SPaMi-FTS: An Efficient Algorithm for Mining Frequent Sequential Patterns

José Kadir Febrer-Hernández[1], José Hernández-Palancar[1], Raudel Hernández-León[1], and Claudia Feregrino-Uribe[2]

[1] Centro de Aplicaciones de Tecnologías de Avanzada, 7ma A ♯21406 e/ 214 y 216, Siboney, Playa, La Habana, C.P. 12200, Cuba
{jfebrer,jpalancar,rhernandez}@cenatav.co.cu
[2] Instituto Nacional de Astrofísica, Óptica y Electrónica, Luis Enrique Erro ♯1, Sta. Ma. Tonantzintla, Puebla, C.P. 72840, México
cferegrino@ccc.inaoep.mx

Abstract. In this paper, we propose a novel algorithm for mining frequent sequences, called SPaMi-FTS (**S**equential **Pa**ttern **Mi**ning based on **F**requent **T**wo-**S**equences). SPaMi-FTS introduces a new data structure to store the frequent sequences, which together with a new pruning strategy to reduce the number of candidate sequences and a new heuristic to generate them, allows to increase the efficiency of the frequent sequence mining. The experimental results show that the SPaMi-FTS algorithm has better performance than the main algorithms reported to discover frequent sequences.

Keywords: Data mining, Sequential pattern mining, Frequent sequences.

1 Introduction

Frequent sequences mining is a well-known data mining technique that aims to compute all frequent sequences from a transactional dataset. Unlike an itemset, in which an item can occur at most once, in a sequence an itemset can occur multiple times. Additionally, in itemset mining, $(abc) = (cba)$ but in sequences mining, $\langle\ (ab)\ c\ \rangle \neq \langle\ c\ (ab)\ \rangle$.

Mining sequential patterns in transactional datasets, introduced in [1], has been applied in several application areas, for example in web access analysis [2], text mining [3], disease treatments [4], among others.

Let $I = \{i_1, i_2, ..., i_n\}$ be a set of elements, called items. Let DS be a set of transactions, where each transaction is a non empty list (or sequence) $\langle\ \alpha_1\ \alpha_2\ ...\ \alpha_m\ \rangle$, with $\alpha_i \subseteq I$. The size of an itemset is defined as its cardinality, an itemset containing k items is called a k-itemset. Similarly, a sequence containing k itemsets is called a k-sequence.

A sequence $\alpha = \langle\ \alpha_1\ \alpha_2\ ...\ \alpha_n\ \rangle$ is a subsequence of (or contained in) a sequence $\beta = \langle\ \beta_1\ \beta_2\ ...\ \beta_m\ \rangle$ if there exists integers $1 \leq j_1 < j_2 < ... < j_n \leq m$ such that $\alpha_1 \subseteq \beta_{j_1}, \alpha_2 \subseteq \beta_{j_2}\ , ...\ , \alpha_n \subseteq \beta_{j_m}$. Additionally, we will call occurrence position of α in β ($occPos(\alpha, \beta)$) to the set of positions of all possibles β_{j_m} in β. It is valid to clarify that a sequence α can have more than one occurrence position in a sequence β.

The support of a sequence α is the fraction of transactions in DS containing α. A sequence is called frequent if its support is greater than or equal to a given support threshold ($minSup$).

The main contribution of this paper is a novel algorithm for frequent sequences mining, called SPaMi-FTS, which introduces a new heuristic to generate the candidate sequences. Additionally, SPaMi-FTS proposes a new data structure to store the frequent sequences and a new pruning strategy to reduce the number of candidate sequences.

This paper is organized as follows. The next section describes related work. Section 3 introduces the SPaMi-FTS algorithm. In Section 4 the experimental results are shown. Finally, the conclusions are given in Section 5.

2 Related Work

In general, most of the sequential pattern mining algorithms can be separate into two main groups: (1) apriori-like algorithms (AprioriAll, AprioriSome and DynamicSome [1], GSP [7], SPIRIT [5], SPADE [8]) and (2) pattern-growth based algorithms (PrefixSpan [6], LAPIN [11], ESPE [9], PRISM [10]). The algorithms of the first group are based on a generate-and-test technique while the algorithms of the second one are based on the divide and conquer technique.

In [7], the authors proposed the GSP algorithm, which includes time constraints and taxonomies in the mining process. In the experiments, the authors show that the GSP algorithm runs from 2 to 20 times faster than AprioriAll, AprioriSome and DynamicSome algorithms. Following similar ideas, the use of regular expressions as a flexible constraint was introduced in [5], with the SPIRIT algorithm.

The PrefixSpan algorithm, proposed in [6], is based on recursively constructing the patterns by growing on the prefix, and simultaneously, restricting the search to projected datasets. In this way, the search space is reduced at each step, allowing for better performance in the presence of small support thresholds.

The LAPIN (LAst Position INduction) algorithm [11] uses an item-last-position list and a prefix border position set instead of the tree projection or candidate generate-and-test techniques introduced so far. LAPIN checks the last position of an item i to decide if a frequent k-sequence can be extended to be a candidate $(k + 1)$-sequence by appending the item i to it.

The ESPE algorithm, proposed by Hsieh in [9], enumerates all 2-sequences avoiding to generate too many candidates and thus wasting memory space. This algorithm reads a transaction and enumerates all of its 2-sequences. Simple mathematical equations are used to compute the index of all 2-sequences.

PRISM, the algorithm introduced by Gouda et al. in [10], uses a vertical approach for enumeration and support counting, based on the novel notion of primal block encoding, which is based on prime factorization theory.

The algorithm that we propose in this paper follows some ideas from the algorithms LAPIN and ESPE. As LAPIN, our algorithm stores a list of occurrence positions but unlike LAPIN that stores the last position of each single item in each transaction, SPaMi-FTS stores for each frequent 1-sequence α, and for each transaction t, a list with the occurrence positions of α in t. Regarding the 2-sequences, SPaMi-FTS uses them to extend the candidate sequences while ESPE employs them for enumerating all the 2-sequences from each transaction.

3 Our Proposal

In this section, we describe the proposed algorithm (SPaMi-FTS), including the new data structure used to store the frequent sequences with their occurrence lists, and the new pruning strategy used to reduce the number of candidate sequences.

3.1 Storing Useful Information

Let α be a frequent sequence and T be a transactional dataset, the proposed data structure stores, for each $t \in T$, a list L_t with the occurrence positions of α in t. Additionally, a bit-vector of 1's and 0's is stored representing the presence or absence of α in each transaction of T. For example, in Figure 1, five transactions and the bit-vector associated to sequence $\langle f \rangle$ can be observed. Additionally, in Table 1 the occurrence positions of the sequence $\langle f \rangle$ in the transactional dataset of Figure 1 can be seen.

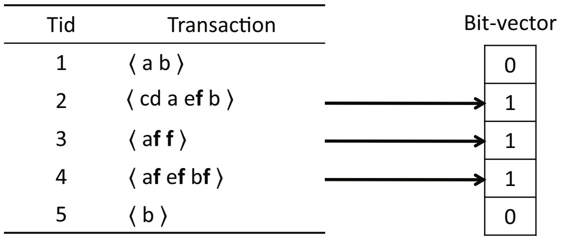


Fig. 1. Example of five transactions and the bit-vector of the sequence $\langle f \rangle$

Table 1. Occurrence positions of sequence $\langle f \rangle$ in the dataset of Figure 1

Tid	Occurrence positions
2	3
3	1, 2
4	1, 2, 3

3.2 SPaMi-FTS Algorithm

Similar to the reported algorithms [1,6,10,11], in a first step, SPaMi-FTS computes all the frequent 1-sequences. As we mentioned above, SPaMi-FTS stores for each frequent 1-sequence α, and for each transaction t, a list with the occurrence positions of α in t. Also a bit-vector representing the presence or absence of α in each transaction is stored.

In a second step, SPaMi-FTS computes the frequent 2-sequences from the frequent 1-sequences. For this, SPaMi-FTS first generates the candidate 2-sequences by combining the 1-sequences obtained in the first step and later, it applies a new pruning strategy to reduce the number of support counting.

Let $\langle i \rangle$ and $\langle j \rangle$ be two frequent 1-sequences, the new pruning strategy intersects (using AND operation) the bit-vectors of $\langle i \rangle$ and $\langle j \rangle$, respectively. The obtained bit-vector stores the highest possible support value (number of bits equal to 1) of the sequence $\langle i\ j \rangle$. If the highest possible support is less than the minimum support threshold then the real support counting of $\langle i\ j \rangle$ is not computed. In the other case, SPaMi-FTS requires to iterate the occurrence lists of $\langle i \rangle$ and $\langle j \rangle$ to compute the real support of $\langle i\ j \rangle$ because it is not enough that sequences $\langle i \rangle$ and $\langle j \rangle$ appear in the same transaction, also it is needed that $occPos(\beta) > occPos(\alpha)$ (see Section 1). The pseudo code to compute the frequent 2-sequences is shown in Algorithm 1, where the method $pruningMethod(\langle i \rangle, \langle j \rangle)$ works as follows: the bit-vectors of $\langle i \rangle$ and $\langle j \rangle$ are intersected to compute the highest possible support value of $\langle i\ j \rangle$, named *highPosSup*. If $highPosSup < minSup$ then the sequence $\langle i\ j \rangle$ is pruned.

Algorithm 1. Pseudo code for computing the frequent $2-sequences$

```
Input: Transactional dataset $T$ and support threshold $minSup$.
Output: Set of frequent $2-sequences$.

$L_1 = oneFrequentSequences(T)$
$L_2 = \emptyset$
foreach $\langle i \rangle \in L_1$ do
    foreach $\langle j \rangle \in L_1$ do
        $prune = pruningMethod(\langle i \rangle, \langle j \rangle)$
        if not $prune$ then
            $Sup = 0$
            foreach $t \in T$ do
                if $occPos(\langle j \rangle, t) > occPos(\langle i \rangle, t)$ then
                    $Sup = Sup + 1$
                end
            end
            if $Sup \geq minSup$ then
                $L_2 = L_2 \cup \{\langle i\ j \rangle\}$
            end
        end
    end
end
return $L_2$
```

Similar to frequent $1-$ *sequences*, SPaMi-FTS stores for each frequent $2-$ *sequence* α, and for each transaction t, a list with the occurrence positions of α in t and additionally, a bit-vector representing the presence or absence of α in each transaction is stored. Before continuing, let us to introduce the following definition:

Definition 1. *Let $\alpha = \langle \alpha_1\ \alpha_2\ ...\ \alpha_m \rangle$ be a frequent $m-sequence$ ($m \geq 2$) and $\beta = \langle \alpha_m\ \beta_1 \rangle$ be a frequent $2-sequence$, we will call the sequence $\langle \alpha_1\ \alpha_2\ ...\ \alpha_m\ \beta_1 \rangle$ the union of α and β, and we will use the operator $\bigoplus$ to indicate this union.*

Finally, in a third stage, the procedure used to compute the frequent $2-$ *sequences* is extended to compute the frequent $k-$ *sequences* ($k > 2$). For this, the candidate $(k+1)-$ *sequences* are obtained by combining each frequent $k-$ *sequence* $\alpha = \langle \alpha_1\ \alpha_2\ ...\ \alpha_k \rangle$ with all frequent $2-$ *sequences* of the form $\beta = \langle \alpha_k\ \beta_1 \rangle$. Later, the proposed pruning strategy is used to reduce the number of support counting. The pseudo code to compute the frequent $k-$ *sequences* ($k > 2$) is shown in Algorithm 2, where the method *pruningMethod*(α, β) works in a similar way to the pruning method of Algorithm 1.

Algorithm 2. Pseudo code to compute the frequent $(k+1)-$ *sequences*

```
Input: Transactional dataset $T$, set of frequent $k-sequences\ kFreq$, set of
       frequent $2-sequences\ twoFreq$ and support threshold $minSup$.
Output: Set of frequent $(k+1)-sequences$.
$L_1 = \emptyset$
$L_2 = \emptyset$
foreach $\alpha = \langle \alpha_1\ \alpha_2\ ...\ \alpha_k \rangle \in kFreq$ do
    foreach $\beta = \langle \alpha_k\ \beta_1 \rangle \in twoFreq$ do
        $prune = pruningMethod(\alpha, \beta)$
        if not $prune$ then
            $Sup = 0$
            foreach $t \in T$ do
                if $occPos(\beta, t) > occPos(\alpha, t)$ then
                    $Sup = Sup + 1$
                end
            end
            if $Sup \geq minSup$ then
                $L_2 = L_2 \cup \{\alpha \bigoplus \beta\}$
            end
        end
    end
end
return $L_2$
```

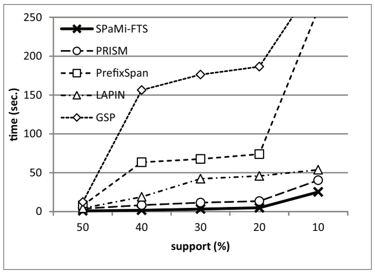
4 Experimental Results

In this section, we present the results of our experimental comparison between SPaMi-FTS and the main sequence mining algorithms reported in the literature: GSP [1], PrefixSpan [6], LAPIN [11] and PRISM [10]. All codes (implemented in ANSI C standard) were provided by their authors. Our experiments were carried out over seven datasets, built with a synthetic dataset generator (see its main parameters in Table 2) developed by the Data Mining Research Group at the Department of Computer Science, University of Illinois ($http:\backslash\backslash illimine.cs.uiuc.edu$).

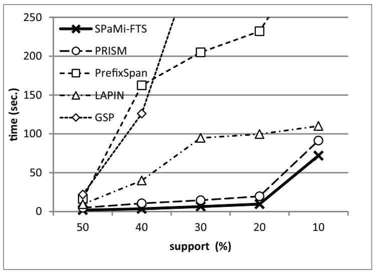
Table 2. Main input parameters for synthetic dataset generator

Parameter	Description
C	average number of itemsets per sequence
T	average number of items per itemset
N	number of items
D	number of transactions (sequences)

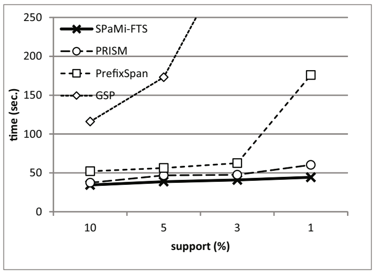
All the tests were performed on a PC with an Intel Core 2 Quad at 2.5 GHz CPU with 4 GB DDR3 RAM, running Windows 7. We considered CPU time, without input and output times, as runtime for all the algorithms compared in this paper. Several experiments were conducted to evaluate the performance of the algorithms when these parameters change (see Fig. 2).



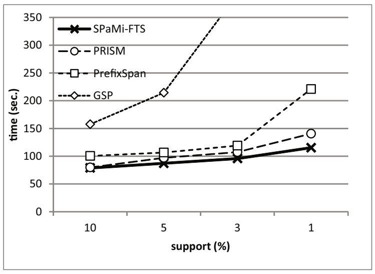
(a) $C = 20, T = 3, N = 20, D = 10000$.



(b) $C = 40, T = 3, N = 20, D = 20000$.



(c) $C = 40, T = 3, N = 50, D = 5000$.



(d) $C = 40, T = 3, N = 50, D = 10000$.

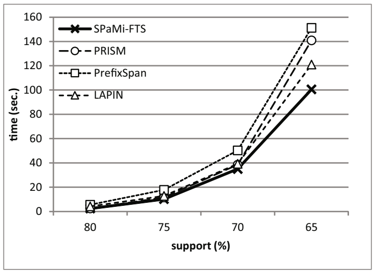
Fig. 2. Runtime comparison using (a) DS_1, (b) DS_2, (c) DS_3 and (d) DS_4 datasets

In the first experiment we used the parameter values most employed in the literature (see their parameter values in Fig. 2(a)). As it can be seen in Fig. 2(a), SPaMi-FTS obtains the best result, followed by PRISM algorithm.

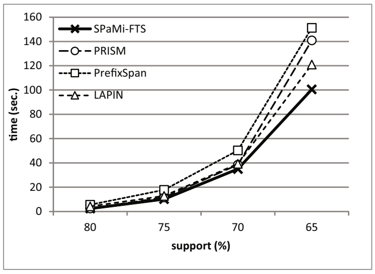
For the second experiment, we simultaneously increased D (from 10000 to 20000) and C (from 20 to 40). In this case, GSP and PrefixSpan were the most affected algorithms, in both cases the runtimes were over 250 seconds (see Fig. 2(b)).

In the third and fourth experiments, we increased N (from 20 to 50) keeping C set to 40 and varying D from 5000 in Fig. 2(c) to 10000 in Fig. 2(d). In both figures, we show the runtime of all evaluated algorithms except for LAPIN algorithm because it crashed with support thresholds under 10%.

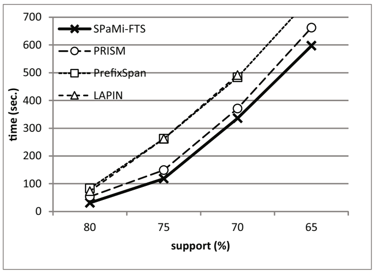
With the last experiments, we studied the scalability of the evaluated algorithms. Three tests were performed with $C = 10$, $T = 20$, $N = 100$ and D set to 1000, 5000 and 10000, respectively. The support thresholds were 80 %, 75 %, 70 % and 65 %. Since GSP failed to run on these datasets, we could not report on its scalability. As it can be seen in Fig. 3, the trend is approximately linear, only changing the order of PRISM and LAPIN algorithms, PRISM beats LAPIN in this case.



(a) $C = 10$, $T = 20$, $N = 100$, $D = 1000$.



(b) $C = 10$, $T = 20$, $N = 100$, $D = 5000$.



(c) $C = 10$, $T = 20$, $N = 100$, $D = 10000$.

Fig. 3. Runtime comparison using (a) DS_5, (b) DS_6 and (c) DS_7 dataset

In general, the SPaMi-FTS algorithm had the best performance of all tested algorithms both in scalability and in runtime.

5 Conclusions

In this paper we have introduced a novel algorithm for mining frequent sequences, called SPaMi-FTS. In order to reach fast candidate sequence generation, SPaMi-FTS introduces a new data structure to store the frequent sequences and a new pruning strategy to reduce the number of candidate sequences. Additionally, SPaMi-FTS proposes a new heuristic to generate the frequent sequences based on the frequent two-sequences. The experiments showed that SPaMi-FTS has better performance than the main algorithms reported in the literature to discover frequent sequences both in scalability and in runtime.

References

1. Agrawal, R., Srikant, R.: Mining Sequential Patterns. In: Agrawal, R., Srikant, R. (eds.) Proceedings of 1995 International Conference Data Engineering, ICDE 1995, pp. 3–14 (1995)
2. Yu, X., Li, M., Gyu Lee, D., Deuk Kim, K., Ho Ryu, K.: Application of Closed Gap-Constrained Sequential Pattern Mining in Web Log Data. Advances in Control and Communication 137, 649–656 (2012)
3. García-Hernández, R.A., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A.: A fast algorithm to find all the maximal frequent sequences in a text. In: Sanfeliu, A., Martínez Trinidad, J.F., Carrasco Ochoa, J.A. (eds.) CIARP 2004. LNCS, vol. 3287, pp. 478–486. Springer, Heidelberg (2004)
4. Liao, V.C.-C., Chen, M.S.: An Efficient Sequential Pattern Mining Algorithm for Motifs with Gap Constraints. In: Proceedings of the 2012 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), vol. 1, pp. 1–1 (2012)
5. Garofalakis, M., Rastogi, R., Shim, K.: SPIRIT: Sequential Pattern Mining with Regular Expression Constraints. In: Proceedings of the 25th International Conference on Very Large Data Bases, pp. 223–234 (1999)
6. Pei, J., Han, J., Mortazavi-asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, pp. 215–224 (2001)
7. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 1–17. Springer, Heidelberg (1996)
8. Zaki, M.J.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning Journal 42, 31–60 (2001)
9. Ying Hsieh, C., Lin Yang, D., Wu, J.: An Efficient Sequential Pattern Mining Algorithm Based on the 2-Sequence Matrix. In: IEEE International Conference on Data Mining Workshops, pp. 583–591 (2008)
10. Gouda, K., Hassaan, M., Zaki, M.J.: PRISM: A Prime-Encoding Approach for Frequent Sequence Mining. Journal of Computer and System Sciences 76, 88–102 (2010)
11. Yang, Z., Wang, Y., Kitsuregawa, M.: LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction for Dense Databases, pp. 1020–1023 (2007)