LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction for Dense Databases

Zhenglu Yang, Yitong Wang, and Masaru Kitsuregawa

Institute of Industrial Science, The University of Tokyo
4-6-1 Komaba, Meguro-Ku, Tokyo 153-8305, Japan
{yangzl,ytwang,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. Sequential pattern mining is very important because it is the basis of many applications. Although there has been a great deal of effort on sequential pattern mining in recent years, its performance is still far from satisfactory because of two main challenges: large search spaces and the ineffectiveness in handling dense datasets. To offer a solution to the above challenges, we have proposed a series of novel algorithms, called the LAst Position INduction (LAPIN) sequential pattern mining, which is based on the simple idea that the last position of an item, α, is the key to judging whether or not a frequent k-length sequential pattern can be extended to be a frequent (k+1)-length pattern by appending the item α to it. LAPIN can largely reduce the search space during the mining process, and is very effective in mining dense datasets. Our performance study demonstrates that LAPIN outperforms PrefixSpan [4] by up to an order of magnitude on long pattern dense datasets.

1 Introduction

Sequential pattern mining, which extracts frequent subsequences from a sequence database, has attracted a great deal of interest during the recent surge in data mining research because it is the basis of many applications. Efficient sequential pattern mining methodologies have been studied extensively in many related problems, including the basic sequential pattern mining [1] [6] [4], constraint-based sequential pattern mining [2], maximal and closed sequential pattern mining [3].

Although there are many problems related to sequential pattern mining, we realize that the basic sequential pattern mining algorithm development is the most fundamental one because all the others can benefit from the strategies it employs, i.e. Apriori heuristic and projection-based pattern growth. Therefore we aim to develop an efficient basic sequential pattern mining algorithm in this paper.

1.1 Overview of Our Algorithm

For any sequence database, the last position of an item is the key used to judge whether or not the item can be appended to a given prefix (k-length) sequence.

Example 1. We will use the sequence database S shown in Fig. 1 (a) with min_support = 2 as our running example in this paper. When scanning the database in Fig. 1 (a) for

R. Kotagiri et al. (Eds.): DASFAA 2007, LNCS 4443, pp. 1020–1023, 2007.

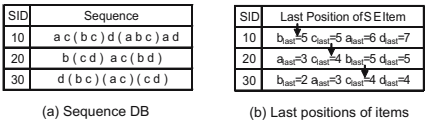

Fig. 1. Sample database

the first time, we obtain Fig. 1 (b), which is a list of the last positions of the 1-length frequent sequences in ascending order. Suppose that we have a prefix frequent sequence $\langle a \rangle$, and its positions in Fig. 1 (a) are 10:1, 20:3, 30:3, where sid:eid represents the sequence ID and the element ID. Then, we check Fig. 1 (b) to obtain the first indices whose positions are larger than $\langle a \rangle$'s, resulting in 10:1, 20:2, 30:3, i.e., (10:b$_{last}$ = 5, 20:c$_{last}$ = 4, and 30:c$_{last}$ = 4), symbolized as "$\downarrow$". We start from these indices to the end of each sequence, and increment the support of each passed item, resulting in $\langle a \rangle : 1, \langle b \rangle : 2, \langle c \rangle : 3$, and $\langle d \rangle : 3$, from which, we can determine that $\langle ab \rangle$, $\langle ac \rangle$ and $\langle ad \rangle$ are the frequent patterns. The $I\text{-}Step$ methodology is similar to the $S\text{-}Step$ methodology, which is not described here due to limited space.

Let $\bar{D}$ be the average number of customers (i.e., sequences) in the projected DB, $\bar{L}$ be the average sequence length in the projected DB, $\bar{N}$ be the average total number of the distinct items in the projected DB, and m be the distinct item recurrence rate or density in the projected DB. Then m=$\bar{L}/\bar{N}$ ($m \geq 1$), and the relationship between the runtime of PrefixSpan (T_{ps}) and the runtime of LAPIN (T_{lapin}) in the support counting part is

$$T_{ps}/T_{lapin} = (\bar{D} \times \bar{L})/(\bar{D} \times \bar{N}) = m \tag{1}$$

Because support counting is usually the most costly step in the entire mining process, Eq.(1) illustrates the main reason why LAPIN is faster than PrefixSpan for dense datasets, whose m (density) can be very high.

2 LAPIN Sequential Pattern Mining

In this section, we describe the LAPIN algorithms in detail. Refer [5] for the notations and lemmas used in this paper. The pseudo code of LAPIN is shown in Fig. 2.

In step 1, by scanning the DB once, we obtain the SE position list table and all the 1-length frequent patterns. At the same time, we can get the SE $item\text{-}last\text{-}position$ $list$, as shown in Fig. 1 (b). In function $Gen\_Pattern$, we obtain the position list of the last item of α, and then perform a binary search in the list for the (k-1)-length prefix border position (step 3). Step 4, shown in Fig. 2, is used to find the frequent SE (k+1)-length pattern based on the frequent k-length pattern and the 1-length candidate items. We can test each candidate item in the local candidate item list ($LCI\text{-}oriented$), which is similar to the method used in SPADE [6]. Another choice is to test the candidate item in the projected DB, just as PrefixSpan [4] does ($Suffix\text{-}oriented$).

We found that $LCI\text{-}oriented$ and $Suffix\text{-}oriented$ have their own advantages for different types of datasets. Thus we formed a series of algorithms categorized into two classes, LAPIN_LCI and LAPIN _Suffix. Please refer [5] for detail.

```
Input: A sequence database, and the minimum support threshold, ε
Output: The complete set of sequential patterns

Function: Gen_Pattern(α, S, CanI_s, CanI_i)
Parameters: α = length k frequent sequential pattern; S = prefix
         border position set of (k-1)-length sequential pattern;
         CanI_s = candidate sequence extension item list of (k+1)
         -length sequential pattern; CanI_i = candidate itemset
         extension item list of (k+1)-length sequential pattern
Goal: Generate (k+1)-length frequent sequential pattern

Main():
1. Scan DB once to do:
  1.1 P_s ← Create the position list representation of the 1-
           length SE sequences
  1.2 B_s ← Find the frequent 1-length SE sequences
  1.3 L_s ← Obtain the item-last-position list of the 1-length
           SE sequences
2. For each frequent SE sequence α_s in B_s
  2.1 Call Gen_Pattern (α_s, 0, B_s, B_i)

Function: Gen_Pattern(α, S, CanI_s, CanI_i)
3. S_α ← Find the prefix border position set of α based on S
4. FreItem_{s,α} ←Obtain SE item list of α based on CanI_s and S_α
5. For each item γ_s in FreItem_{s,α}
  5.1 Combine α and γ_s as SE, results in θ and output
  5.2 Call Gen_Pattern (θ, S_α, FreItem_{s,α}, FreItem_{i,α})
```

Fig. 2. LAPIN algorithm pseudo code

3 Performance Study

We conducted experiments on synthetic and real life datasets to compare LAPIN with PrefixSpan. We used a 1.6 GHz Intel Pentium(R)M PC with 1G memory. Refer [1] for the meaning of the different parameters used to generate the datasets. We first compared PrefixSpan and our algorithms using several small-, medium-, and large- sized datasets. The statistics of the datasets is shown in Fig. 3 (a).

Fig. 3 (b) and Fig. 3 (c) show the running time and the searched space comparison between PrefixSpan and LAPIN and clearly illustrate that PrefixSpan is slower than LAPIN using the medium dataset and the large dataset. This is because the searched spaces of the two datasets in PrefixSpan were much larger than that in LAPIN. For the small dataset, the initial overhead needed to set up meant that LAPIN was slower than PrefixSpan. LAPIN_Suffix is faster than LAPIN_LCI for small datasets because the former searches smaller spaces than the latter does. However, for medium and large dense datasets, LAPIN_LCI is faster than LAPIN_Suffix because the situation is reversed. The memory usage of the algorithms is shown in Fig. 3 (d).

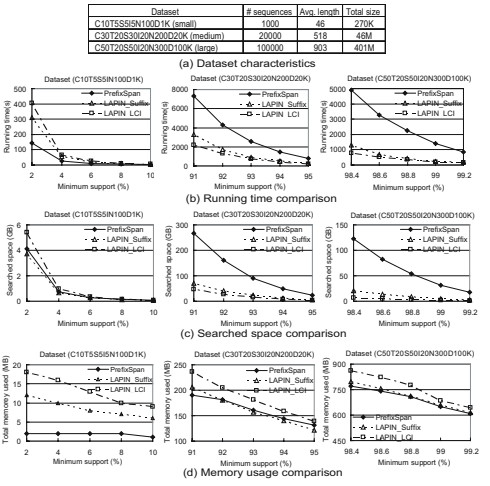

Fig. 3. The different sizes of the datasets

Different parameters analysis. When C increases, T increases, and N decreases, then the performance of LAPIN improves even more relative to PrefixSpan, by up to an order of magnitude. The reason is that on keeping the other parameters constant, increasing C, T and decreasing N, respectively, will result in an increase in the distinct item recurrence rate, m.

4 Conclusions

We have proposed a series of novel algorithms, LAPIN, for efficient sequential pattern mining. By thorough experiments, we have demonstrated that LAPIN outperforms PrefixSpan by up to an order of magnitude on long dense datasets.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pp. 3-14, 1995.
2. M.N. Garofalakis, R. Rastogi and K. Shim. SPIRIT: Sequential PAttern Mining with Regular Expression Constraints. In *VLDB*, pp. 223-234, 1999.
3. C. Luo and S.M. Chung. Efficient Mining of Maximal Sequential Patterns Using Multiple Samples. In *SDM*, pp. 64-72, 2005.
4. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. Mining Sequential Patterns by Pattern-growth: The PrefixSpan Approach. In *TKDE*, Volume 16, Number 11, pp. 1424-1440, 2004.
5. Z. Yang, Y. Wang, and M. Kitsuregawa. LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction. *Technical Report*, Info. and Comm. Eng. Dept., Tokyo University, 2005. http://www.tkl.iis.u-tokyo.ac.jp/∼yangzl/Document/LAPIN.pdf
6. M. J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. In *Machine Learning*, Vol. 40, pp. 31-60, 2001.