A Survey On Sequential Pattern Mining

Erhan Sezerer
232630003
erhansezerer@iyte.edu.tr

Abstract

In this paper, I will Introduce Sequential Pattern Mining, a subcategory of frequent pattern mining. Unlike frequent pattern mining methods which finds the inter-class frequencies, sequential pattern mining aims at finding intra-class frequencies which gives information about a specific customer's behaviour or gene patterns for a specific DNA and/or RNA.

Given a sequence database *D* with an order(temporal order,genom order, etc.), the task of sequential pattern mining is to find all subsequences whose frequency is above a certain minimum supprt threshold specified by the user. There are various algorithms that use various methods like Apriori, pattern-growth and vertical bitmap resolution to find such subsequences. In this paper, all of these methods will be introduced and the advantages and disadvantages of each of them will be discussed.

Introduction

Sequential pattern mining(SPM) has a very broad area of applications, including retail purchases, biologic data, software bug detection and so on. In these areas, SPM aims to discover frequent events that appear in some order. In retail customer databases this order is the temporal order of transactions that belong to a single user. In software bug detention the order is also the temporal order of events which lead to a bug or a crash of a system. However the order can also be the order of proteins and genes in a DNA/RNA, in biological data.

Problem Definitions

Let $i_1, i_2, ..., i_n$ = items. A **sequence** s is denoted by $\langle s_1 s_2 s_3 ... s_k \rangle$ where, $s_1 s_2 s_3 ... s_k$ are itemsets which can be denoted as $(i_1 i_2 ... i_l)$. we can omit the paranthesis If s i has only 1 item, for more convenience. Number of instances of items in a sequence is called the **length** and a sequence with a length k is called **k-sequence**. i.e., for customer purchase databases, the sequence $\langle a(ab)ac(ac) \rangle$ is a 7-sequence where the customer have made 5 different transactions over time and bought 7 items. Here the items (ab) in paranthesis belong to the same transaction and same time stamp. These transactions are also called **elements**.Note that items may not be unique throughout the sequence since generic items such as bread and milk can be bought on daily basis, but the items are unique inside a single transaction. So, even if a customer buys 2 of the same item at the same time, only a single instance will exist in a single transaction. This means that a transaction like this, $\langle (aa)(bc)(de) \rangle$, is not valid.

A sequence $\alpha = \langle a_1 a_2 a_3 ... a_n \rangle$ is called the **subsequence** of the sequence $\beta = \langle b_1 b_2 b_3 ... b_m \rangle$ if and only if there exist integers $1 \leq k_1 \leq k_2 \leq ... \leq k_n \leq m$ Such that $a_1 \subseteq b_{k1}, a_2 \subseteq b_{k2}, ..., a_n \subseteq b_{kn}$. For example, all of these sequences, $\langle aac \rangle, \langle (ab)a \rangle, \langle a(ab)acc \rangle$ are subsequences of the sequence $\langle a(ab)ac(ac) \rangle$, but the sequences, $\langle aacb \rangle$, $\langle (abc)a \rangle$ are not.

Problem Statement

Given a sequence database with tuples $\langle sid, s \rangle$, such as the one in Table 1, and a user defined *min-support* threshold, find all frequent sub-sequences(sequential patterns) in the database. For a customer sales database, we want to find out sequence of the form "if a user buys a laptop computer, there will be %80 percent chance that he/she will buy a mouse in one week". We can use this information to reorganise the shelves and pre-order the items that will most likely to be sold for not to lose a potential customer. If we know that "if Google's stock drops by %5 then the microsoft's stock prices also drops", then we can decide which stock options to buy or sell. Same type of question can be answered by using SPM on biological data, software bug detention and so on.

Rest of the paper is organized as follows: In Chapter 2 I will talk about the type of algorithms and their classifications used in SPM, in Chapter 3 I will describe each of the state of the art algorithms for each method and finally in 4rt Chapter I will discuss the advantages and disadvantages of each algorithm on the others.

Sequential Pattern Mining Algorithms

SPM algorithm can be represented in roughly three categories: Apriori, pattern-growth, pattern-growth with vertical bitmap representation. Furthermore concepts like close SPM, multi dimensional SPM are also implemented in various researches. State of the art algorithms for each category can be seen from the Table 2.

In next subsections I will describe each of these algorithms in detail.

Table 1: A Sequence Database Example

sid	*sequence*
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ad)c(bc)(ae)\rangle$
30	$\langle (ef)(ab)(df)cb\rangle$
40	$\langle eg(af)cbc\rangle$

Table 2: Algorithm Classifications

Method	Algorithms
Apriori	AprioriAll
Apriori	GSP
Pattern-Growth	FreeSpan
Pattern-Growth	PrefixSpan
Pattern-Growth with Vertical Bitmap Representation	SPAM
Pattern-Growth with Vertical Bitmap Representation	LAPIN-SPAM
Close SPM	CloSpan

Apriori Algorithms

In this section I will describe two state of the art Apriori algorithms: AprioriAll (Agrawal et al, 1995) and GSP (Agrawal et al, 1996).

AprioriAll AprioriAll is an algorithm for finding maximal frequent sequential Patterns(SP). In this algorithm SP's are found by generating candidate sets from smaller sequences. Each candidate k-sequence are genrated by joining (k-1)-sequences. The Algorithm is divided into 5 phases: sort phase, Large Itemset (Litemset) phase, transformation phase, sequence phase, maximal phase.

The difference of this algorithm from the traditional apriori methods lies in the litemset and transformation phases. Intead of using and searching for items, the author proposed the litemsets which are mapped to each itemset found in the previous iterations of the algorithm. In this way, instead of performing a complicated and computationally expensive subsequence matching, we just have to find the corresponding id of the litemset we are counting.

In order to create such a mapping we have to find and include every combination of the items in a certain element and add them to the transformed sequence database. Table 3 shows an example database and its transformation. Note that the infrequent items such as 10 and 20 are pruned from the transformed sequence database to further improve the performance of the algorithm.

In sequence phase all the candidates are generated from previous sequences and their supports are counted using the mapping described above. Finally in Maximal phase, all the frequent sequences which are not maximal are eliminated from the list and all maximal sets are returned to the user. If we remove the final phase, this algorithm can also be used for finding all frequent sets.

GSP In GSP, the authors have improved the AprioriAll algorithm by adding some constraints to the frequent sequence searching. Those improvements are: time constraints, sliding time windows and taxonomies. Instead of other SPM algorithms, in this paper, authors considers an element as a time window instead of a single transaction. For example if the time window is set to one week then everything a customer buys in a week will be considered as the same transaction. Vice versa if the time gap between an element and another is too long then that sequence will not be considered as a subsequence. For example, as the authors state; “A book club probably does not care if someone bought ‘Foundation’, followed by a ‘Foundation and Empire’ three years later”. Both of these time constraints are included in GSP algorithm as a user defined thresholds.

In addition, authors also improved the standard SPM algorithms by adding taxonomies. For example a customer who have bought “Foundation” then “Foundation and Empire” would also support “Asimov” then “Foundation and Empire”. This leads to finding more usefull association rules.

For finding all frequent sequential patterns of items and their taxonomies, GSP algorithm introduces two distintive pruning methods on top of the previous algorithms.

First pruning method is to use **contiguous subsequences** at candidate generation phase. Contiguous subsequences can be defined by (Agrawal et al, 1996):

Definition: Given a sequence $s = \langle s_1 s_2 ... s_n\rangle$ and a subsequence c, c is a *contiguous* subsequence of s if any of the following conditions hold:

- c is derived from s by dropping an item from either s_1 or s_n
- c is derived from s by dropping an item from an element S_i which has at least 2 items
- c is a contiguous subsequence of c' and c' is a contiguous subsequence of s

GSP prunes out any candidate k-sequence from the candidate set, if one of the contiguous subsequence of that k-sequence does not exist in frequent (k-1)-sequences. This comes from the fact that if a contiguous subsequence is not frequenti then its superset cannot be frequent either.

Second powerfull pruning mechanism in GSP is to construct (k+1)-sequence candidate sequences from frequent (k-1)-sequences, instead of generating them from frequent k-

Table 3: Transformed Database in Apriori All Algorithm

customer	*sequence*	*transformed Sequence*	*after Mapping*
1	⟨ (30) (90)⟩	⟨ {(30)}{(90)} e	⟨ {(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}⟩

sequences. For example, if we find sequence ⟨(1 3) (4 5)⟩ to be frequent generating its frequent subsequence ⟨ (1 3)⟩ , then we don't have to count the support of the subsequences ⟨(1 3) (4)⟩ and ⟨(1 3) (4)⟩ since we know that they are frequent, using their superset.

Pattern-Growth Algorithms

There are two successfull pattern-growth algorithms that will be mentioned here: FreeSpan (Han et al, 2000) and PrefixSpan (Han et al, 2001)

FreeSpan FreeSpan is a depth-first search algorithm that uses the projected databases to generate larger sequences.

FreeSpan algorthm first finds the frequent items in the database by scanning the entire database once. Then, in the descending order of frequency, FreeSpan algorithm constrcuts the projected databases. First projected database consists of only the item i_1, second one consists of items i_1 and i_2 but not does not have the rest of the items and so on. An example database and its projection with minimum support threshold as 2 can be seen in Figure 1. FreeSpan algorithm recursively searches for the frequent subsequences by creating further projection if necessary. In order to do that FreeSpan makes another scan of the projected database and finds the items which appear more than the threshold(which are called *annotations*) and further projects the database for those frequent items. For example in Figure 1 in a,b projected database both a and b are frequent, so the algorithm creates the ab-projection bb-projection and so on. Also note that, FreeSpan prunes from the projected databases each item whose support is lower than the minimum support threshold.

For making fewer database passes for support counting, FreeSpan uses S-matrix. S-matrix holds the frequency of three different subsequences in each index. For example if $M[i, j] = (2, 2, 0)$ it means that the support of $\langle ij \rangle$ = 2, support of $\langle ji \rangle$ = 2 and the support of $\langle (ij) \rangle$ = 0. Instead of scanning the database for each item separately, we can scan the database once with these s-matrices and create projected databases using the values in them.

PrefixSpan Similar to the FreeSpan algorithm PrefixSpan also uses the advantages S-matrix and projected databases. But, it also improves it by changing the projection mechanism and adding more pruning.

The authors have added 3-way apriori checking as a pruning mechanism in this algrithm. 3-way Apriori checking works as follows: If $\langle ac \rangle$ is infrequent, exclude c from $\langle ab \rangle$-projection since we know that it cannot be frequent.

In PrefixSpan, instead of creating projected databases for a, a and b, and so on, they create prefix-projections. Similar to the FreeSpan,it starts with finding every frequent item in database. This time for each item a projected database is created using that item as a prefix and converting each sequence on the database with the postfix. Figure 2 shows an example database and prefix-projected databases it creates. Possible advantages or disadvantages of using prefixes intead of projections made in FreeSpan are discussed in next section.

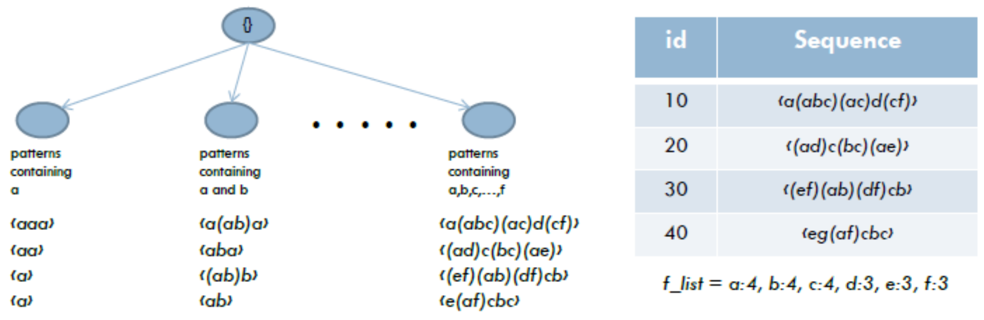
Pattern-Growth Algorithms with Vertical Bitmap Representations

SPAM (Ayres et al, 2002) and LAPIN-SPAM (Yang et al, 2015) are two of the most successful algorithms that ,mplements vertical bitmap represetation of sequences.

SPAM SPAM algorithm is a depth first search algorithm that creates a bitmap of items and sequences where a one means that the item exists in the transaction and a zero means it does not. A typical transaction database and a bitmap representation looks like the one in the Table 4

With these bitmap representation of transaction and items, now the generation of candidates is just a simple ANDing of two columns of bitmaps. If we want to create a candidate sequence i.e. ({a},{b}) only thing we have to do is take AND of {a} column and {b} column in the Table 4. Likewise, for support counting we don't have to match the subsequences, instead we only have to check whether a transaction has all zeros or not. If for a customer c, the bitmap of {a} contains all zeros, that means that c did not buy any item a, therefore such a sequence {a} does not hold for c. Otherwise, if the bitmap does not contain all zeros we can add one to the support count of {a}.

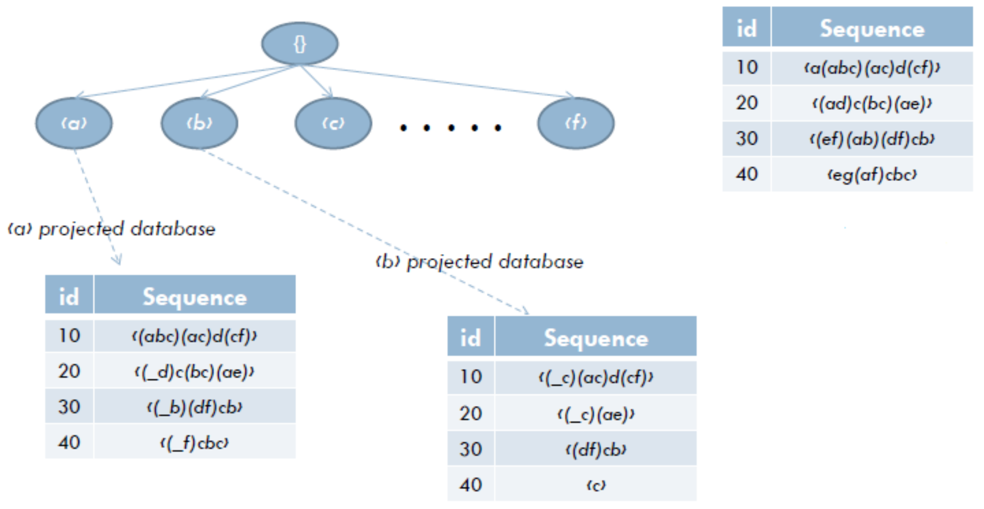
LAPIN-SPAM LAPIN-SPAM is an improved version SPAM, by adding a position check for bitmaps. Position checking works as follows: instead of ANDing the bitmap for generating longer sequences or counting the supports, LAPIN-SPAM first cehcks whether the item we are trying the add is before the last position of the tranaction we currently are. For example, if we have a customer sequence ⟨ ac(bd)c(ab) ⟩ and we are at the position 5 (which means we already constructed the subsequence, let's say ⟨acc⟩) we do not AND the current sequence (⟨acc⟩) with item d, since we already get past that position with second c. This leads us to use less ANDing operations and less zero checking. Detailed comparisons of vertical bitmap representation algorithms can be found in the next chapter.



id	Sequence
10	⟨a(abc)(ac)d(cf)⟩
20	⟨(ad)c(bc)(ae)⟩
30	⟨(ef)(ab)(df)cb⟩
40	⟨eg(af)cbc⟩

f_list = a:4, b:4, c:4, d:3, e:3, f:3

Figure 1: Database Projection in FreeSpan



id	Sequence
10	⟨a(abc)(ac)d(cf)⟩
20	⟨(ad)c(bc)(ae)⟩
30	⟨(ef)(ab)(df)cb⟩
40	⟨eg(af)cbc⟩

id	Sequence
10	⟨(abc)(ac)d(cf)⟩
20	⟨(_d)c(bc)(ae)⟩
30	⟨(_b)(df)cb⟩
40	⟨(_f)cbc⟩

id	Sequence
10	⟨(_c)(ac)d(cf)⟩
20	⟨(_c)(ae)⟩
30	⟨(df)cb⟩
40	⟨c⟩

Figure 2: Database Projection in PrefixSpan

Closed Sequential Pattern Algorithms

Only algorithm we will investigate here is the CloSpan Algorithm (Yan et al, 2006).

CloSpan CloSpan algorithm, which is built on the PrefixSpan, aims to find closed frequent sequential patterns instead of all frequent sequential patterns. Since we can represent and recreate all of the frequent patterns with closed patterns, CloSpan is able to find all frequent sequences without searching for them all.

Major advantage of CloSpan algorithm lies on its ability to prune projected databases without even constructing them at the first place. In order to do that CloSpan uses the closeness property: If every sequence in ⟨ a ⟩ projected database consists of the item f, then it means that ⟨ af ⟩ has the same support as ⟨ a ⟩ and ⟨ f ⟩, which in turns mean that ⟨ a ⟩ and ⟨ f ⟩ cannot be closed, and so does not the all of the sequences that has the prefix ⟨ a ⟩. So , using this information we can prune out the entire ⟨ f ⟩ branch of the tree.

After finding all of the frequent sequences with pruning, CloSpan eliminates the remaining non-closed sequences and finds the closed frequent sequences.

Comparison of Algorithms

Although the GSP algorithm handles much more constraint than the AprioriAll algorithm, it is still faster than AprioriAll thanks to its powerfull pruning techniques. both of the pruning techniques introduced in GSP, helps us get rid of a lot of unnecessary counting operation by pruning the candidate sequences whose contiguous subsequences are found to be infrequent and by not support-counting any k-sequence whose superset is found to be frequent by generating (k+2)-sequences at each step of the algorithm. As the authors shows in GSP, GSP outperforms AprioriAll by a slight margin in all of the tests. Furthermore, as the authors pointed out, AprioriAll algorithm nearly doubles the disk size by transforming the database into a sequence database, while the GSP algorithm does not sufer from such a bad space complexity.

No matter how much performance gain can GSP get, it still performs much worse than the pattern-growth algorithms. This is mostly due to the fact that candidate generation and testing takes too much time since we have to generate and count the supports of exponentially many sub-

Table 4: A Transaction Database and Its Bitmap Representation

customerID	*transactionID*	*{a}*	*{b}*	*{c}*	*{d}*
1	1	1	1	0	1
1	3	0	1	1	1
1	6	0	1	1	1
2	2	0	1	0	0
2	4	1	1	1	0
3	5	1	1	0	0
3	7	0	1	1	1

sequences. Also, Apriori methods are not very good when the frequent subsequences starts to get longer. While pattern-growth methods, thanks to their depth-first search strategy, are finding the longer sequences faster, Apriori methods keep struggling with shorter sequences first before even begininnig to examine longer ones.

In (Han et al, 2001), authors show that PrefixSpan outperforms FreeSpan by a big margin. This is mostly due to the projection mechanisms, althuogh the pruning mechanism added in PrefixSpan also have an effect. In FreeSpan, as you can see in Figure 1, although the leaves on leftmost side of the tree have shrunk projected databases, rightmost leaves of the tree have almost the same projected database with the original ones. So, the projection mechanism in FreeSpan does not shrink the database as much as the projection mechanism in PrefixSpan, and this leads to more search and count operations. In return, it leads to huge performance gains in favor of PrefixSpan.

In CloSpan, authors aim to reduce the computation time even further by finding the closed frequent sequences instead of all frequent sequences which lead to a performance increase by an order of magnitude. This is due to the fact that a lot of the search space is pruned because of their non-closedness. As author ponted out, CloSpan also decreases the space complexity since it does not have to generate as much projected database as the PrefixSpan algorithm.

Although the pattern-growth algorithms improve the performance by orders of magnitude, they are still slower than the vertical bitmap representation algorithms. This is because instead of performing computationally expensive sequence matching and concatenating for support counting and pattern growing respectively, all SPAM does is logical operations like ANDing and zero checking. Since these logical operations will have much less load on the CPU, the algorithms are significantly faster than the pattern-growth algorithms like FreeSpan and PrefixSpan. However, as the author of SPAM noted, if the database is small then the perfromance of SPAM algorithm drops below the PrefixSpan. This is due to the fact that the overhead of creating bitmaps outweights the performance gains by using less CPU heavy operations.

LAPIN-SPAM algorithm further increases the performance gains of vertical bitmap representation algorithms by 2 to 3 times thanks to its last position search. The last position searching accomplishes this by reducing the number of ANDing operations on transaction, thus reducing the CPU load.

Conclusion

There are various methods for finding frequent sequential patterns including Apriori, pattern-growth, pattern-growth with vertical bitmap representation and finally closed SPM. Although all of them has their advantages and disadvantages respectfully, according to the research best performances are obtained by using vertical bitmap representations unless the database is too small. However to get rid of the redundant sequences one might use the closed SPM to get shorter sequences with more meaning.

References

Agrawal, R and Srikant, R (1995) Mining Sequential Patterns. In: Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan. March 1995

Agrawal, R and Srikant, R (1996) Mining Sequential Patterns: Generalizations and Performance Improvements. In: Proceedings of the 5th Intl Conf Extending Database Technology (EDBT'96), Avignon, France. March 1996, pp 3-17

Han, J and Pei, J and Mortazav,-Asl, R and Chen, Q and Dayal, U andHsu, M.S (2000)FreeSpan: Frequent Pattern-Projected Sequential PAttern Mining. In: Proc. 2000 ACM-SIGMOD Int. COnf. Management of Data (SIGMOD'00), pp 355-359, May 2000

Han, J and Pei, J and Mortazav,-Asl, R and Chen, Q and Dayal, U andHsu, M.S (2001) PrefixSpan: Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In ICDE 2001, pp 215-226, April 2001

Ayres, J and Flannick, J and Gehrke, J. and Yiu, T (2002) Sequential Pattern Mining Using Bitmap Representation. In ACM SIGKDD Conference, pp 429-435, 2002

Yang, Z and Kitsuregawa M. (2005) LAPIN-SPAM: an Improved Algorithm for Mining Sequential Patterns. In Proceeding of the 21st International Conference on Data Engineering (ICDE'05), 2005

Yan, X and Han, J and Afshar, R (2003) CloSpan: Closed Sequential Patterns in Large Datasets. 2003 SIAM conference on Data Mining, 2003