ISSN : 0976-8491(Online) | ISSN : 2229-4333(Print)

An Implementation of UpDown Directed Acyclic Graph Approach

[1]Durgarani Mahankali, [2]Mrs. A.M. Sowjanya, [3]Mrs.M.Shashi
[1,2,3]Dept. of CS & SE Andhra University, Visakhapatnam, AP, India

Abstract
UpDown Direcrted Acyclic Graph Approach (UDDAG) is the one of the fastest Approaches for Sequential pattern Mining. In this paper we implemented the UDDAG Approach using Java .For identifying the frequent itemsets (FIs) we use FP-Growth Algorithm which is one of the fastest algorithm to identify frequent items. This implementation can be applicable to Market data applications.

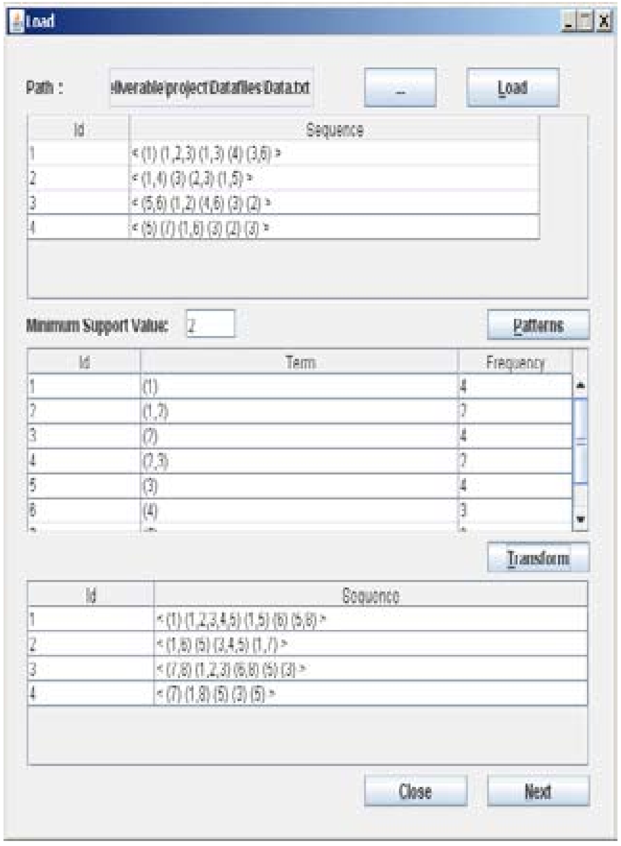
Keywords
Directed acyclic graph, sequential pattern

I. 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. In this paper, a new approach based on UpDown Directed Acyclic Graph Approach is implemented. UDDAG is a novel data structure which supports bidirectional pattern growth from both ends of detected patterns. UDDAG may be extended to other areas where efficient searching in large searching spaces is necessary. UDDAG implementation involves 3steps i.e., database transformation, pattern partitioning, UDDAG-based pattern mining. A sequence database is a set of tuples <sid, s>, where sid is a sequence id and s is a sequence. A tuple <sid, s> is said to contain a sequence β if $\beta \prec s$. The absolute support of a sequence β in a sequence database D is defined as $Sup_D = |\{<sid,s> | (\beta \prec s) \wedge (<sid,s> \in D)\}|$, and the relative support of β is defined as $Sup_D(s) / |D|$. Given a positive value minSup as the support threshold, β is called a sequential pattern in D if $Sup_D(s) \geq minSup$. In this paper we describe efficient implementation of UDDAG approach. In section II we explain the how sequence database is transformed. In section III we explain identification of patterns for the transformed database. In section IV involves the finding subsets of patterns and final sequential patterns for the given database.

II. Database Transformation

Based on frequent item sets, we transform each sequence in a database D into an alternative representation. First, we assign a unique id to each FI in D. We then replace each item set in each sequence with the ids of all the FIs contained in the item set. To detect all FIs we use FP-Growth Algorithm.
For example, 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 fig 1 (infrequent items are eliminated).



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) >

Id	Term	Frequency
1	(1)	4
2	(1,2)	2
3	(2)	4
4	(2,3)	2
5	(3)	4
6	(4)	3

Id	Sequence
1	< (1) (1,2,3,4,5) (1,5) (8) (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) >

Fig.1: Database Transformation

III. Pattern Partitioning

Let $\{x_1, x_2, \ldots, x_t\}$ be the frequent item sets in a database D; $x_1 < x_2 < \ldots < x_t$, the complete set of patterns (P) in D can be divided into t disjoint subsets. The ith subset (denoted by P_{xi} ; $1 \leq i \leq t$} is the set of patterns that contains x_i and FIs smaller than x_i. To support bidirectional pattern growth, instead of partitioning patterns based on common prefix, we can partition them based on common root items. For a database with n different frequent items (without loss of generality, we assume that these items are 1; 2; . . . ; n), its patterns can be divided into n disjoint subsets. The ith subset ($1 \leq i \leq n$) is the set of patterns that contains i (the root item of the subset) and items smaller than i. Since any pattern in subset i contains i, to detect the i th subset, we need only check the subset of tuples whose sequences contain i in database D, i.e., the projected database of i, or ${}^iD$. In the ith subset, each pattern can be divided into two parts, prefix and suffix of i. Since all items in the ith subset are no larger than i, we exclude items that are larger than i in ${}^iD$.
Example 2: Given the following database:
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>,

8D 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>.

If minSup is 2, the 8th subset of patterns is
{<8>;<3 8>; <4 8>; <5 8>; <4 5 8>; <8 3>;
<8 4>; <8 6>;<8 3 6>; <8 4 3>; <8 4 6>;
<3 8 3>; <5 8 3>; <4 5 8 3>}
Observing the patterns in the 8th subset, except for <8>, which only contains 8 and can be derived directly, all other patterns can be clustered and derived as follows: 1. f<3 8>; <4 8>; <5 8>; <4 5 8>g, the patterns with 8 at the end. This cluster can be derived based on the prefix subsequences of 8 in 8D, or Pre(^{8}D), which is,
1. <4 5>;
2. <3 4 5>;
3. <3>; and
4. <2>.
By concatenating the patterns (<3>;<4>;<5>; <4 5>) of Pre(^{8}D) with 8, we can derive patterns in this cluster.

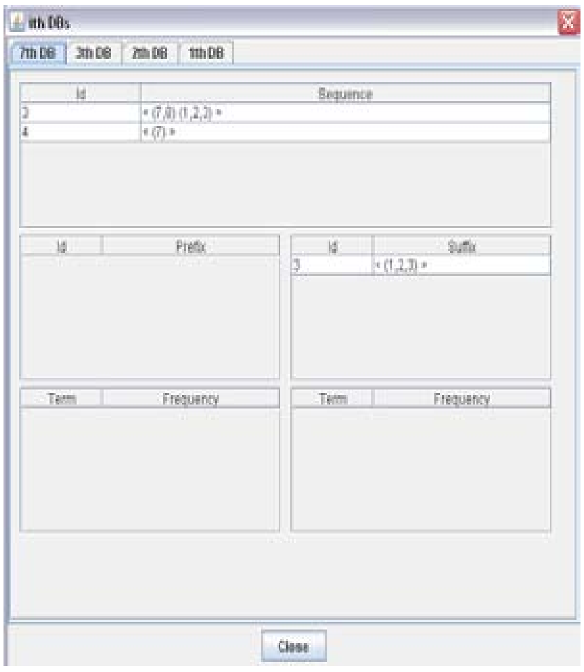
2. {<8 3>; <8 4>; <8 6>; <8 3 6>; <8 4 3>; <8 4 6>}, the patterns with 8 at the beginning. This cluster can be derived based on the suffix subsequences of 8 in ^{8}D, or Suf(^{8}D), which is
1. <3 6>;
2. <3 1 5>;
3. <2 4 6 3>;
4. <4 3 6>.

By concatenating 8 with the patterns (<3>; <4>; <6>; <3 6>; <4 3>; <4 6>) of Suf(^{8}D), we can derive patterns in this cluster.

3. {<3 8 3>; <4 8 3>; <5 8 3>; <4 5 8 3>}, the patterns with 8 in between the beginning and end of each pattern. This cluster can be mined based on the patterns in Pre(^{8}D) and Suf(^{8}D). In this case, a pattern (e.g., <4 8 3>) can be derived by concatenating a pattern of Pre(^{8}D) (e.g., 4) with the root item 8 and a pattern of Suf(^{8}D) (e.g., 3).

IV. Finding subsets of Patterns

To detect Px, we first detect patterns in Pre(^{x}D) and Suf(^{x}D) and then combine them to derive Px. This is a recursive process because for Pre(^{x}D) and Suf(^{x}D), we perform the same action until reaching the base case, where the projected database has no frequentitem set.
A. Finding P8. First, we find Pre(^{8}D) and Suf(^{8}D), which are,
Pre(^{8}D): 1) <1 (1,2,3,4,5) (1,5) 6>, 3) <(7,8) (1,2,3) >, 4) <7>;
Suf(^{8}D): 1) <>, 3) < (1,2,3) (6,8) 5 3>, 4) <5 3 5>.
Let PP be all the patterns in Pre(^{8}D), since the FIs in Pre(^{8}D) are (1), (2), (3), and (7), we can partition PP into four subsets, PP$_7$; PP$_3$; PP$_2$, and PP$_1$. First, we detect PP$_7$. Since the prefix-projected database of 7 in Pre(^{8}D) is empty, and the suffix-projected database of 7 is: 3) <(1, 2, 3)>, 4) < >, the only pattern in PP$_7$ is <7> as shown in fig. 2



Id	Sequence
3	< (7,8) (1,2,3) >
4	< (7) >

Id	Prefix

Id	Suffix
3	< (1,2,3) >

Term	Frequency

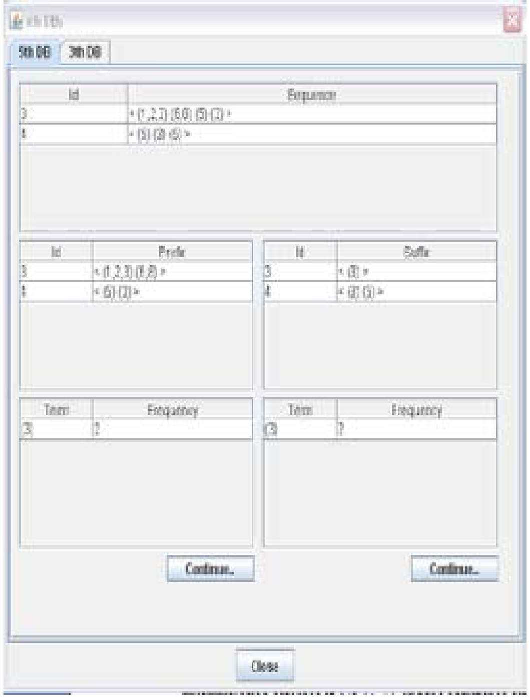
Term	Frequency

Fig. 2: PP$_7$ for Pre(^{8}D)

Similarly, PP $_3$ {<3>}; PP$_2$ {<2>}, and PP$_1$ { <1>}. Thus, PP= {<1>;<2>; <3>;<7>}.

Let PS be all the patterns in Suf(^{8}D), since the FIs in Suf(^{8}D) are (3) and (5), we can partition PS into two subsets, PS5 and PS3. First, we detect PS5. The prefix-projected database of 5 in Suf(^{8}D) is: 3) < (1,2,3) (6,8) >, 4) <5 3>, which contains a pattern <3>. The suffix-projected database of 5 in Suf(^{8}D) is: 3) < 3>, 4) <3 5>, which also contains a pattern <3>.

Suf(^{8}D) of the given database is shown in fig.3



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

Id	Prefix
3	< (1,2,3) (6,8) >
4	< (5) (3) >

Id	Suffix
3	< (3) >
4	< (3) (5) >

Term	Frequency
(3)	2

Term	Frequency
(3)	2

Fig. 3: Suf(^{8}D)

Since both databases have patterns, we need consider case 3, i.e., whether concatenating <3> with root 5 and <3> is also a pattern. Here, the occurrence set of <3> in the prefixprojected database of 5 is {3, 4}, and the occurrence set of <3> in

the suffix-projected database of 5 is also {3, 4}. Thus, their intersection set is {3, 4}, which means that the support of <3 5 3> is at most 2. However, since 5 occurs twice in tuple 4, we need check whether it really contains <3 5 3>, which is not true by verification. Thus, the support of <3 5 3> is 1, and it is not a pattern. Therefore, PS5 = { <3 5>;<5 3>;<5>}. S i m i l a r l y , PS3 ={<3>}. All together, PS ={<3 5>; <5 3>;<5>;<3>}.
Next, we detect P8 based on the Up and Down DAGs of 8 {(Fig. 4(a), 4(b)} by evaluating each candidate vertex pair. First, we detect the VDVSs for length-1 pattern in Pre(8D), i.e., up vertexes 1, 2, 3, and 7. For vertex 1, first, we check its combination with down vertex 3, the intersection of the occurrence sets is {3}. Thus, the corresponding support is at most 1, which is not a valid combination. Similarly, up vertex 1 and down vertex 5 are also invalid combination. All the children of down vertex 5 are not valid for up vertex 1. Therefore, VDVS1=Ø. Similarly, VDVS3 = Ø , VDVS7= {ov(<3>); ov(<5>); ov(<5 3>)}: Since no length-2 pattern exists in Pre(^{8}D), the detection stops. Eventually, we have

P8 ={<8>;<1 8>;<2 8>;<3 8>;<7 8>;
<8 3>;<8 5>;<8 3 5>;<8 5 3>;
<7 8 3>;<7 8 5>;<7 8 5 3>}
8-UDDAG based on detected patterns in P_8 is shown in fig. 4c.

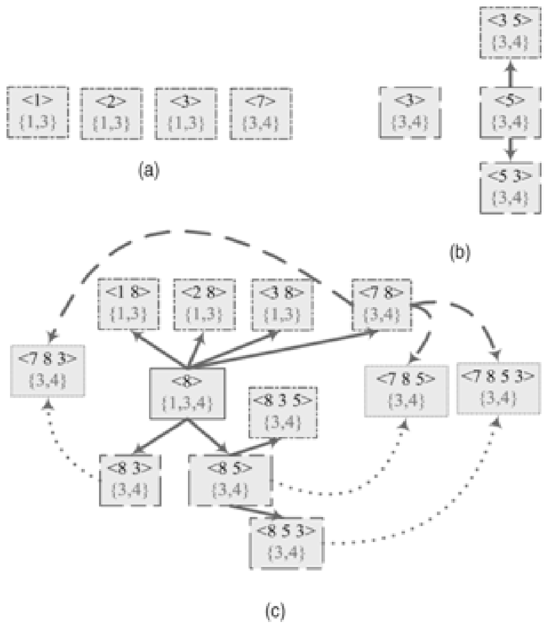


Fig. 4: UpDown DAG for P8. (a) Up DAG of patterns in pre(^{8}D). (b) Down
DAG patterns in Suf(^{8}D). (c) 8-UDDAG.

Similarly, we have

P7 ={<7>;<7 1>;<7 3>;<7 1 3>;<7 5>,
<7 1 5>;<7 3 5>;<7 5 3>};

P6 ={<6>;<1 6>;<2 6>;<3 6>;<6 3>;<6 5>;
<6 5 3>;<3 6 5>;<2 6 5>;<1 6 5>};

P5 = {<5>;<1 5>;<2 5>;<3 5>;<5 5>;
<1 3 5>;<1 5 5>; <5 1>;<5 3>;
<5 5>;<1 5 1>;<1 5 3>};

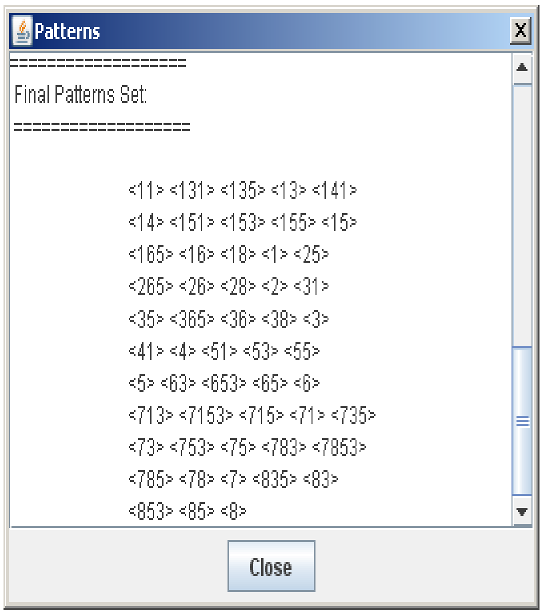
P4 = {<4>;<1 4>;<4 1>;<1 4 1>};

P3 ={<3>;<1 3>;<3 1>;<1 3 1>};

P2 = {<2>};

P1 = {<1>;<1 1>};

The complete set of patterns is union of all subsets of patterns detected above is shown in fig. 5



```
==================
Final Patterns Set:
==================

<11> <131> <135> <13> <141>
<14> <151> <153> <155> <15>
<165> <16> <18> <1> <25>
<265> <26> <28> <2> <31>
<35> <365> <36> <38> <3>
<41> <4> <51> <53> <55>
<5> <63> <653> <65> <6>
<713> <7153> <715> <71> <735>
<73> <753> <75> <783> <7853>
<785> <78> <7> <835> <83>
<853> <85> <8>
```

Fig. 5:

V. Conclusion

In this paper we describe the implementation of UpDown Directed Acyclic Graph Approach for a given sequence database. This implementation work for any type of market data .

References

[1] Jinlin Chen, "An UpDown Directed Acyclic Graph Approach for Sequential Pattern Mining", 2009, pp. 913-928
[2] R. Agrawal, R. Srikant, "Fast algorithms for mining association rules", Proc. of 20th Intl. Conf. on VLDB, pp. 487-499, 1994.
[3] C. Antunes, 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] M. Garofalakis, R. Rastogi, K. Shim, “SPIRIT: Sequential Pattern Mining with Regular Expression Constraints”, Proc. VLDB’99, pp. 223-234, 1999.
[5] H. Mannila, H Toivonen, A.I. Verkamo, “Discovery of Frequent Episodes in Event Sequences”, Data Mining and Knowledge Discovery, vol. 1, pp.259-289, 1997.
[6] F. Masseglia, F. Cathala, P. Poncelet, “The PSP Approach for Mining Sequential Patterns”, Proc. Eurpn. Symp. Principle of Data Mining and Knowledge Discovery, pp. 176-184, 1998.
[7] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, 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, 2001.

[8] Bahrami, "Object Oriented Systems Development", pp.325-338,2008.
[9] E. M. Reingold, J. Nievergelt, N. Deo, “Combinatorial Algorithms. Theory and Practice”, Prentice-Hall, Inc.: Englewood, Cliffs, NJ, 1977.
[10] R. Srikant, R. Agrawal, “Mining Sequential Patterns: Generalizations and Performance Improvements”, Proc. Int’l Conf. Extending Database Technology 1996, pp. 3-17, 1996.
[11] Z. Zhang, M. Kitsuregawa, “LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern”, Proc. of SWOD’05, pp. 8-11, Apr. 2005.

M.DURGARANI Received B.Tech Degree in Information Technology from Aharya Nagarjuna University in 2009 and M.Tech in Computer Science and Technology from Andhra University in 2011.

A.M. SOWJANYA completed B.Tech and M.Tech in computer Science and register for Ph.D in Andhra university and working as Assistant Professor in Andhra University College of Engineering.

M.SHASHI is a Professor in Andhra university College of Engineering and Presently working as Head of the Department in Andhra University College Of Engineering