# Gradual Evolution of Sequential Sequence Mining for Customer relation database

Kiran Amin
Assoc. Prof. & Head, Department of Computer Engineering,
U. V. Patel College of Engineering, Kherva, Gujarat
E-mail:kiran.amin@ganpatuniversity.ac.in

J. S. Shah
Prof. & Head, Department of Computer Engg.,
L.D. College of Engineering, Ahmedabad
E-mail:jssld@yahoo.com

***Abstract*— The sequential sequence mining takes time interval between various transactions. Here we have discussed various techniques for finding large sequence from the historical database. Based on various methods the sequence will be useful to predict and plan various the strategies. Here inclusion of time interval between the items to be purchased will be useful. This paper focuses on the various techniques to generate large sequence.**

***Keywords- Time Interval, Pseudo Projection, Sequential Sequence Mining, User defined Support***

## I. INTRODUCTION

The Sequential Sequence Mining is useful in data mining to find the useful sequence from the large sequence. We have compared various methods by considering various parameters like memory, sequences and time intervals. The time interval sequential mining is challenging. The Association Rule Mining finds only frequent patterns fails with time interval gap. A sequential sequence mining finds the useful sequences with various timing of customers' purchase.

## II. ASSOCIATION RULE MINING

Association rule mining finds frequent sequences with given user defined support value. Association Rule Mining is used to find the frequent sequence which occurred in maximal way. Sequence mining is to find out the frequent sequence which occurred in maximum no of sequences[1]. Here example 1 shows the small dataset of customers' transactions. The SID against various sequences are given.

| SID | Sequences |
|---|---|
| 1 | <ab(bcd)(efg)ad> |
| 2 | <abcd> |
| 3 | <ab> |
| 4 | <abc> |

Example 1

The sequences represent in <…> brackets. Here for SID 1, we find out the Sequence <ab(bcd)(efg)ad>. Here a,b,c,d,e,f are the item code. The (…) bracket indicates that these items are bought by the customer at same time means in single transaction. No bracket is shown for the customer buys a single item in transaction. Similarly all transactions are shown with various SID. We have 5 transactions for SID=1. ($1^{st}$ transaction is a, $2^{nd}$ truncation we have 2 items b and c, $3^{rd}$ transaction we have e and f, $4^{th}$ transaction we have a, and last $5^{th}$ transaction we have d). Support in sequence mining is indicating that sequence occurred in how many SIDs. The support for item 'a' is 2 because it is occurred in SID 1 and 3. First sequence mining concept found by the Srikant and Agrawal in IBM research center. In that they found various algorithms for the sequence mining then lots of research was done in the field of sequential mining. The association rule mining gives only the frequent sub-sequence but it doesn't give the time interval between successive items.

The new method is useful to find time interval sequence patterns. It doesn't only find the sub-sequences but also gives the time interval between two successive items. Now consider Example 2 in which time interval is added between various transactions.

| SID | Sequences |
|---|---|
| 1 | <(a,2)(bc,4)(ef,7)(a,8)(d,9)> |
| 2 | <(b,4)(c,6)(d,7)> |
| 3 | <(a,2)(d,3)(b,6)> |
| 4 | <(c,2),(d,4)> |

Example 2

Here in Example 2 you can see that the item or itemset you will find the time stamp. Here (a,2), means item 'a' is occurred at time stamp 2.

Now here we will see the real life example for Association Rule Mining,

a) Having bought a printer, a customer will come back to buy a paper and then a toner.
While Time interval sequential mining
b) Having bought a laser printer, a customer will come back to buy a scanner in three months and then a CD burner in six months.

Now we will see the some of the algorithms for the sequential mining.

Given two sequences $\alpha=< a_1, a_2 \dots a_n>$ and $\beta=< b_1 b_2 \dots b_m>$. $\alpha$ is called a subsequence of $\beta$, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}, \dots, a_n \subseteq b_{jn}$. $\beta$ is a super sequence of $\alpha$. Here <abf> is the subsequence of the <a(bcd)(efg)ad>.

The length of a sequence is the number of itemsets in the sequence. A sequence of length *k* is called a *k*-sequence. i.e

Candidate 1-subsequences:

$<i_1>, <i_2>, <i_3>, \dots, <i_n>$

Candidate 2-subsequences:

$<i_1, i_2>, <i_1, i_3>, \dots, <(i_1\ i_1)>, <(i_1\ i_2)>, \dots, <(i_{n-1}\ i_n)>$

**AprioriAll**

It was developed by the Srikant and Agawal in IBM research lab. It finds all the frequent subsequence which will satisfy the minimum support threshold. Here also subsequence item sets will be generated with the help of the candidate itemsets. Also we have to scan dataset every time to fine out *k*-large sequences. AprioriALL have five phases and at the end of all the phases

**1. Sort Phase :** First the database is sorted, with customer-id as the major key and transaction-time as the minor key. This step converts the dataset in the sequential order as shown in Figure 1.

| Customer Id | TransactionTime | Items Bought |
|---|---|---|
| 1 | June 25 '93 | 30 |
| 1 | June 30 '93 | 90 |
| 2 | June 10 '93 | 10, 20 |
| 2 | June 15 '93 | 30 |
| 2 | June 20 '93 | 40, 60, 70 |
| 3 | June 25 '93 | 30, 50, 70 |
| 4 | June 25 '93 | 30 |
| 4 | June 30 '93 | 40, 70 |
| 4 | July 25 '93 | 90 |
| 5 | June 12 '93 | 90 |

Figure 1

**2. LitemsetPhase :** In this phase It finds the set of all Litemsets (Large itemset) L. It also simultaneously finds the set of all large 1-sequences. Here the data are arranged in form of sequential dataset. It combines all items by particular customer. i.e. here you can see that customer 1 buy the item 30 and 90 in different transaction and we show it <(30)(90)>. Here '<…>' will indicate as a sequence id or customer ID and

'(…)' will indicate the 1 transaction. So here, we continue with our older example and note that we have minimum support threshold id 2(40%).

| Customer Id | Customer Sequence |
|---|---|
| 1 | ⟨(30) (90)⟩ |
| 2 | ⟨(10 20) (30) (40 60 70)⟩ |
| 3 | ⟨(30 50 70)⟩ |
| 4 | ⟨(30) (40 70) (90)⟩ |
| 5 | ⟨(90)⟩ |

⇨

| Large Itemsets | Mapped To |
|---|---|
| (30) | 1 |
| (40) | 2 |
| (70) | 3 |
| (40 70) | 4 |
| (90) | 5 |

min_sup_count=2

Figure 2

**3.Transformation Phase:** In a transformed customer sequence, each transaction is replaced by the set of all litemsets contained in that transaction. If a transaction does not contain any litemset, it is not retained in the transformed sequence.

| Customer Id | Original Customer Sequence | Transformed Customer Sequence | After Mapping |
|---|---|---|---|
| 1 | ⟨(30) (90)⟩ | ⟨{(30)} {(90)}⟩ | ⟨{1} {5}⟩ |
| 2 | ⟨(10 20) (30) (40 60 70)⟩ | ⟨{(30)} {(40), (70), (40 70)}⟩ | ⟨{1} {2, 3, 4}⟩ |
| 3 | ⟨(30 50 70)⟩ | ⟨{(30), (70)}⟩ | ⟨{1, 3}⟩ |
| 4 | ⟨(30) (40 70) (90)⟩ | ⟨{(30)} {(40), (70), (40 70)} {(90)}⟩ | ⟨{1} {2, 3, 4} {5}⟩ |
| 5 | ⟨(90)⟩ | ⟨{(90)}⟩ | ⟨{5}⟩ |

Figure 3

**4.Sequence Phase:** It scans the dataset multiple times. In each scan, we start with a seed set of large sequences. At the end of the scan, it determines which of the candidate sequences are actually frequent. These large candidates become the seed for the next scan.

We have two families of algorithms, which we call count-all and count-some. The count-all algorithms count all the large sequences, including non-maximal sequences. The non-maximal sequences must then be pruned out (in the maximal phase). We present one count-all algorithm, called AprioriAll, based on the Apriori algorithm. We present two count-some algorithms: AprioriSome and DynamicSome.

**5.Maximal Phase :** It finds the maximal sequences among the set of large sequences. Having found the set of all large sequences S in the sequence phase, the following algorithm can be used for finding maximal sequences. Let the length of the longest sequence be n. Then,

| Sequence | Support |
|---|---|
| ⟨1 2 3 4⟩ | 2 |
| ⟨1 3 5⟩ | 2 |
| ⟨4 5⟩ | 2 |

Figure 4

Here we show that how Apriori works. Now we will just take over view of AproriSome [1] and DynamicApriori [1]. AprioriSome algorithm runs in forward and backward pass. In forward pass, It only counts sequence of certain lengths. For example, we might count sequences of length 1, 2, 4 and 6 in the forward pass and count sequences of length 3 and 5 in the backward pass. It saves to the time by not count those sub-sequences which are not maximum. But some time s we required all the frequent sub-sequences rather than only max-subsequences. So it also saves the time and memory.

DynamicSome algorithm is work as same as the AprioriSome. Just difference is in generating the candidate sequence. The candidate sequences that are counted, is determined by the variable step. In the initialization phase,

all the candidate sequences of length upto and including step are counted. Then in the forward phase, all sequences whose lengths are multiples of step are counted. Thus, with step set to 3, we will count sequences of lengths 1, 2, and 3 in the initialization phase, and 6,9,12,... in the forward phase.

However, unlike in AprioriSome, these candidate sequences were not generated in the forward phase. The intermediate phase generates them. Then the backward phase is identical to the one for AprioriSome.

**Comparison**

DynamicSome performs worse than the other two algorithms mainly because it generates and counts a much larger number of candidates in the forward phase because it will include those candidate sequence also which subsequences are not frequent and large. The major advantage of AprioriSome over AprioriAll is that it avoids counting many non-maximal sequences. However, this advantage is reduced because of two reasons. First, candidates $C_k$ in AprioriAll are generated using $L_{k-1}$, whereas AprioriSome sometimes uses $C_{k-1}$ for this purpose. Since $L_{k-1} \subseteq C_{k-1}$, the number of candidates generated using AprioriSome can be larger. Second, although AprioriSome skips over counting candidates of some lengths, they are generated nonetheless and stay memory resident. For lower supports, there are longer large sequences, and hence more non-maximal sequences, and AprioriSome does better. That means for the lower number of customer AprioriSome perform better than the AprioriAll.

After sometime they introduced the GSP algorithm [2]. However it has some advantages over AprioriAll. The performs better over following limitations.

1. **Absence of time constraints**

Users often want to specify maximum and/or minimum time gaps between adjacent elements of the sequential pattern. For example, a book club probably does not care if someone bought “C++ book", followed by“ JAVA book" three years later; they may want to specify that a customer should support a sequential pattern only if adjacent elements occur within a specified time interval, say three months. (So for a customer to support this pattern, the customer should have bought “JAVA book" within three months of buying “C++ book".)

2. **Rigid definition of a transaction**

For many applications, it does not matter if items in an element of a sequential pattern were present in two different transactions, as long as the transaction-times of those transactions are within some small time window. That is, each element of the pattern can be contained in the union of the items bought in a set of transactions, as long as the difference between the maximum and minimum transaction-times is less than the size of a sliding time window. For example, if the book-club specifies a time window of a week, a customer who ordered the “C++ book" on Monday, “C book" on Saturday, and then “JAVA book" and “HTML book" in a single order a few weeks later would still support the pattern “`C++ book' and `C book', followed by `JAVA book' and `HTML book' ".

**3. Absence of taxonomies**

Many datasets have a user-defined taxonomy (is-a hierarchy) over the items in the data, and users want to find patterns that include items across different levels of the taxonomy.

**Improved PrefixSpan(I-PrefixSpan)**

Dhanysaputra come with the improve version of the PrefixSpan[5]. It is called I-PrefixSpan( improved PrefixSpan). This algorithm improves PrefixSpan in two ways: (1) it implements sufficient data structure for Seq-Tree framework to build the in-memory database sequence and to construct the index set which help to reduce the load on the main memory, and (2) instead of keeping the whole in-memory database, it implements separator database to store the transaction alteration signs. Seq-Tree framework with sufficient data structure is the other contribution in I-PrefixSpan which is used to store in-memory sequence database and to construct the index set. Seq-Tree is a general tree with two certain characteristics: (1) all leaves must be located at the same depth and (2) the height of the tree is at least 2. It is shown in Figure 5.

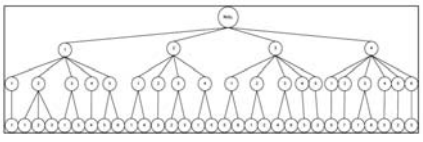

Figure 5

**I-PrefixSpan**

1st it was introduced by dr. Yen Chen in 2003[7]. He finds the sequential patterns that include time intervals, called time-interval sequential patterns. This work develops two efficient algorithms for mining time-interval sequential patterns. So we will learn it with example. Let us see,

Here you can see small dataset which will use to find the Time Interval Sequential Mining.

| Sid | Sequence |
|---|---|
| 10 | ( (a,1) , (c,3) , (a,4) , (b,4) , (a,6) , (e,6) , (c,10) ) |
| 20 | ( (d,5) , (a,7) , (b,7) , (e,7) , (d,9) , (e,9) , (c,14) , (d,14) ) |
| 30 | ( (a,8) , (b,8) , (e,11) , (d,13) , (b,16) , (c,16) , (c,20) ) |
| 40 | ( (b,15) , (f,17) , (e,18) , (b,22) , (c,22) ) |

Time interval Sequence dataset

It works same as the typical PrefixSpan but have to deal with the Time Interval. So here we have to take all the possible projection of frequent items. In the first step we will follow the same step like original PrefixSpan. Find out the 1-length sequences. So here we will scan the dataset (table-1) and find out the 1-sequences. If support are 50% and TI= $\{I_0, I_1, I_2, I_3\}$, where $I_0 : t = 0$, $I_1 : 0 < t \leq 3$, $I_2 : 3 < t \leq 6$ and $I_3 : 6 < t \leq \infty$, then we find out the frequent items are (a), (b), (c), (d) and (e). We eliminate the (f) as it is not frequent and not pass the minimum support thresh hold. Now we have to make projection table for all the frequent 1-sequence. Let us take 'a' as the frequent items and make projection table base on that. Here you can see that 'a' item occurred 3 time in the 1st sequence so we have to take make projection for the all the 3 times because we deal with the time interval along with the sequence. We can use different notation for that so we can identify. We use [*Sid* :*Pos*]**,** where *Sid* is the identifier of the sequence and *Pos*is the position of item. Now we have 5 postfix sequences in the projected database and they are as follow.

[10:1] ((*c*, 3)(*a*, 4)(*b*, 4)(*a*, 6)(*e*, 6)(*c*, 10)),

[10:4] ((*b*, 4)(*a*, 6)(*e*, 6)(*c*, 10)),

[10:6] ((*e*, 6)(*c*, 10)),

[20:7] ((*b*, 7)(*e*, 7) (*d*, 9)(*e*, 9)(*c*, 14)(*d*, 14)),

[30:8] ((*b*, 8)(*e*, 11)(*d*, 13)(*b*, 16)(*c*, 16), (*c*, 20) ).

Now have to scan this projected dataset again and have to find out the frequent sequences. Now we have to make the table for that. In that table 1st column indicated the time interval and 1st raw will indicate the frequent items. Remaining cell we show that frequency of data occurrence with particular time interval in the projected dataset and base on that we will get the 2-length sequences base on table-2.

As we can see from the table that cell belongs to B-$I_0$ are frequent because it satisfy the minimum support (which is 2 here). So we can say that (a, $I_0$, b) is the frequent sequence. That means customer come to by item 'b' after $I_0$ time of purchase 'a'

**References:**


[1] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. 1995 Int'l Conf. Data Eng. (ICDE '95), pp. 3-14, Mar. 1995.

[2] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96), pp. 3-17, Mar. 1996.

[3] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu.Freespan: Frequent pattern-projected sequential pattern mining. In *Proc. 2000 Int. Conf. KnowledgeDiscovery and Data Mining (KDD'00)*, pages 355–359, Boston, MA, Aug. 2000.

[4] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," Proc. 2001 Int'l Conf. Data Eng. (ICDE '01), pp. 215-224, Apr. 2001.

[5] Dhany , Saputra and RambliDayang, R.A. and Foong, Oi Mean (2007) Mining Sequential Patterns Using I-PrefixSpan. In: World Academy of Science, Engineering and Technology , 14-16 December, 2007.

[6] Chen, Y.L., Chiang, M.C. and Ko, M.T. (2003). "Discovering time- interval sequential patterns in sequence databases," Expert Syst. Appl., Vol. 25, No. 3, (pp. 343-354).

AUTHOR PROFILE

Name Prof. Kiran R. Amin
Designation: Head & Associate Professor, Computer Engineering
Place of working : U. V. Patel College of Engineering, Ganpat Vidyanagar, Dist : Mehsana, Gujarat India
Email : kiran.amin@ganpatuniversity.ac.in