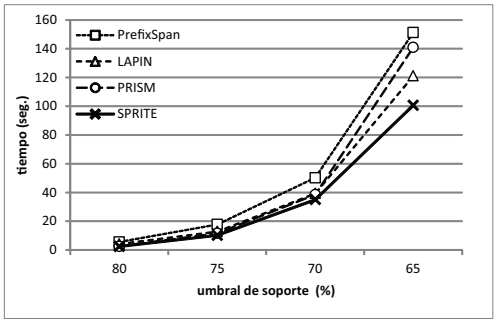

Fig. 5. Tiempo de ejecución utilizando el conjunto de datos DS_4.

En el último experimento solo se incrementó el número de secuencias (de 1000 a 10000) para evaluar la escalabilidad de los algoritmos con respecto al número de secuencias (transacciones). Los resultados mostrados en la Figura 6 son similares a los mostrados en la Figura 5, solo cambió el orden de los algoritmos PRISM y LAPIN.

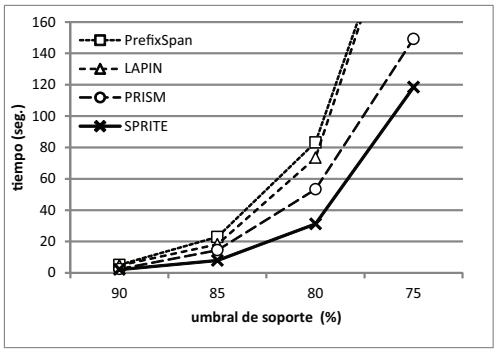

Fig. 6. Tiempo de ejecución utilizando el conjunto de datos DS_5.

En los experimentos realizados, el algoritmo SPRITE mostró el mejor desempeño entre todos los algoritmos evaluados tanto en escalabilidad como en tiempo de ejecución.

6. Conclusiones

En este reporte técnico se propuso un nuevo algoritmo para la obtención de secuencias frecuentes, denominado SPRITE. El algoritmo SPRITE introduce una nueva estructura de datos que incrementa la eficiencia en el proceso de generación de las secuencias candidatas. Adicionalmente, se propone una estrategia de poda basada en la intersección de vectores binarios para reducir la cantidad de secuencias candidatas a analizar.

Los resultados experimentales muestran que el algoritmo SPRITE tiene mejor desempeño que los principales algoritmos del estado del arte como son: GSP, PrefixSpan, LAPIN y PRISM.

Referencias bibliográficas

1. Febrer, J.K., Hernández, J.: Estado del arte para el minado de patrones secuenciales. Technical report, La Habana, Cuba (2011)
2. 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)
3. 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
4. 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
5. M.Parimala, S.Sathiyabama: Spmls : An efficient sequential pattern mining algorithm with candidate generation and frequency testing. International Journal on Computer Science and Engineering **4** (2012) 601–607
6. 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
7. Yang, Z., Wang, Y., Kitsuregawa, M.: Lapin: Effective sequential pattern mining algorithms by last position induction for dense databases. In: DASFAA. (2007) 1020–1023
8. 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
9. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proceedings of 1995 International Conference Data Engineering (ICDE'95). (March 1995) 3–14
10. 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
11. 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
12. Antunes, C., Oliveira, A.L.: Generalization of pattern-growth methods for sequential pattern mining with gap constraints. Springer (2003) 239–251
13. Saputra, D., Rambli, D.R.A., Foong, O.M.: Mining sequential patterns using i-prefixspan. International Journal of Computer Science and Engineering 2 (2008)
14. Ya-Han Hu, Tony Cheng-Kui Huang, H.R.Y.c., Chen, Y.L.: On mining multi-time-interval sequential patterns. Data & Knowledge Engineering (2009)
15. Bhensdadia, C.K., Kosta, Y.P.: An efficient algorithm for mining frequent sequential patterns and emerging patterns with various constraints. International Journal of Soft Computing and Engineering (IJSCE) **1**(6) (January 2012)
16. Shyur., H.J., Jou1, C., Chang, K.: A data mining approach to discovering reliable sequential patterns. The Journal of Systems and Software (2013)
17. Chia-Ying Hsieh, D.L.Y.a.: An efficient sequential pattern mining algorithm based on the 2-sequence matrix. In: 2008 IEEE International Conference on Data Mining Workshops. (2008)
18. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. (2002) 429–435

RT_026, mayo 2014
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. A No. 21406 e/214 y 216, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba