A Novel Approach for Scalability – Two Way Sequential Pattern Mining using UDDAG

Dr.P.Raguraman
Assistant Professor
Department of Computer Science
Sri Sankara Arts and Science College
Enathur, Kanchipuram, TN, India

Mr.S.Hariharan
Assistant Professor
Department of Computer Science
Sri Sankara Arts and Science Collegee
Enathur, Kanchipuram, TN, India

Mr.V.Raji
Assistant Professor
Department of Computer Science
Sri Sankara Arts and Science Collegee
Enathur, Kanchipuram, TN, India

Abstract—Traditional pattern growth-based approaches for sequential pattern mining derive length- (k + 1) patterns based on the projected databases of length-k patterns recursively. At each level of recursion, they unidirectionally grow the length of detected patterns by one along the suffix of detected patterns, which needs k levels of recursion to find a length-k pattern. In this paper, a novel data structure, UpDown Directed Acyclic Graph (UDDAG), is invented for efficient sequential pattern mining. UDDAG allows
bidirectional pattern growth along both ends of detected patterns. Thus, a length-k pattern can be detected in | $\log_2$ k + 1| levels of recursion at best, which results in fewer levels of recursion and faster pattern growth. When minSup is large such that the average pattern length is close to 1, UDDAG and PrefixSpan have similar performance because the problem degrades into frequent item counting problem. However, UDDAG scales up much better. It often outperforms PrefixSpan by almost one order of magnitude in scalability tests. UDDAG is also considerably faster than Spade and LapinSpam. Except for extreme cases, UDDAG uses comparable memory to that of PrefixSpan and less memory than Spade and LapinSpam. Additionally, the special feature of UDDAG enables its extension toward applications involving searching in large spaces.

Index Terms—Data mining algorithm, directed acyclic graph, performance analysis, sequential pattern, transaction database.

1 INTRODUCTION

SEQUENTIAL pattern mining is an important data mining problem, which detects frequent subsequences in a sequence database. A major technique for sequential pattern mining is pattern growth. Traditional pattern growth-based approaches(e.g., PrefixSpan) derive length-(k + 1) patterns based on the projected databases of a length-k pattern recursively. At each level of recursion, the length of detected patterns is grown by 1, and patterns are grown unidirectionally along the suffix direction. Consequently, we need k levels of recursion to mine a length-k pattern, which is expensive due to the large number of recursive database projections. In this paper, a new approach based on UpDown Directed Acyclic Graph (UDDAG) is proposed for fast pattern growth. UDDAG is a novel data structure, which supports bidirectional pattern growth from both ends of detected patterns. With UDDAG, at level i recursion, wemay grow the length of patterns by 2i_1 at most. Thus, a length-k pattern can be detected in | $\log_2$ k + 1| levels of recursion at minimum, which results in better scale-up property for UDDAG compared to PrefixSpan. Our extensive experiments clearly demonstrated the strength of UDDAG with its bidirectional pattern growth strategy. When minSup is very large such that the average length of patterns is very small (close to 1), UDDAG and PrefixSpan have similar performance because in this case,
the problem degrades into a basic frequent item counting problem. However, UDDAG

IJERT

scales up much better compared to PrefixSpan. It often outperforms PrefixSpan by one order of magnitude in our scalability tests. UDDAG is also considerably faster than two other representative algorithms, Spade and LapinSpam. Except for some extreme cases, the memory usage of UDDAG is comparable to that of PrefixSpan. UDDAG generally uses less memory than
Spade and LapinSpam. UDDAG may be extended to other areas where efficient
searching in large searching spaces is necessary.

Related Work

The problem of sequential pattern mining was introduced by Agrawal and Srikant [1]. Among the many algorithms proposed to solve the problem, GSP [17] and PrefixSpan[13], [14] represent two major types of approaches: a prioribased and pattern growth-based.A priori principle states that any supersequence of a nonfrequent sequence must not be frequent. A priori-based approaches can be considered as breadth-first traversal algorithms because they construct all length-k patterns before constructing length-(k+1) patterns. The AprioriAll algorithm [1] is one of the earliest a prioribased approaches. It first finds all frequent item sets, transforms the database so that each transaction is replaced by all frequent item sets it contains, and then finds patterns. The GSP algorithm [16] is an improvement over AprioriAll.To reduce candidates, GSP only creates a new length-k candidate when there are two frequent length-(k _ 1) sequences with the prefix of one equal to the suffix of the other. To test whether a candidate is a frequent length-k
pattern, the support of each length-k candidate is counted by examining all the sequences. The PSP algorithm [12] is
similar to GSP except that the placement of candidates is improved through a prefix tree arrangement to speed up pattern discovery. The SPIRIT algorithm [9] uses regular expressions as constraints and developed a family of algorithms for pattern mining under constraints based on a priori rule. The SPaRSe algorithm [3] improves GSP by using both candidate generation and projected databases to achieve higher efficiency for high pattern density conditions.

Problem Definition

UPDOWN DIRECTED ACYCLIC GRAPH-BASED SEQUENTIAL PATTERN MINING

UDDAG-based pattern mining approach, which first transforms a database based on
frequent item sets, then partitions the problem, and finally, detects each subset using UDDAG. The absolute support for an item set in a sequence database is the number of tuples whose sequences contain the item set. An item set with a support
larger than minSup is called a frequent item (FI) set. Based on frequent item sets, we transform each sequence in a database D into an alternative representation.

Transformed database

Definition: Let D be a database and P be the complete set of sequential patterns in D, D’ be its transformed database, substituting the ids of each item pattern contained in D’ with the corresponding item sets, and denoting the resulted pattern set by P’, we have P = P’.

Based on frequent item sets, we transform each sequence in a database D into an alternative representation. Steps involved in Database Transformation:
1)Find the set of frequent items in D
2) Assign a unique id to each FI in D and then replace each item set in each

IJERT

sequence with the ids of all the FIs contained in the item set.

Seq. Id	Sequence
1	<1 (1,2,3) (1,3) 4 (3,6)>
2	<(1,4) 3 (2,3) (1,5)>
3	<(5,6) (1,2) (4,6) 3 2>
4	<5 7 (1,6) 3 2 3>

Table 1: sequence database

Seq. Id	Sequence
1	<1 (1,2,3,4,5) (1,5) 6 (5,8)>
2	<(1,6) 5 (3,4,5) (1,7)>
3	<(7,8) (1,2,3) (6,8) 5 3>
4	<7 (1,8) 5 3 5>

Table 2: Transformed Database

For the database in Table 1, **the FIs are:**
(1),(2), (3), (4), (5), (6), (1,2), (2,3).
By assigning a unique id to each FI, e.g., (1)-1, (1,2)-2, (2)-3, (2,3)-4, (3)-5, (4)-6, (5)-7, (6)-8, we can transform the database as shown in Table 2 (infrequent items are eliminated).

IJERT

Problem Partitioning
Definition: Let {x1, x2, . . . , xt} be the frequent item sets in a database D, x1 < x2 <. . . . < xt, the complete set of patterns (P) in D can be divided into t disjoint subsets. The ith subset (denoted by $P_{xi,}$ 1 < = i <= t) is the set of patterns that contains xi and FIs smaller than xi.

Definition (Projected database):
The collection of all the tuples whose sequences contain an item set x in a database D is called x-projected database, denoted by ^{x}D.

The total number of different Frequent Items FIs in the Sequential Database are found by Database Transformation module. For a database with n different frequent items, its patterns can be divided into n disjoint subsets. The ith subset (1 < i < n) is the set of patterns that contain i (the root item of the subset) and items smaller than i.Each subset i of the problem is mapped into a projected database denoted by (^{i} **D)**

P is partitioned into eight subsets as there are 8 FIs in table-2, the one contains 1 (P1), the one contains 2 and smaller ids (P2), . . . ,and the one contains 8 and smaller ids (P8).

Given the following database (P8) alone is found as given by ^{8}D:
1. <9 4 5 8 3 6>
2. <3 9 4 5 8 3 1 5>
3. <3 8 2 4 6 3 9>
4. <2 8 4 3 6>
5. <9 6 3>

^{8}D is,
1. <4 5 8 3 6>
2. <3 4 5 8 3 1 5>
3. <3 8 2 4 6 3>
4. <2 8 4 3 6>

UpDown Directed Acyclic Graph

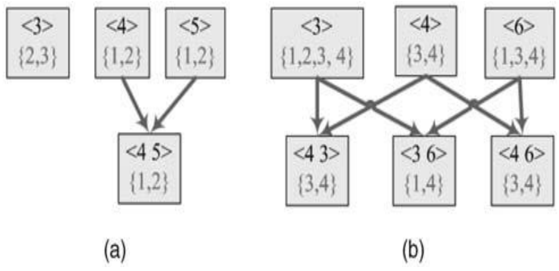
Definition: Directed acyclic graph (UDDAG) is a graphical approach that represents patterns as vertices and contain relationships as directed edges in between vertices.

Given an FI x and ^{x}D, an UpDown Directed Acyclic Graph based on P_x, denoted by x-UDDAG, is constructed as follows:

1. Each pattern in P_x corresponds to a vertex in x-UDDAG. <x> corresponds to the root vertex, denoted by V_x
2. Let P_U be the set of length-2 patterns ending with x in P_x . Add a directed edge from V_x to U_v called up root child of $V_{x.}$ This represents the common prefix set to root X.
3. Let P_D be the set of length-2 patterns starting with x in P_x, add a directed edge from V_x to V_D called a down root child of V_x. This represents the common suffix set to root X.

Such a DAG can be recursively constructed in an efficient way to derive the contain relationship of patterns

Example of UDDAG Construction:



By concatenating the patterns {<3>,<4>,<5>,<4 5>} of Pre(^{8}D) with 8, we derive patterns {<3 8>, <4 8>, <5 8>, <4 5 8>} in figure-a.
By concatenating patterns {<3>, <4>, <6>, <3 6>, <4 3>, <4 6>} of Suf(^{8}D) with 8,we derive patterns {<8 3>, <8 4>, <8 6>, <8 3 6>, <8 4 3>, <8 4 6>} in figure-b.

UDDAG based Pattern Mining:
In the ith subset, each pattern in the projected database (^{x}D) can be divided into two parts, prefix and suffix of i.

The collection of all the prefix/suffix tuples of a frequent item set X in ^{x}D is called the prefix/suffix-projected database of x, denoted by Pre(^{x}D) / Suf(^{x}D).

To detect the sequential pattern in projected database (^{x}D) P_x,

- Detect patterns in Pre(^{x}D) called pattern prefix (PP)
- Detect pattern in Suf(^{x}D) called pattern suffix (PS)
- The above steps are repeated recursively until no frequent items are found in the pre(^{x} D) / suf(^{x} D) .
- Combine the patterns of all the iterations to derive P_x.

The complete set of patterns is the union of patterns of the all subsets or projected database (^{x}D) detected above.
The Apriori property is used to reduce the number of candidate sets to be considered

Example for Pattern Mining:
Assuming ^{8}D is,
1. <4 5 8 3 6>
2. <3 4 5 8 3 1 5>
3. <3 8 2 4 6 3>
4. <2 8 4 3 6>.
The prefix subsequences of 8 in ^{8}D, or Pre(^{8}D) is :**{<3 8>, <4 8>, <5 8>, <4 5 8>}** the patterns with 8 at the end.

The suffix subsequences of 8 in ^{8}D, or Suf(^{8}D) is : **{<8 3>, <8 4>,<8 6>, <8 3 6>, <8 4 3>, <8 4 6>}** the patterns with 8 at the beginning.

The patterns with 8 in between the beginning and end of each pattern is:
{<3 8 3>,<4 8 3>, <5 8 3>, <4 5 8 3>}

Example: UDDAG based Pattern Mining:

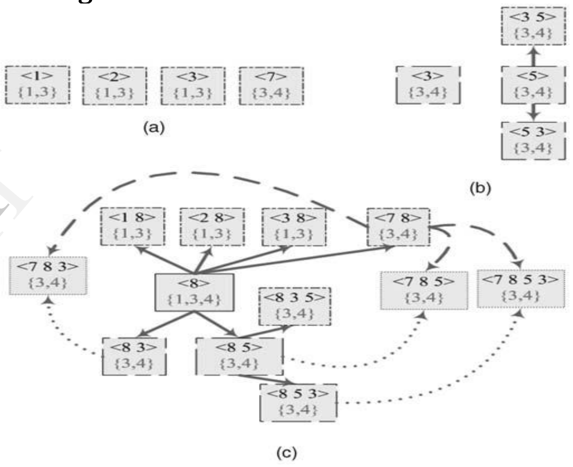


Fig a: UP DAG
Fig b: DOWN DAG
Fig c: UPDOWN DAG

Algorithm 1. UDDAG based pattern Mining.
Input: A database D and the minimum support
Output: P, the complete set of patterns in D
Method: findP (D, minSup){

```
P = Φ
FISet=D:getAllFI(minSup);
D.transform( );
for each FI x in FISet{
UDVertexrootVT = newUDVertex(x)
findP(D.getPreD(x), rootVT, up, minSup)
findP(D.getSurD(x),rootVT,down,minSup)
findPUDDAG(rootVT)
P = P U rootVT.getAllPatterns( )
```

IJERT

```
}
}

Subroutine:findP(PD,rootVT,type, minSup){
FISet=PD.getAllFI(minSup);
for each FI x in FISet{
UDVertexcurVT=new DVertex(x, rootVT)
if(type==up)
rooVT.addUpChild(curVT)
else rootVT.addDownChild(curVT)
findP(PD. getPreD(x), curVT, up, minSup)
findP(PD.getSufD(x),curVT,down,minSup)
findPUDDAG(curVT)
}
}

Subroutine: findPUDDAG(rootVT){
upQueue.enQueue (rootVT.upChildren)
while(!upQueue.isEmpty()){
UDVertex upVT=upQueue.deQueue()
if(upVT.upParent==rootVT)
downQueue.enQueue(rootVT.downChildren)
else if (upVT.downParent==null)
downQueue.enQueue(upVT.upParent.VDVS)
else
downQueue.enQueue(upVT.upParent.VDVS ∩ upVT.downParent.VDVS)
while(!downQueue.isEmpty()){
UDVertexdownVT=downQueue.deQueue()
if(isValid(upVT, downVT){
UDVertexcurVT=new UDVertex (upVT, downVT)
upVT.addVDVS(downVT)
if(upVT.upParent==rootVT)
downQueue.enQueue(downVT.children)
}
}
if(upVT.VDVS.size>0)upQueue.enQueue(upVT.children)
}
}
```

IJERT

Performance Evaluation

We conducted an extensive set of experiments to compare our approach with other representative algorithms. All the experiments were performed on a windows Server 2003 with 3.0 GHz Quad Core Intel Xeon Server and 16 GB memory. The algorithms we compared are PrefixSpan,
Spade, and LapinSpam, which were all implemented in C++ by their authors (Minor changes have been made to adapt Spade to Windows). Two versions of UDDAG were tested. UDDAG-bv uses bit vector to verify candidates and UDDAG-co uses co-occurrences to verify candidates whenever possible. We perform two studies using the same data generator as in [14]: 1) Comparative study, which uses similar data sets as that in [14]; 2) Scalability study. The data sets were generated by maintaining all except one of the parameters as shown in Table 4 fixed, and exploring different values for the remaining ones.

CONCLUSION

In this paper, a novel data structure UDDAG is invented for efficient pattern mining. The new approach grows patterns from both ends (prefixes and suffixes) of detected patterns, which results in faster pattern growth because of less levels of database projection compared to traditional approaches. Extensive experiments on both comparative and scalability studies have been performed to evaluate the proposed algorithm.
One major feature of UDDAG is that it supports efficient pruning of invalid candidates. This represents a promising approach for applications involving searching in large spaces. Thus, it has great potential to related areas of data mining and artificial intelligence. In the future, we expect to further improve UDDAG-based pattern mining algorithm as follows: 1) Currently, FI detection is independent from

pattern mining. Practically, the knowledge gained from FI detection may be useful for pattern mining. In the future, we will integrate the solutions of the two so that they can benefit from each other. 2) Different candidate verification strategies may have different impacts to the efficiency of the algorithm. In the future, we will study more efficient verification strategy. 3) UDDAG has big impact to the memory usage when the number of patterns in a subset is extremely large. In the future, we will find an efficient way to store UDDAG.

REFERENCES

[1] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. Int'l Conf. Data Eng. (ICDE '95), pp. 3-14, 1995. CHEN: AN UPDOWN DIRECTED ACYCLIC GRAPH APPROACH FOR EQUENTIAL PATTERN MINING 927

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Int'l Conf. Very Large Data Bases (VLDB), pp. 487-499, 1994.

[3] C. Antunes and A.L. Oliveira, "Generalization of Pattern-Growth Methods for Sequential Pattern Mining with Gap Constraints," Proc. Int'l Conf. Machine Learning and Data Mining 2003, pp. 239-
251, 2003.

[4] J. Ayres, J. Gehrke, T. Yu, and J. Flannick, "Sequential Pattern Mining Using a Bitmap Representation," Proc. Int'l Conf. Knowledge Discovery and Data Mining 2002, pp. 429-435, 2002.

[5] S. Berkovich, G. Lapir, and M. Mack, "A Bit-Counting Algorithm Using the Frequency Division Principle," Software: Practice and Experience, vol. 30, no. 14, pp. 1531-1540, 2000.

[6] J. Chen and T. Cook, "Mining Contiguous Sequential Patterns from Web Logs," Proc. World Wide Web Conf. (WWW '07) Poster Session, May 2007.

[7] J. Chen and K. Xiao, "BISC: A Binary Itemset Support Counting Approach Towards Efficient Frequent Itemset Mining," to be published in ACM Trans. Knowledge Discovery in Data.

[8] G. Grahne and J. Zhu, "Efficiently Using Prefix-Trees in Mining Frequent Itemsets," Proc. Workshop Frequent Itemset Mining Implementations (FIMI '03), 2003.

[9] M. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints," Proc. Int'l Conf. Very Large Data Bases (VLDB '99), pp. 223-234, 1999.

[10] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern

IJERT