Binary Sequences and Association Graphs for Fast Detection of Sequential Patterns

Selim Mimaroglu*, Dan A. Simovici**

* Bahcesehir University, Istanbul, Turkey,selim.mimaroglu@gmail.com
**University of Massachusetts Boston, Massachusetts 02125, USA, dsim@cs.umb.edu

Abstract. We develop an efficient algorithm for detecting frequent patterns that occur in sequence databases under certain constraints. By combining the use of bit vector representations of sequence databases with association graphs we achieve superior time and low memory usage based on a considerable reduction of the number of candidate patterns.

1 Introduction

Mining sequential patterns was originally proposed in Agrawal and Srikant (1995), where three algorithms, (*AprioriAll*, *AprioriSome*, and *DynamicSome*) were introduced. *PrefixSpan*, based on the prefix projection idea, was introduced in Pei et al. (2001). SPADE Zaki (2001) performs space efficient joins on prefix-based equivalence classes. *PRISM* Gouda et al. (2007), uses prime number encoding for support counting. A related but distinct problem (discussed in Mannila et al. (1997)) is finding frequent episodes in very long sequences. *SPAM* Ayres et al. (2002) finds sequential patterns using a bitmap representation. An extension of SPAM, which incorporates gap and regular expression constraints was achieved in Ho et al. (2005). The GSP algorithm Srikant and Agrawal (1996) is similar to *AprioriAll*; additionally it can handle three types of constraints: minimum and maximum gap between consecutive elements of a sequence (referred to as *min_gap* and *max_gap*), and window size between rows. When *min_gap* $= 0$, *max_gap* $= \infty$, and *window_size* $= 0$, the sequential patterns found by GSP are the classical sequential patterns as introduced in Agrawal and Srikant (1995). The algorithm *cSPADE* Zaki (2000) introduces similar constraints, and it is implemented on top of *SPADE*. *SPIRIT* Garofalakis et al. (1999) is more general than both *GSP* and *cSPADE* as it deals with regular expression constraints.

In this note we describe SPAG, an algorithm that combines the dual use of bit vector representations of sequence databases with association graphs to achieve superior performance in identifying patterns in sequences.

2 Apriori Frameworks on Sequence Sets

We refer the reader to Simovici and Djeraba (2008) for mathematical concepts and notations. Let I be a set of items, and let $\mathbf{Seq}(I)$ be the set of sequences of items of I. We consider a a graded poset $(P, \leq, h)$, where $P \subseteq \mathbf{Seq}(I)$, and $h : P \longrightarrow \mathbb{N}$, referred to as the *set of patterns*, and a *data set* $\mathcal{D}$ defined as a sequence of sequences, $\mathcal{D} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\} \subseteq \mathbf{Seq}(\mathbf{Seq}(I))$. A *sequence Apriori framework* is a triple $((P, \leq, h), \mathcal{D}, \sigma)$, where σ is a relation between patterns and data, such that $\mathbf{t} \leq \mathbf{t}'$ and $(\mathbf{t}', \mathbf{s}) \in \sigma$ implies $(\mathbf{t}, \mathbf{s}) \in \sigma$.

If $\mathbf{x}$ is a subsequence of $\mathbf{y}$, we denote this by $\mathbf{x} \sqsubseteq \mathbf{y}$. For $\mathbf{y} \in \mathcal{D}$, and $\mathbf{t} \in P$ we define several partial functions form $\phi : (\mathbf{Seq}(I))^2 \longrightarrow \mathbb{R}$ such that $\mathrm{Dom}(\phi) = \{(\mathbf{t}, \mathbf{y}) \in (\mathbf{Seq}(I))^2 \mid \mathbf{t} \sqsubseteq \mathbf{y}\}$: $\alpha(\mathbf{t}, \mathbf{y}) = \ell(\mathbf{y}) - \ell_{\mathbf{y}}(\mathbf{t})$, $\beta(\mathbf{t}, \mathbf{y}) = \ell_{\mathbf{y}}(\mathbf{t}) - \ell(\mathbf{t})$, and $\omega(\mathbf{t}, \mathbf{y})$ as the number of occurrences of $\mathbf{t}$ in $\mathbf{y}$, for $t, y \in \mathbf{Seq}(I)$.

The function $\alpha(\mathbf{t}, \mathbf{y})$ measures the *outer* gap of $\mathbf{t}$ in $\mathbf{y}$, while $\beta(\mathbf{t}, \mathbf{y})$ measures the *inner* gap of $\mathbf{t}$ in $\mathbf{y}$. For a fixed $\mathbf{y}$, if $\mathbf{t}$ is scattered in $\mathbf{y}$ then $\alpha(\mathbf{t}, \mathbf{y})$ is relatively small, and $\beta(\mathbf{t}, \mathbf{y})$ is relatively large. On another hand, if $\mathbf{t}$ is condensed in $\mathbf{y}$ then $\alpha(\mathbf{t}, \mathbf{y})$ is relatively large, and $\beta(\mathbf{t}, \mathbf{y})$ is relatively small. For example, if $\mathbf{t} = a_1 a_2$, and $\mathbf{y} = a_1 a_3 a_4 a_5 a_2$, we have $\alpha(\mathbf{t}, \mathbf{y}) = 0$, and $\beta(\mathbf{t}, \mathbf{y}) = 3$. If $\mathbf{y}' = a_3 a_4 a_1 a_2 a_5$, then $\alpha(\mathbf{t}, \mathbf{y}') = 3$ and $\beta(\mathbf{t}, \mathbf{y}') = 0$. The values of both functions α, and β are changed as expected, because $\mathbf{t}$ is condensed in $\mathbf{y}'$.

Let $\mathbf{y}$ be a database sequence, $\mathbf{u}, \mathbf{v}$ be two sequences such that $\mathbf{u}$ is an infix of $\mathbf{v}$ and suppose that $\mathbf{v} \sqsubseteq \mathbf{y}$. If ϕ is any of the partial functions α or ω then $\phi(\mathbf{u}, \mathbf{y}) \geq \phi(\mathbf{v}, \mathbf{y})$. We also have $\beta(\mathbf{v}, \mathbf{y}) \geq \beta(\mathbf{u}, \mathbf{y})$.

Let $\sigma_{min,k}$ be the relation that consists of those pairs $(\mathbf{v}, \mathbf{y})$ for which there exists an occurrence of $\mathbf{v}$ in $\mathbf{y}$ such that the least gap between two consecutive symbols of $\mathbf{v}$ is at least k. Similarly, if $\sigma_{max,k}$ consists of pairs $(\mathbf{v}, \mathbf{y})$ for which there is an occurrence of $\mathbf{v}$ in $\mathbf{y}$ such that the largest gap between two consecutive symbols of $\mathbf{v}$ is at most k. These relations were introduced in Srikant and Agrawal (1996).

The relations $\sigma_{\alpha,k}$ and $\sigma_{\beta,k}$ are given by $\sigma_{\alpha,k} = \{(\mathbf{v}, \mathbf{y}) \in (\mathbf{Seq}(I))^2 \mid \mathbf{v} \sqsubseteq \mathbf{y} \text{ and } \alpha(\mathbf{v}, \mathbf{y}) \geq k\}$ and by $\sigma_{\beta,k} = \{(\mathbf{v}, \mathbf{y}) \in (\mathbf{Seq}(I))^2 \mid \mathbf{v} \sqsubseteq \mathbf{y} \text{ and } \beta(\mathbf{v}, \mathbf{y}) \leq k\}$. It is easy to see that $\sigma_{min,k}$, $\sigma_{max,k}$, $\sigma_{\alpha,k}$, and $\sigma_{\beta,k}$ are Apriori relations.

If σ is an Apriori relation and $\mathbf{u}$ is an infix of $\mathbf{v}$, then $\mathsf{supp}_{\mathcal{D},\sigma}(\mathbf{v}) \leq \mathsf{supp}_{\mathcal{D},\sigma}(\mathbf{u})$. This allows a straightforward extension of the well-known Apriori algorithm to an algorithm defined on sequences.

3 Association Graphs and Bit Vectors

Storing bit vectors is very space-efficient and bitwise operations are very fast, which allows storing large databases into memory.

Each distinct item i in the database is represented by a bit vector, denoted by *ibv*$_i$ (item bit vector), which contains as many bits as the number of rows in the database. If i is present in the j^{th} row, the j^{th} entry of *ibv*$_i$ is 1, and is 0 otherwise.

Each row of a table that contains m distinct items is represented by a collection of m row bit vectors *rbv*$_1, \ldots,$ *rbv*$_m$ whose length l equals the length of the row. If *rbv*$_j = (b_{j1}, \ldots, b_{jl})$ then

$$b_{jh} = \begin{cases} 1 & \text{if item } i_j \text{ occurs at position } h, \\ 0 & \text{otherwise,} \end{cases}$$

for $1 \leq h \leq l$.

The dual use of item and row bit vectors speeds up the mining process considerably by providing fast support count. A similar bit vector representation is used in PRISM Gouda et al. (2007), where prime number encoding, and integer operations are used instead of binary operations.

Next we define the association graph of a sequence database.

Definition 3.1 Let $\mathcal{D} = (\mathbf{s}_1, \ldots, \mathbf{s}_N)$ be a sequence database on the set of items I, *min_sup* be the minimum support count, and σ be an Apriori relation. The *association graph of* $\mathcal{D}$,

$AG = (V, E)$ is a labeled directed graph defined as follows. The set of vertices V consists of those items i such that $|\{j | (i, \mathbf{s}_j) \in \sigma$ and $i \sqsubseteq \mathbf{s}_j, \mathbf{s}_j \in \mathcal{D},$ for $1 \leq j \leq N\}| \geq$ *min_sup*.

The set of edges E consists of those pairs $(i, i') \in V \times V$ such that $|\{j | (ii', \mathbf{s}_j) \in \sigma$ and $ii' \sqsubseteq \mathbf{s}_j, \mathbf{s}_j \in \mathcal{D},$ for $1 \leq j \leq N\}| \geq$ *min_sup*. An edge (i, i') is labeled by a sequence $L(i, i') = (\tau_1, \ldots, \tau_N)$, where $\tau_p = 1$ if $(ii', \mathbf{s}_p) \in \sigma$, and $ii' \sqsubseteq \mathbf{s}_p$, and $\tau_p = 0$, otherwise, for $1 \leq p \leq N$. □

```
Input: Database: $\mathcal{D}$ in ibv and rbv format, minimum support count: min_sup, Apriori
       relation: $\sigma$
Output: The Association graph, $AG = (V, E)$
// Create the vertices
foreach item $i$ do
    // On each row such that $ibv_i$ is 1, check constraint
    if $suppcount_{\mathcal{D},\sigma}(i) \geq min\_sup$ then
        $V = V \cup \{i\}$ ;
// Create the edges with labels
foreach $i \in V$ do
    foreach $i' \in V$ do
        $ibv_T = ibv_i \cap ibv_{i'}$ ;
        if $ibv_T.count() \geq min\_sup$ then
            // On each row such that $ibv_T$ is 1, use the row bit
               vectors of $i$ and $i'$ to compute
            if $suppcount_{\mathcal{D},\sigma}(ii') \geq min\_sup$ then
                $E = E \cup \{(i, i')\}$ ;
                label $(i, i')$ by $ibv_{ii'}$ ;
            // On each row such that $ibv_T$ is 1, use the row bit
               vectors of $i$ and $i'$ to compute
            if $suppcount_{\mathcal{D},\sigma}(i'i) \geq min\_sup$ then
                $E = E \cup \{(i', i)\}$ ;
                label $(i', i)$ by $ibv_{i'i}$ ;
output $AG$;
```

FIG. 1: Association Graph Construction Algorithm

In the C++ implementation of SPAG, *KFrequentC* class represents the frequent sequences and stores the members of the sequence, a bit vector which is set to 1 at the row indexes containing this sequence, and for each row containing this sequence (there may be more than one occurrence), the end position(s) and the length(s).

To extend a sequence $\mathbf{p} \in F_i$ there must be an edge from the last member of $\mathbf{p}$ to a frequent item in the association graph. Since $\mathbf{p} \in F_i$, the length of $\mathbf{p}$ is i. If there exists an edge from p_i, the last component of $\mathbf{p}$, to some other frequent item z, then $(p_i, z) \in E$ in the association graph AG. Initially, $ibv_{\mathbf{p}z}$ is obtained as $ibv_{\mathbf{p}z} = ibv_{\mathbf{p}} \cap ibv_{p_i z}$. We must have $count(ibv_{\mathbf{p}z}) \geq min\_sup$.

```
  Input: Database: $\mathcal{D}$, minimum support count $min\_sup$, Apriori relation: $\sigma$
  Output: Frequent Sequences
1 Read the database $\mathcal{D}$, convert the items in $ibv$ format, and convert the rows in $rbv$ format
  ;
2 Create the association graph $AG = (V, E)$ by using the Association Graph Construction
  Algorithm ;
3 $F_1 = V$ ;
4 $F_2 = \{ii' | (i, i') \in E\}$ ;
5 $i = 2$ ;
6 while $F_i \neq \emptyset$ do
7     $C_{i+1} = \{\mathbf{p}z | \mathbf{p} \in F_i, (p_i, z) \in E, |ibv_{\mathbf{p}z} = ibv_{\mathbf{p}} \cap ibv_{p_i z}| \geq min\_sup, prune_{\sigma}(\mathbf{p}z)\}$ ;
      // Use row bit vectors of z, and information at
         KFrequentC of p to compute following
8     Compute $F_{i+1}$ from $C_{i+1}$ ;
9     $i++$ ;
```

FIG. 2: Sequential Pattern Mining with Association Graph (SPAG) Algorithm

Since σ is an Apriori relation, all sequences $\mathbf{q}$ such that $\mathbf{q}$ is an infix of $\mathbf{p}z$ must be frequent. If any $\mathbf{q}$ violates this condition, it means that $\mathbf{p}z$ can not be frequent either, so $prune_{\sigma}(\mathbf{p}z)$ returns *false*. In Figure 2 these steps are represented in line 7. Following this, actual frequent sequences are computed from the candidate set as shown in line 8 of Figure 2. Row bit vectors of z and the end positions of $\mathbf{p}$ are used to make sure that z follows $\mathbf{p}$. Rows violating this order or the constraint σ are set to 0 in $ibv_{\mathbf{p}z}$. Checking the order and the constraint σ has to be done on every row of the database where $ibv_{\mathbf{p}z}$ is initially set to 1. At the end of this procedure, if the $count(ibv_{\mathbf{p}z}) \geq min\_sup$, then $\mathbf{p}z$ is placed into F_{i+1}.

Both Ayres et al. (2002) and Ho et al. (2005) use bit vectors and operations on bit vectors. SPAG combines the use of bit vectors with association graphs and allows the use of vital global information. Finding all occurrences of the items in a data set is immediate by using item bit vectors. In addition, association graphs handle what can follow an item in a data set; this improves the candidate pattern generation process considerably. Neither Ayres et al. (2002) nor Ho et al. (2005) have the global information provided by our dual use of item bit vectors and association graphs; therefore, they must keep track of each row. Moreover, the tremendous amount of bit vector transformations make both Ayres et al. (2002) and Ho et al. (2005) memory-inefficient. SPAG avoids these difficulties by working on the original bit vectors without modifying them.

4 Experimental Results

Extensive experiments were conducted on synthetically generated sequential data sets using a Pentium 3.0GHz computer having 4GB of main memory running on Linux. The results of SPAG are compared with SPADE Zaki (2001), the implementation of Ho et al. (2005) (which we refer to as cSPAM), and with cSPADE Zaki (2000).

It is shown in Zaki (2001) that SPADE outperforms GSP by using space efficient joins. Although cSPAM is not very space efficient, it is fast because it uses bit vectors. cSPAM can also handle constraints. cSPADE is implemented on top of SPADE; it can handle some constraints.

Experimental results show that SPAG outperforms SPADE, cSPADE and cSPAM in almost every support level for every data set.

In Figures 3(a)-3(c) we show execution times for cSPAM, SPADE and SPAG are shown for synthetic databases that range from $1,000$ to $100,000$ rows, having 7 items per row on average, and a total of 30 distinct items. SPAG has superior performance on this data set for every support level. On all these data sets SPAG outperforms SPADE. For only one support level cSPAM outperforms SPAG. cSPAM and SPAG are comparable in two cases, and for the remaining 13 cases SPAG performs much better than cSPAM. Experiments are expanded to include a larger data set of 100,000 rows having 1,000 unique items and 50 items per row on average. Test results for this data set are shown in Fig 3(d). In most cases SPAG outperforms SPADE by a factor of 10. Fig. 3(e) shows that SPAG and cSPADE perform similarly for the minimum gap constraint. SPAG is 4 times faster than cSPAM in most cases. SPAG clearly outperforms both cSPAM, and cSPADE considerably for the maximum gap constraint, as illustrated in Figure 3(f).

5 Conclusions and Future Work

We present SPAG, an efficient algorithm for detecting frequent patterns which combines the dual use of bit vector representations of sequence databases with association graphs. Experimental results show that SPAG is faster than the other state-of-the-art algorithms (cSPAM, SPADE, and cSPADE) both with constraints and without constraints. We believe that the SPAG has a broader applicability and its use for determining variability patterns or to frequency patterns shall be investigated.

References

Agrawal, R. and R. Srikant (1995). Mining sequential patterns. *Proceedings of the Eleventh International Conference on Data Engineering*, 3–14.

Ayres, J., J. Flannick, J. Gehrke, and T. Yiu (2002). Sequential pattern mining using a bitmap representation. In *KDD*, pp. 429–435.

Garofalakis, M. N., R. Rastogi, and K. Shim (1999). Spirit: Sequential pattern mining with regular expression constraints. In *VLDB*, pp. 223–234.

Gouda, K., M. Hassaan, and M. J. Zaki (2007). Prism: A prime-encoding approach for frequent sequence mining. *IEEE Int. Conference on Data Mining*.

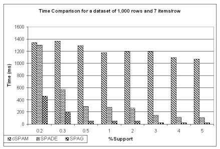
Ho, J., L. Lukov, and S. Chawla (2005). Sequential pattern mining with constraints on large protein databases. In *Proceedings of the 12th International Conference on Management of Data (COMAD 2005b)*, pp. 89–100. Computer Society of India.

Mannila, H., H. Toivonen, and A. Inkeri Verkamo (1997). Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery 1*(3), 259–289.

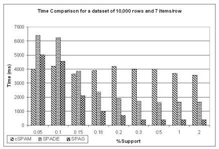
Pei, J., J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu (2001). PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern. *IEEE Int. Conference on Data Engineering*.

Simovici, D. A. and C. Djeraba (2008). *Mathematical Tools for Data Mining – Set Theory, Partially Ordered Sets, Combinatorics*. London: Springer-Verlag.

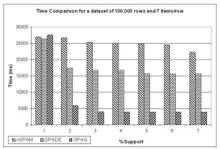
Srikant, R. and R. Agrawal (1996). Mining sequential patterns: Generalizations and performance improvements. In *EDBT*, pp. 3–17.



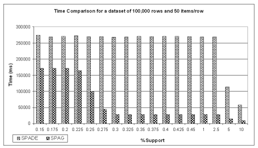
(a) Small database



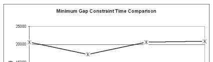
(b) Medium database

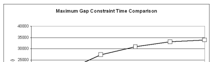


(c) Large database



(d) Very large database





Zaki, M. J. (2000). Sequence mining in categorical domains: Incorporating constraints. In *CIKM*, pp. 422–429.

Zaki, M. J. (2001). Spade: An efficient algorithm for mining frequent sequences. *Machine Learning 42*(1/2), 31–60.

Résumé

Nous développons un algorithme efficace pour détecter des motifs fréquents qui se produisent dans des bases de données séquentielles sous certaines contraintes. En combinant l'utilisation des représentations des bases de données séquentielles par séquences binaires avec des graphes d'association, nous obtenons un meilleur temps et une utilisation moins grande de la mémoire basée sur une réduction considérable du nombre des motifs candidates.