An Algorithm for Mining Large Sequences in Databases

Bharat Bhasker, Indian Institute of Management, Lucknow, India, bhasker@iiml.ac.in

ABSTRACT

Frequent sequence mining is a fundamental and essential operation in the process of discovering the sequential rules. Most of the sequence mining algorithms use apriori methodology or build the larger sequences from smaller patterns, a bottom-up approach. In this paper, we present an algorithm that uses top-down approach for mining long sequences. Our algorithm defines dominancy of the sequences and uses it for minimizing the scanning of the data set.

Keywords: Sequential Patterns, Data Mining, Large Sequence Mining

1 Introduction

The sequential pattern mining problem [1,2,3,4,5] introduced by Agrawal and Srikant [1] has been of great interest for researchers as it has a very wide scope of applications spanning from the predicting the customer purchase patterns, and scientific discoveries.

For example, given a time stamped sequences of purchases made by customers, where each event is made up of list of items purchased at the same timestamp by a customer. The objective of the sequential pattern mining algorithm is to discover all the frequent sequences. A sequence is called a frequent sequence provided it occurs more frequently than the minimum support specified by users.

There are several heuristics such as GSP[1], SPADE [3], PrefixSpan [2] and the SPIRIT [4] that try to discover the frequent patterns in a efficient manner by trying to prune a large number of subsequences and thus reduce the search space. The GSP algorithm [1] uses the anti-monotone property (all subsequences of a frequent sequence are also frequent). The SPADE finds frequent sequences using the lattice search [3] and intersection based approach. In this approach the sequence database is transformed to a vertical format. The candidate sequences are divided in groups. The frequent sequences are enumerated in SPADE using both methods – breadth first and depth first methods. The support is counted for the generated sequences. The basic approach of te above three algorithm can be classified as the candidate-generation followed by support evaluation. The PrefixSpan [2] algorithm follows a pattern growth method. It uses projected databases for achieving this. Prefix is Projected Sequential Pattern mining. It examines prefix subsequences and projects the postfix subsequences into the databases.

2 Model & Notation:

Based on the Agrawal and Srikant[1], a *sequence* is an ordered set of events, denoted by $(s_1, s_2, s_3, \ldots, s_n)$ where s_j is an event. Each *event*, also referred to as an itemset, is a non-empty, unordered, finite set of items, denoted by $(i_1, i_2, i_3, \ldots, i_n)$ where i_j is an item. The *length* of a sequence is the number of items present in the sequence. A sequence of length k is called a *k-sequence*. A sequence X denoted by $(x_1, x_2, x_3, \ldots, x_n)$ is said to be a *subsequence* of another sequence Y denoted by $(y_1, y_2, y_3, \ldots, y_n)$ if $x_1 \subseteq y_{i_1}, x_2 \subseteq y_{i_2}, \ldots, x_n \subseteq y_{i_n}$ where $i_1, i_2, i_3, \ldots, i_n$ are integers such that $i_1 < i_2 < i_3 < \ldots < i_n$. The sequence Y is said to contain the sequence X if X is a subsequence of Y. The sequence database S is a set of the form (cid, s) where cid, is the customer-id and s is the sequence. Let min-support be the user defined support, a sequence that occur more frequently than min-support is called *frequent sequence*. A *maximal sequence* is the one that is not a subsequence of any other frequent sequence.

We define dominancy of an event e consisting of items $(i_1, i_2, \ldots, i_k)$ as ED =Max $(\sup(i_1), \sup(i_2), .. \sup(i_k))$, where sup(i) is the frequency of occurece of item i. And, the dominancy factor of a sequence s_i, denoted by DF_i, as the sum dominancy of individual events that appear in the sequence. For a sequence $s = \{x_\ell \mid \ell = 1, 2 \ldots N_i ; x_\ell \in s\}$, $DF_i = \sum_{\ell=1}^{Ni} ED(x_\ell)$, $\forall x_\ell \in T_i$.

The algorithm proposed in this work utilizes the dominancy factors for pruning the search. Thus, during the preprocessing, we sort the entire dataset based on descending order of DF as shown in the table 1. By performing such transformations to the

initial dataset, we bring the dominant transactions i.e. those transactions that have longer potentially frequent patterns, to the top in order to mine the long patterns efficiently.

Table -1: Sequence Data set

CId	Sequence	Dominancy
C1	cg(af)cbc	4+1+Max(4,2)+4+4+4=21
C2	(cf)(ab)(df)cbe	20
C3	a(abc)(ac)d(cf)	19
C4	(ad)c(bc)(ac)	16

Sup(a) = 4, sup(b)= 4, sup(c)=4 sup(d)=3
sup(e)=1 sup(f)=2 sup(g)=1

In a dataset sorted based on descending order of DF, for a sequence X to be present in a sequence C_i, it should satisfy the property DF (X) ≤ DF_i. So, as we are walking down the data set, we can determine a point beyond which a sequence, X, would not exists. For example in table 1, for the sequence = (c->a->b->a->c) the dominancy factor, computes to the value of 20. Hence, in this case, we need to scan till C2 only as the DF_2 = 20. So, we define a point beyond which a sequence X of size k would not exist as the Maximum Depth of Traversal (MDT_k) for a k-sequence, X_k. A point beyond which any sequence of size k will not exist is defined as $MaxMDT_k$.

In other words, the dataset or transaction list, D, is an ordered set of quadruplet Q,

D = {<Qi> | DFi ≥ DFi+1 } where <Qi> = {(S_i, N_i, DF_i, Sup_i) | i = 1, 2,….L; S_i is the ith customer sequence; N_i is the number of events in S_i; DF_i = $\sum_{\ell=1}^{Ni} ED(x_\ell)$; Sup_i = Sup (S_i);} Where L is the total number of sequences in the dataset.

The preprocessed dataset with the sorted transaction list has several properties that can be used for pruning the search space. These properties are as follows.

Property 1:

A candidate k-sequence, $X_k \not\subset S_i \quad \forall S_i > MDT_k$. In other words, a candidate sub sequence X_k is definitely not a subset of sequence S_i beyond the Maximum Depth of Traversal, MDT_k of the candidate, X_k.

Since, the customer sequences in the dataset, D are sorted on descending order of DF_i, we can identify a point at which DF_i < DF (X_k). Further, the dominancy factor is just the sum of supports of individual items in a transaction list (DF_i). So, for a sequence to be present in a transaction, its total sum of supports or the dominancy factor should be more than the sum of supports of the subset to be evaluated DF (X_k).

Property 2:

For an sequence X_k to be frequent, the upper bound of the support should be at least equal to the user defined minimum support, minsupport. This property is used to prune infrequent sequences even before adding the sequence to the sequence tree. The duplicate transactions are combined together in the preprocessing step and the support (sup_i) is incremented for the i^{th} sequence appropriately. Thus, the support upper bound on support can be computed as follows.

$$\text{Upper bound } (X_k) = \sum_{m=fk}^{MDTk} (Sup_m)$$

If $X_k \in \Im$, then minsupport ≤ upper bound (X_k) Otherwise $X_k \notin \Im$.

As each and every item in a sequence is distinct, any candidate sequence can occur at-most as many times as the support of a particular sequence. As per the property 1, any sequence X_k cannot exist beyond its MDT_k. So, once we know MDT_k and the sequence's first occurrence in data set, D, i.e. S_{fk}, we can ascertain the upper bound of the support by summing up the support of all sequences between these two points

Property 3: Reverse Apriory Principle

All subsequence of a maximal sequence are frequent and hence need not be evaluated for generating maximal sets.

Property 4: Apriori Principle

All supersets of an infrequent sequence are infrequent and hence need not be evaluated.

3 VARIOUS PHASES OF OUR ALGORITHM

Our algorithm follows top-down approach for mining the maximal sets. The frequent sequences are divided in two categories- one with repeated items and other with non-repeated items. The different phases of the algorithm are as follows:

Pre-Processing Phase

During this phase, we prepare the data and organize it in the form of sequence list described in the earlier section. The original sequence database is scanned. During the scan we count the support for all the 1-

sequence and 2-seqences. The sequence list transformation requires that all the duplicate sequences are collapsed together and the support value reflects the number of identical sequences that have been collapsed. The final contains unique sequences with the frequency of each sequence in the original data set. We compute the dominancy factor of the sequences and add it to the Sequence database D. Finally, the transaction list is sorted on the descending order of dominancy factor.

Sequence Generation, Counting and Pruning

Sequence Graph Construction

During the preprocessing phase the frequent sequences of length 1 and 2 were identified. We construct a directed graph with the items being represented as nodes. The edge between any pair of nodes i and j has a weight w_{ij} where w_{ij} is equal to the frequency of the 2-sequence i->j provided that the sequence i->j is frequent. We store the frequent sequences in a set FS. At end of this step, we have a graph which represents all the frequent 2-sequences. In the first pass of algorithm we try to identify and evaluate potential *long and rich candidates*. The rich sequences are the one whose constituent 2-sequences have high support. In the directed graph, the 2-sequenece frequencies are represented by the edge weights; we can easily compute the path with the highest weights between all pairs of nodes. We can use modified Floyd-Warshal algorithm for the purpose finding longest path as the rich candidates. For the data set shown in Table -1, the supports of all the 2-sequences are

a->b : 1, a->c: 4, a->d: 2;
a->f: 1; b->a: 2; b->c: 3;
b->d: 1, b->f: 1; c->a: 3;
c->b: 3; c->d: 1; c->f: 0;
d->a: 0, d->b: 0; d->c:3;
d->f: 1; f->a: 1, f->b:0,
f->c:2, f->d: 0;

The directed graph representing frequent 2-sequences is shown in Figure -1:

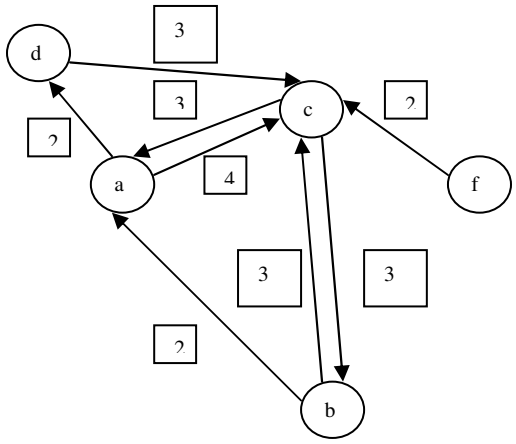

Figure-1: Sequence Graph for the data set

The solution for the above graph (entries show the weight and previous node in the path) is as follows.

Table-2: Longest Paths

	a	B	C	D	f
A	NA	8/c	5/d	2/d	-
B	6/c	NA	7/d	9/a	-
C	5/b	3/c	NA	7/a	-
D	8/b	6/c	3/d	NA	-
F	7/b	5/c	2/f	9/a	NA

The algorithm generates the following path k-sequences (k>2): f->c->b->a->d, a->d->c->b, b->a->d->c, b->c->a->d, c->b->a->d, d->c->b->a, f->c->b->a, a->d->c, b->c->a, c->b->a, d->c->b, f->c->b. These long candidate sequences are evaluated by scanning the sequence data for support frequencies using the property 1-4. As a result the infrequent sets found are- f->c->b->a->d, a->d->c->b, b->a->d->c, b->c->a->d, c->b->a->d, d->c->b->a, f->c->b->a, b->c->a , c->b->a, and the frequent sequences are a->d->c, d->c->b, f->c->b.

Sequence Generation and Evaluation

Candidate Generation **-** As our algorithm follows top down approach, we start with generating all subsequences of size, k equal to the maximum length of sequence in dataset, Maxlen, from the sequence graph.

Pruning -We check the already existing infrequent sequence list to determine that no subsequence of the candidate sequence is in the infrequent list. If any subsequence is in the infrequent list the candidate sequence is also added to the infrequent list (property 3). We also check to see if the candidate sequence is subsequence of any sequence contained in the

frequent list (property 4). If so, the candidate sequence is added to the frequent list. Otherwise the sequence is added to the candidate list. Also, we apply the support upper bound (property 2) to check whether the subset could be infrequent, If the generated sequence is infrequent, the sequence is pruned. Otherwise, we add the sequence to the candidate sequence.

Support Counting **-** We compute $MaxMDT_k$ of all the candidate sequences and evaluate the frequency of occurrence by scanning the customer sequence list to the point $MaxMDT_k$. The membership of sequence X_k can be decided once the customer sequence corresponding to its MDT_k has been processed, as no transactions beyond this point are going to contribute towards the support of X_k. The process of 3.2.2 is repeated by generating all paths of length k -1 from the sequence graph, till we have evaluated the paths of size 3 and above or in the previous step all the paths generated were subsumed in the frequent list.

In the example table the maxlen is 6, in the first pass, algorithm finds that sequence graph does not contain any path of length 6, in subsequent pass a sequence (f-c-b-a-d) of length 5 generated but is pruned as it is already part of the infrequent list. In the next pass seven sequences of length 4 are generated, 6 of them are already in the infrequent list and are pruned. The candidate sequence f-c-a-d is pruned based on support upper bound. Finally, the sequences of path length 3 are generated from the graph and final scan of the data set is made to check the occurrences, the additional 3-sequences c->a->d is found to be frequent as well.

Sequence with the repeated items

The data set is transformed to a vertical set. In the vertical set, each entry in the table has information about the item's customer id (cid) and the position of event (eid). This enables us to compute the distance between the successive occurrences of the same item in a sequence. We compute the distance and the frequency of such occurrences. a->d->c, d->c->b, f->c->b. c->a->d, a->c, a->d, b->a, b->c, c->a, c->b, d->c, f->c

The vertical layout for the database with the (cid,eid) pair showing the occurrence of an item in the customer id and event id. For example entry (1,2) in the list of 'A' means that 'A' has occurred in the customer 1's 2nd event. The vertical dataset for the example is as follows:
Items: (cid,eid) pair for the items

A: (1,1),(1,2), (1,3), (2,1), (2,4), (3,2), (4,3)
B: (1,2),(2,3),(3,2), (3,5), (4,5)
C: (1,2),(1,3),(1,5),(2,2),(2,3),(2,4),(3,1),(3,4), (4,1),(4,4),(4,6)
D: (1,4),(2,1),(3,3)
F: (1,5),(3,1),(3,3),(4,3)

From the above vertical layout, the repetition distance and frequency table can be derived and is shown in Table-3.

Table-3: Repetition Distance and Frequency

Nodes	Distance, frequency			
a and a	1,2	3,1		
b and b	3,1	-		
c and c	1,3	2,3	3,3	5,1
d and d	-	-		
f and f	2,1	-		

Since, the min support is defined to be 2, from the table it can be seen tht only repetition of a at a distance of 1 and c at a distance of 1,2 and 3 in the sequence has minimum support. We take all the frequent sequences found in the earlier phase, and generated repetition patterns. So the candidates generated by adding 'a' are illustrated here.
a->ac, a->ad, b->a->a, c->a->a, c->a->ad and
a->ad->c

All the candidates sequences generated after adding a's and c's are checked in the data a set. Out of these, a->ac, a->c->c and c->a->d->c are found to be frequent.

4 Conclusion

The algorithm uses top down approach and generates the longest possible candidate sequences. It also uses the dominancy of sequences to determine the Maximum depth of traversal to compute the support for a candidate sequence. For the long patterns usually these numbers are likely to contain a small percentage of data set. This ensures that during the evaluation the whole data set is not scanned. Also, if the data set contains long frequent sequences, it is identified early enough and thus all the subsequences of this are also marked frequent and need not be evaluated. The bottom-up algorithms start from 2-sequences and keep building it up to the longest possible sequence. In case of really long sequences these algorithm scan the data set as many times as is the length of longest times, in contrast this algorithm will identify such patterns early enough and use it for pruning all its subsequences.

References

1. Agrawal, R., and Srikant, R., Mining Sequential Patterns: Generalizations and Performance Improvements Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology p. 3 – 17, (1996).
2. Pei J, Han J. et al: "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth" in Int'l Conf Data Engineering,p 215-226 (2001)
3. Zaki, M. J., SPADE: An Efficient Algorithm for Mining Frequent Sequences, Machine Learning, v.42 n.1-2, p.31-60, January-February 2001.
4. Garofalakis M, Rastogi R and Shim, K, "Mining Sequential Patterns with Regular Expression Constraints", in IEEE Transactions on Knowledge and Data Engineering, vol. 14, nr. 3, pp. 530-552, (2002).
5. Antunes, C and Oliveira, A.L: "Generalization of Pattern-Growth Methods for Sequential Pattern Mining with Gap Constraints" in Int'l Conf Machine Learning and Data Mining, (2003) 239-251
6. Lin, D I, and Kedem, Z M, "Pincer Search: A new algorithm for discovering the maximum frequent set", in International conference on Extending database technology, 1998.