1-2020

# Mining Integrated Sequential Patterns From Multiple Databases

C. I. Ezeife
*University of Windsor*

Vignesh Aravindan
*Royal Bank of Canada*

Ritu Chaturvedi
*University of Guelph*

# Table of Contents


## International Journal of Data Warehousing and Mining

Volume 16 • Issue 1 • January-March-2020 • ISSN: 1548-3924 • eISSN: 1548-3932


### Research Articles






The *International Journal of Data Warehousing and Mining* is indexed or listed in the following: ABI/Inform; ACM Digital Library; Australian Business Deans Council (ABDC); Bacon's Media Directory; Burrelle's Media Directory; Cabell's Directories; Compendex (Elsevier Engineering Index); CSA Illumina; Current Contents®/Engineering, Computing, & Technology; DBLP; DEST Register of Refereed Journals; Gale Directory of Publications & Broadcast Media; GetCited; Google Scholar; INSPEC; JournalTOCs; Library & Information Science Abstracts (LISA); MediaFinder; SCOPUS; The Index of Information Systems Journals; The Standard Periodical Directory; Ulrich's Periodicals Directory; Web of Science; Web of Science Science Citation Index Expanded (SCIE)

**International Journal of Data Warehousing and Mining**
Volume 16 • Issue 1 • January-March 2020

# Mining Integrated Sequential Patterns From Multiple Databases

Christie I. Ezeife, University of Windsor, Ontario, Canada
Vignesh Aravindan, Royal Bank of Canada, Canada
Ritu Chaturvedi, School of Computer Science, University of Guelph, Ontario, Canada

## ABSTRACT

Existing work on multiple databases (MDBs) sequential pattern mining cannot mine frequent sequences to answer exact and historical queries from MDBs having different table structures. This article proposes the transaction id frequent sequence pattern (TidFSeq) algorithm to handle the difficult problem of mining frequent sequences from diverse MDBs. The TidFSeq algorithm transforms candidate 1-sequences to get transaction subsequences where candidate 1-sequences occurred as (1-sequence, itssubsequenceidlist) tuple or (1-sequence, position id list). Subsequent frequent i-sequences are computed using the counts of the sequence ids in each candidate i-sequence position id list tuples. An extended version of the general sequential pattern (GSP)-like candidate generates and a frequency count approach is used for computing supports of itemset (I-step) and separate (S-step) sequences without repeated database scans but with transaction ids. Generated patterns answer complex queries from MDBs. The TidFSeq algorithm has a faster processing time than existing algorithms.

## KEYWORDS

Candidate Generation, Complex Queries, Foreign key, Frequent Itemsets, Frequent Patterns, Frequent Sequences, Multiple Databases, Sequence Database, Transaction Ids

## INTRODUCTION

Existing works are mostly for mining frequent itemsets/sequences from single databases (Han, Kamber & Pei, 2012; Nanopoulos & Manolopoulos, 2000). Work does not exist for a sequential pattern algorithm that mines exact frequent sequences from multiple tables or databases that are related through foreign key attributes. For more useful interpretation and application of frequent patterns to real life cases where patterns from different tables or databases related through foreign key attributes need to be integrated to answer relevant queries, algorithms for mining frequent sequences from multiple data sources that carry foreign key tags (e.g., transaction id) are important. Existing work on mining frequent itemsets from transaction tables can be classified into Apriori- and nonApriori-based algorithms, including the Fp-tree algorithm (Agrawal & Srikant, 1994; Srikant & Agrawal, 1995; Han, Pei, Yin & Mao, 2004). Some prominent Apriori-based sequence pattern mining (SPM) algorithms on single databases include GSP (Srikant & Agrawal, 1996). Frequent sequence mining algorithms that are non-Apriori based include SPAM and Prefix-span (Ayres, Flannick, Gehrke, & Yiu, 2002; Pei, Han, Mortazavi-asl, & Zhu, 2000). Algorithms specifically for mining Web sequential patterns include WAP-tree and PLWAP-tree algorithms (Pei, Han, Mortazavi-asl, & Zhu, 2000; Ezeife, Lu, & Liu, 2005).

DOI: 10.4018/IJDWM.2020010101

However, these single sequence/itemset database mining algorithms cannot mine frequent patterns from MDBs or tables like a database with two tables, example drug/side effects sequence table for recording drugs and their side effects with the schema DrugSE(Drugid, Sequences of side effects). The second table is patient/drug sequence table for recording sequences of drugs taken by patients with the schema PatientDr(Patientid, Sequences of Drugids). The DrugSE and PatientDr tables are related through the Drugid foreign key attribute. Regular SPM algorithms, including GSP, can be run on each of these tables. It finds the table drug/side effects with frequent sequences of side effects, as well as the table patient/drug with frequent sequences of drugs (Srikant & Agrawal, 1996). Multiple table scans provide little or no information on finding the patterns. A complex pattern query requiring associating patterns from these two tables, for example "find frequent sequences of side effects suffered from patients $p_1$ and $p_2$" cannot be directly or easily answered with these algorithms without additional post-processing database scans. Some reasons for the need to mine frequent patterns from MDBs and example queries for each category include:

1. **Comparative Analysis:** in applications like e-commerce websites where product information (e.g., product name, price) and products sold by online stores (e.g., Best Buy, Walmart) are stored in MDBs and updated frequently. An example historical query is "Find the e-commerce website that sells the cheapest Samsung television products".
2. **Frequent Local and Global Product Pattern Analysis:** There is a need to find frequent local and global patterns of products purchased from customer transaction databases with the same table structure in several local branches.
3. **Mining Frequent Patterns from Multiple Tables with Different Table or Attribute Structures:** There is a need to mine frequent itemsets/sequences from related databases with structures related through foreign/primary key attributes (i.e., patient/drugs and drugs/side effects). For example, "Find patients who are affected by frequent sequences of side effect patterns involving side effect $s_1$".
4. **Mining Alternate Types of Information:** Patterns for discovering regular product or customer behavior for targeted marketing, such as stable patterns or identifying important customers.

Existing techniques for mining frequent patterns from MDBs include algorithms mining global frequent patterns from multiple tables with the same structures for local databases. Example algorithms are the ApproxMAP algorithm (Kum, Chang, & Wang, 2006), IndividualMine (Peng & Liao, 2009), the hierarchical gray clustering algorithm (HGCA) (Lin, Hu, Li, & Wu, 2013), and clustering local frequency items in MDBs (Adhikari, 2013). An example algorithm that can mine frequent itemsets (not sequences) from MDBs with different structures is the TidFP algorithm (Ezeife & Zhang, 2009).

The main purpose of this article is to propose an algorithm for mining exact frequent sequences from MDBs with different table structures. These database structures are related through foreign key attributes, which would allow answering informative queries involving shared patterns.

## Contributions and Problem Definition

Single database sequence mining algorithms cannot mine frequent sequential patterns from multiple related sequences. In addition, they cannot integrate the results to answer queries related to MDBs. This article contributes the following features to the problem of SPM through its newly proposed algorithm (TidFSeq) and work from an unpublished thesis (Aravindan, 2016) for mining exact frequent sequential patterns from general sequences (both multiset and uniset sequences) in MDBs (with different or similar structures) using transaction ids:

1. Answers complex sequence database queries involving related data from more than one table or database.

2. Finds records sharing frequent sequences in MDBs with less database scans and faster.
3. Mines alternate types of information from competitive databases (e.g., stable, trending patterns)

Given multiple related sequence tables where each table consists of sequence id and corresponding sequence of items and a minimum support count "s", the problem of mining frequent sequences from multiple related sequence databases is that of mining the exact frequent sequences with support counts greater or equal to the given minimum support count "s" from each sequence table and to be integrated to answer queries.

### Outline

Next, the article discusses related work in mining frequent itemsets, frequent sequences, and frequent patterns from MDBs. Then, the article presents a detailed discussion of the problem addressed with the TidFSeq algorithm. The article presents performance and experimental analysis before offering a conclusion and future work.

## BACKGROUND AND RELATED WORK

Existing work on frequent pattern mining can be classified into Apriori- and nonApriori-based algorithms. The Apriori algorithm is a prominent frequent itemset mining algorithm (Agrawal & Srikant, 1994). Its extensions include algorithms using a hashing technique (Park, Chen, & Yu, 1995) and a partitioning technique (Savasere, Omiecinski, & Navathe, 1995; Taniar, Clement, Leung, Rahayu, & Goel, 2008). NonApriori-based FP-tree and extensions to the FP-growth approach also exist (Han, Pei, Yin & Mao, 2004). Prominent frequent sequence mining algorithms are Apriori-based algorithms like GSP (Srikant & Agrawal, 1996). NonApriori-based algorithms include SPAM (Ayres, Flannick, Gehrke, & Yiu, 2002) and Prefix-span (Pei, Han, Pinto, Chen & Dayal, 2004). Algorithms specifically for mining Web sequential patterns include WAP-tree (Pei, Han, Mortazavi-asl & Zhu, 2000) and PLWAP-tree (Ezeife & Lu, 2005; Ezeife, Lu & Liu, 2005). Hybrid Web SPM approaches combine Apriori and nonApriori (e.g., pattern-growth) techniques.

There are a few notable systems that focus on mining frequent patterns from MDBs (Liu, Lu, & Yao, 2001; Zhang, Wu, & Zhang, 2003; Kum, Chang, & Wang, 2006; Ezeife & Zhang, 2009; Peng & Liao, 2009; Zhang, You, Jin, & Wu, 2009; Mehenni & Moussaoui, 2012; Lin, Hu, Li & Wu, 2013). This article will provide a more detailed discussion of the ApproxMAP algorithm (Kum, Chang, & Wang, 2006), TidFp algorithm (Ezeife & Zhang, 2009), GSP algorithm (Srikant & Agrawal, 1996), and SPAM (Ayres, Flannick, Gehrke, & Yiu, 2002) in relation to the TidFseq algorithm.

### Sequential Pattern Mining Algorithms on Single Databases

#### *GSP*

GSP is an Apriori-based SPM algorithm using the downward-closure property of sequential patterns (Srikant & Agrawal, 1996). It adopts a multiple pass, candidate generate-and-test approach called the GSP-join, which is like the Apriori-gen join function of the Apriori algorithm for frequent itemset mining. Thus, given a database of frequent sequences and a minimum support threshold of two transactions, the GSP algorithm mines all frequent sequential patterns with support count greater than or equal to the minimum support (see Table 1).

GSP in the first pass determines the frequent 1-item patterns ( $L_1$ ) as items with support count greater than or equal to the given minimum support. Each subsequent pass starts with a seed set consisting of the frequent sequences found in the previous pass ( $L_{k-1}$ ). The seed set generates the k-candidate sequences ( $C_k$ ) as the frequent sequence $L_{k-1}$ GSP-joins with itself $L_{k-1}$ or written as $L_{k-1} \bowtie_{GSP_join} L_{k-1}$. The $L_{k-1}$ GSP-join $L_{k-1}$ requires that every sequence *s* in the first $L_{k-1}$ joins with

**Table 1. Sequence table for GSP**

| SID | Sequences |
|---|---|
| 1 | $\langle AB(FG)CD\rangle$ |
| 2 | $\langle BGD\rangle$ |
| 3 | $\langle BFG(AB)\rangle$ |
| 4 | $\langle F(AB)CD\rangle$ |
| 5 | $\langle A(BC)GF(DE)\rangle$ |

other sequences $s'$ in the second $L_{k-1}$ if the last *k-2* items of the first sequence *s* are the same as the first *k-2* items of the second sequence $s'$. For example, the s = ((1, 2)(3)) GSP-join $s'$ = ((2)(3,4)) gives ((1, 2) (3,4)). The s = ((1, 2)(3)) GSP-join ((2)(3)(5)) = ((1, 2)(3)(5)). Note how the two types of sequence elements of itemset sequence (I-step) such as join of (3) in s and (3,4) in s' resulted in (3,4) and not (3)(4) since one of the sequences being joined has both items together in a set. On the contrary, the second type of sequence elements have single items in sequence (S-step), such as a join of (3) in the first $s$ and (3)(5) in the second $s'$ resulting in (3)(5) and not (3,5). Thus, a join of a single item in one $s$ with either an I-step or S-step sequence in the second $s'$ will result in the I-step or S-step sequence they joined with. Following the join phase is the pruning phase, in which the candidate sequences that have any of their contiguous (k-1)-subsequences not frequent in an earlier $L_{k-1}$ are dropped because of the downward closure property. This means that this sequence would not have a chance of being frequent when the database is scanned for support.

The supports for the remaining candidate sequences determine which of the candidate sequences are frequent ( $L_k$ ). These frequent candidates become the seed for the next pass. The algorithm terminates when there are no frequent sequences at the end of a pass or when there are no candidate sequences generated. An example mining of Table **1** given minimum support count of 2 and the $C_1$ items as given below using the GSP algorithm will go through five iterations to find frequent sequences L: $L_1 \cup L_2 \cup L_3 \cup L_4$ = {A, B, C, D, F, G, AB, AC, AD, AF, AG, BC, BD, BF, BG, CD, FA, FB, FC, FD, GD, (AB), ABD, ABF, ABG, ACD, AFD, AGD, BCD, BFD, BGD, FCD, F(AB), ABFD, ABGD}. The GSP algorithm, unlike the proposed TidFSeq algorithm, suffers from generation of long candidate sequences. It is designed for a single transaction database table.

***SPAM***

The SPAM algorithm (Ayres, Flannick, Gehrke, & Yiu, 2002) with vertical bitmap representation first uses the one-candidate items of the database to construct a lexicographic tree representation of the sequential database where each of the items, such as {a, b, c, d}, forms a child node of the root node of the tree. Each of these nodes will have their children extended to generate two-candidate sequences by extending in the two ways of item I-step extension (for example, extending the one-sequence a to the two-sequence in the same set (a, b)) and sequence S-step extension (for example, extending the one-sequence a to the two-sequence in the different sets a, a or a, b). Generally, in building the tree, each node can generate sequence-extended children sequences (in S-step process) and itemset-extended children sequences (in I-step process). Each item (e.g., a, b, c, d) in the sequence database

creates a vertical bitmap showing each transaction in the database. It shows if this item is present in the transaction with a bit of "1." The bit is "0" if the item is absent.

For example, given the example transaction database of Table 2, transaction 1 will have the bitmap for items a, b, c, d in this database as 1101, 0111, and 0111 for its three subsequences (a, b, d)(b, c, d)(b, c, d), respectively. Each sequence in the sequence tree is either a sequence-extended sequence or an itemset-extended sequence. For example:

If we have a sequence sa = $\langle (a,b,c),(a,b) \rangle$, then $\langle (a,b,c),(a,b),(a) \rangle$ is a sequence-extended sequence of sa while $\langle (a,b,c),(a,b,d) \rangle$ is an itemset-extended sequence of sa. If we generate sequences by traversing the tree, then each node in the tree can generate sequence-extended children sequences and itemset-extended children sequences. The process of generating sequence-extended sequences is known as the sequence-extension step (the S-step). The process of generating itemset-extended sequences is known as the item-extension step (the I-step).

The frequency count of an S-step sequence, such as, can be obtained as the count of the results of the Bit AND operation of the inverse of Bit (a) with that of Bit (b) that are TRUE for all the transactions in the database. This means if for any transaction Bit () AND Bit (b) = 1 then the S-step sequence is present in this transaction and should be counted as +1 to the support count of the sub-sequence. On the other hand, the frequency count of an I-step sequence, such as, can be obtained as the count of the results of the Bit AND operation of the Bit (a) with that of Bit (b) that are TRUE for all the transactions in the database. This means if for any transaction Bit (a) AND Bit (b) = 1 then the I-step sequence is present and counted as +1 to the support count of the sub-sequence.

## Algorithms for Mining Patterns in MDBs

### *ApproxMAP*

ApproxMAP (Kum, Chang, & Wang, 2006) finds the approximate frequent sequences from MDBs of sequences having the same table structure. ApproxMap uses multiple alignments to force all sequences in the input database to be of equal length or have the same number of sub-sequences (or columns). For example, if there are four input sequences (<(123)(1)> and <(123)> and <(3)(4)> and <(1)(3)>), they will have the highest length of two subsequences by padding sequences such as the second sequence that has only one subsequence with an empty second subsequence to transform it as <(123)()>. Then, it will find the frequent sequences of the two subsequence columns as those items in the four sequences that have occurred up to minimum support count times. Assume the minimum support count is two, for the mining of the above four sequences. The weighted sequence support count for column one-subsequences is <(123)> because only items 1, 2, and 3 have counts greater than min support count of two. Column two has no frequent patterns from the four and the approximate frequent sequential pattern mined from the above database is <(123)>. It uses the same method to get the approximate frequent sequential patterns from a second database.

**Table 2. The SPAM sequence database**

| Tid | Sequence of Purchases |
|---|---|
| 1 | $\langle (a,b,d)(b,c,d)(b,c,d) \rangle$ |
| 2 | $\langle (b)(a,b,c) \rangle$ |
| 3 | $\langle (a,b)(b,c,d) \rangle$ |

The major drawback of this ApproxMAP algorithm is that it does not generate exact sequential patterns to answer exact queries for multiple sequence tables. Thus, it cannot handle multiple foreign key-related sequence tables with different table structures and attribute names.

**TidFp.** TidFP algorithm (Ezeife & Zhang, 2009) mines frequent itemsets from multiple sources using transaction ids for integrating patterns through set operations (i.e., intersect and union) to answer global queries involving multiple sources. Given two multiple tables such as the single item set versions of the drug/side effects in Table 5 with candidate one-item as side effects, = {1, 2, 3, 4, 5}, and the patient/drug database in Table 6 with candidate one-item as drug ids, $C_1 = \{d_1, d_2, d_3, d_4\}$, a minimum support threshold of three transactions, the TidFP algorithm mines frequent itemsets from the multiple related tables such as frequent side effects of drugs from Table 5 and frequent drugs purchased by patients from Table 6. Therefore, it can use these frequent items to answer queries like "Get the frequent drug side effects suffered by patients." The TidFP algorithm's four steps proceed as follows:

Step 1: It scans the database once and obtains all items with their transaction IDs in the format of a 1-item, list of transaction ids (Tid-list) the item occurred abbreviated as 1-item, Tid-list

tuples. For example, the tuples for the drugs/side effect sets for items shown in Table 5 are presented in the form of side effect id, list of drug ids having these side effect. Thus, the scan

of the database of Table 5 will give candidate one-item, $C_1 = \langle 1,(D_1, D_3)\rangle\langle 2,(D_2, D_3, D_4)\rangle\langle 3,(D_1, D_2, D_3)\rangle\langle 4,(D_1)\rangle\langle 5,(D_2, D_3, D_4)\rangle$. Similarly, for the second

**Table 3. Drug/side effects itemset sequences**

| Drug | Sequence of Side Effects |
|---|---|
| $d_1$ | $\langle(123)(1)\rangle$ |
| $d_2$ | $<(123)()>$ |
| $d_3$ | $\langle(3)(4)\rangle$ |
| $d_4$ | $\langle(1)(3)\rangle$ |

**Table 4. Patient/drugs itemset sequences**

| Patient | Sequence of Drugs purchased by patient |
|---|---|
| $p_1$ | $<(d_1\ d_2\ d_3)()>$ |
| $p_2$ | $\langle(d_3)(d_4)\rangle$ |
| $p_3$ | $\langle(d_1\ d_2)(d_1)\rangle$ |
| $p_4$ | $\langle(d_1)(d_3)\rangle$ |

**Table 5. Drug/side effects just itemsets**

| Drug | Sets of Side Effects |
|---|---|
| $d_1$ | 1 3 4 |
| $d_2$ | 2 3 5 |
| $d_3$ | 1 2 3 5 |
| $d_4$ | 2 5 |

**Table 6. Patient/drugs just itemsets**

| Patient | Sets of Drugs Purchased by Patient |
|---|---|
| $p_1$ | $d_1$ $d_2$ |
| $p_2$ | $d_1$ $d_2$ $d_3$ |
| $p_3$ | $d_3$ $d_4$ |
| $p_4$ | $d_1$ $d_2$ $d_4$ |

database of patients/drug id sequences for single items shown in Table 6, the candidate 1-item, $C_1$ is transformed to $<$ drug id, list of patient ids taking these drugs $>$. For Table 6, the candidate 1-item, $C_1$ tuples are: $C_1 = \langle D_1, (P_1, P_2, P_4)\rangle\ \langle D_2, (P_1, P_2, P_4)\rangle\langle D_3, (P_2, P_3)\rangle\langle D_4, (P_3, P_4)\rangle$.

**Step 2:** Compute frequent 1-item list $F_1$ with less than 3 transactions. $F_1$ tuples for Table 5 are $F_1 = \langle 2, (D_2, D_3, D_4)\rangle\langle 3, (D_1, D_2, D_3)\rangle\langle 5, (D_2, D_3, D_4)\rangle$. The $F_1$ tuples for Table 6 are $F_1 = \langle D_1, (P_1, P_2, P_4)\rangle\langle D_2, (P_1, P_2, P_4)\rangle$.

**Step 3:** Generate candidate 2-itemsets ($C_2$). Generally, $C_{i+1} = F_i$ map-gen join $F_i$ (e.g., $C_2 = F_1$ map-gen join $F_1$). For example, given items $i_1$ and $i_2$ with their Tid lists, The $\langle i_1, (tid_1, tid_2)\rangle$ (Apriori map-gen join) $\langle i_2, (tid_1, tid_5)\rangle = \langle i_1 i_2, (tid_1)\rangle$. For drugs/side effect Table 5, $C_2 = \langle 2\ 3, (D_2, D_3)\rangle\langle 2\ 5, (D_2, D_3, D_4)\rangle\langle 3\ 5, (D_2, D_3)\rangle$ and for the patients/drug in Table 6 is $\langle D_1 D_2, (P_1, P_2, P_4)\rangle$. The frequent-two itemsets $F_2$ are obtained and process continues until an empty set is met in an iteration.

Frequent itemsets and corresponding transaction ids for drugs/side effect Table 5 is $FP = \langle 2, (D_2, D_3, D_4)\rangle\langle 3, (D_1, D_2, D_3)\rangle\langle 5, (D_2, D_3, D_4)\rangle\langle 2\ 5, (D_2, D_3, D_4)\rangle$. Frequent itemsets and corresponding transaction ids for patients/drug Table 6 is:

$$FP = \left\langle D_1, \left(P_1, P_2, P_4\right)\right\rangle \left\langle D_2, \left(P_1, P_2, P_4\right)\right\rangle \left\langle D_1\ D_2, \left(P_1, P_2, P_4\right)\right\rangle$$

TidFp answers queries like: "What are possible frequent side effects suffered by patients $P_1, P_2,$ and $P_4$?" This is obtained using set operators like intersection ($\cap$) in:

$$\left\langle 3, \left(D_1, D_2, D_3\right)\right\rangle \cap \left\langle D_1\ D_2, \left(P_1, P_2, P_4\right)\right\rangle = \left\langle 3, \left(P_1, P_2, P_4\right)\right\rangle$$

This implies that Patients $P_1$, $P_2$, $P_4$ buy drugs $D_1$ and $D_2$. Drugs, have common side-effects 3. A major drawback is that the TidFp algorithm mines and handles queries for multiple related itemset transaction tables. However, it does not handle queries for multiple related sequential database tables requiring mining sequential patterns.

### Other Algorithms for Mining Patterns in MDBs

Peng and Liao (2009) proposed two algorithms for mining multiple-domain, single table sequential databases that are categorized to have co-occurred in the same time window (e.g. sequential purchase patterns from a book store and movies rented from a movie store in the same or different month time windows). However, unlike our proposed foreign key based multiple sequence database miner, TidFSeq, the algorithms in (Peng & Liao, 2009) do not mine multiple tables with different schemas that are related through foreign key attributes to answer more complex real-life queries. Also, their work did not extend any existing SPM algorithm but focused on how to combine sequential patterns after they are mined.

The HGCA algorithm mines stable patterns (Lin, Hu, Li & Wu, 2013). It defines an item "a" as stable if the item satisfies the minimum support count "s" in each of the local transaction tables (, where, is a local transaction table) that it occurs and the variation of the support count of that item "a" is less than or equal to a user-defined variation value "v".

## PROPOSED TIDFSEQ ALGORITHM FOR SEQUENTIAL MINING IN MDBS

Not all multiple tables have the same structure. There is also a need to mine frequent itemsets or sequences from multiple tables with different attribute structures but that are related through foreign key attributes. An example is the patient/drugs database in Table 4 and the drugs/side effects database in Table 3. The proposed TidFSeq algorithm, unlike the existing algorithms discussed in the related work section, is designed to mine frequent sequences from related multiple tables, including those with different attribute structures.

### Definitions

This section presents formal concepts and definitions used in the proposed algorithm.

**Definition 1.** Elements in an n-sequence(S) ( $e_{1S}, e_{2S}, \ldots e_{nS}$ ) are the subsequences of the n-length sequence with *n* ordered elements (or subsequences), $e_{1S}, e_{2S}, \ldots e_{nS}$ .

For example, given the three-sequence $\left\langle \left(d_1\ d_2\ d_3\right)\left(d_1\right)\left(d_4\right)\right\rangle$, the first element (or subsequence) $e_{1S}$ is ( $d_1\ d_2\ d_3$ ). It has three items in its set. The second element $e_{2S}$ is ( $d_1$ ). It has one item. The third element $e_{2S}$ is ( $d_4$ ). It also has one item in the third set.

**Definition 2.** Position id list ( $\mathrm{Pid}_{\mathrm{I_{kS}}}$ ) of a kth candidate one-item in a database sequence S specifies all the sequence elements of *S* where the kth candidate one-item occurred. In other words, it specifies all the subsequences of sequence *S* where the kth candidate one-item has occurred.

For example, given the sequence with Sid $S_1 = \left\langle \left(d_1\ d_2\ d_3\right)\left(d_1\right)\left(d_4\right)\right\rangle$, the one-item $d_1$ in this sequence is found in the first element ( $d_1\ d_2\ d_3$ ) and the second element ( $d_1$ ). Item "$d_1$" in this sequence is in positions $e_1$ and $e_2$. Thus, the sequence $S_1$'s position id list for item $d_1$ ( $Pid_{I_{d1;S1}}$ ) is $e_1$ and $e_2$. Similarly, the position id list of item $d_3$ is $e_1$ because it is found only in the first element ( $d_1\ d_2\ d_3$ ).

**Definition 3.** For $\left\langle \mathrm{Sequence\ id, Position\_id\ list} \right\rangle$ tuple, each item/sequence is associated with the sequence ids in which they occur, as well as the position ids in which they occur in each sequence id. This information is represented in the form of a tuple associated with each item/sequence. If an item "1" appears in sequence ids sid1 (the position ids of item 1 in sid 1 are $e_1, e_3$ ) and $sid_3$ (the position ids of item 1 in sid 3 are $e_1, e_2$ ), then $\left\langle sid, pid\_list \right\rangle$ tuple for item 1 is given as $\left\langle \left(sid_1, \left(e_1, e_3\right)\right), \left(sid_3, \left(e_1, e_2\right)\right)\right\rangle$.

**Definition 4.** For the I-step and S-step sequences, the I-Step sequence and S-Step terms were first defined (Ayres, Flannick, Gehrke, & Yiu, 2002) as:

1. I-Step sequence is a sequence of the form (a b) such as (1 2 3), meaning items 1, 2, and 3 occur together in a subsequence.
2. S-Step sequence is a sequence of the form (a) (b) such as (1) (2) (3), meaning items 1, 2, and 3 occur separately in a sequence.

**Definition 5.** For candidate 1-item tuples tuples, the Candidate 1-item tuples list is the representation of the input items. It lists each candidate 1-item with its position id list showing the occurrences of all subsequences. The tuples are of the form for a database with items.

For example, the tuples list for the drugs/side effects database of Table 3 in the form of <side effect id: its drug position id list> is:

$C_1$ = {1: <($d_1$, ($e_1$, $e_2$)), ($d_2$, ($e_1$)), ($d_4$, ($e_1$))>, 2: <$d_1$, ($e_1$)), ($d_2$, ($e_1$))>, 3: <$d_1$, ($e_1$)), ($d_2$, ($e_1$)), ($d_3$, ($e_1$)), ($d_4$, ($e_2$))>, 4: <($d_3$, ($e_2$))>}.

**Definition 6.** The support count computation rule for I-step sequences is the basic rule for computing support count of an I-step sequence of the form (ab). For example, "If items $a$ and $b$ have the same sequence ids and occupy the same column (subsequence) positions (i.e., have the same position ids), then support count of sequence (ab) is incremented by 1".

**Definition 7.** The support count computation rule for S-step sequences is the basic rule for computing support count of an S-step sequence of the form (a)(b). For example, "If items $a$ and $b$ have the same sequence ids and $a$ occupies an earlier (subsequence) column position than that of $b$ (i.e., position id of $a$ is less than position id of $b$ ), then support count of sequence (a)(b) is incremented by 1".

## Steps in the Proposed TidFseq Algorithm

The formal algorithm for mining frequent sequential patterns from two or more related database tables (called TidFseq) is given as algorithm 1. Details of its steps are included in this section.

**Input to the Algorithm.** Multiple related sequence database table MDBs, the input user defined minimum support count "s," and the set of candidate 1-items for each database. For example, input of db1 = Drug/sequence of side effects, db2 = Patient/drugs sequence tables, user-defined minimum support count = 2, $C_1^{db1}$ for db1 = {side effects ids} = {1, 2, 3, 4, 5}, $C_1^{db2}$ for db2 = {drug ids} = $\{d_1, d_2, d_3, d_4\}$.

**Output of the Algorithm.** Frequent sequences of each database $FS^{db_1}$, $FS^{db_2}$, …, $FS^{db_n}$ with the associated subsequence ids where they occurred listed in the form of { $F_{n-sequence}$ : < subsequence that $F_{n-sequence}$ occurred >}.

**Other Data.** Initial iteration k = 1. For example, mining the proposed algorithm TidFseq on the input databases of $db_1$ and $db_2$ yields the following output frequent sequential patterns. For db1, the frequent sequences of side effects $FS^{db1}$ = {(1): < $d_1$, $d_2$, $d_4$ >, (2): < $d_1$, $d_2$ >, (3): < $d_1$, $d_2$, $d_3$, $d_4$ >, (1, 2): < $d_1$, $d_2$ >, (1, 3): < $d_1$, $d_2$ >, (2, 3): < $d_1$, $d_2$ >, (1, 2, 3): < $d_1$, $d_2$ >}. For *db*2, the frequent sequences of drugs $S_{db2}$ = {($d_1$): < $p_1$, $p_3$, $p_4$>, ($d_2$): < $p_1$, $p_3$ >, ($d_3$): < $p_1$, $p_2$, $p_3$, $p_4$ >, ($d_1$, $d_2$): < $p_1$, $p_3$ >, ($d_1$)($d_3$): < $p_3$, $p_4$ >}.

**Algorithm 1:** (TidFSeq() - Mines multiple related table sequences)

**Input:** Multiple related sequence tables $TB_1$, $TB_2$, …, $TB_N$ and their corresponding candidate one-items sets $C_{1_{TB_K}}$, min-support count "s."

**Output:** Frequent sequences $FP_{TB_K}$ and their associated sequence ids in the form $< (FS_1 : ssid_1, \ldots, ssid_n), \left(FS_2 : ssid_1 \ldots, ssid_m\right), \ldots \left(FS_p :, ssid_1 \ldots, ssid_q\right) >$ where $FS_i$ is frequent sequence i and the $ssid_m$ is the mth subsequence id that $FS_i$ occurred.

**Other variables:** $C_k$ candidate k-sequences, $F_k$ k-Frequent sequences, $pid\_list_{sid_j}$, an array of sequence $sid_j$'s elements or subsequence ids, k=1 initially, $F_p$-final list of frequent sequences.

Begin

1. for each database table sequence $TB_i$ do Begin

    1.1 Scan the sequence table to compute the candidate 1-sequences with their position id lists in the form of $C_1 = \{item_1 : \langle sid_1 : pid\_list_1 \rangle item_2 : \langle sid_2 : pid\_list2 \rangle \quad \ldots item_n : \langle sid_n : pid\_list_n \rangle\}$, where k = 1 and $item_1, \ldots item_n \in C_1$ in the table sequence $TB_i$

    1.2 Compute Frequent k-sequences ($F_k$) as sequences with support greater than or equal to minsupport "s".

    1.2.1 if $F_k = \varnothing$ then go to step 1.7.

    1.3 k = k + 1 //to prepare for the next iteration

    1.4 Compute the next k-candidate ($C_k$) sequence as: $C_k = F_{k-1} \bowtie_{GSP-join} F_{k-1}$

        1.4.1 if $C_k = \varnothing$ then go to step 1.7.

    1.5 while (candidate k-seq $C_k \neq \varnothing$) do Begin

        1.5.1 Compute Frequent k-sequences ($F_k$) as those with support >= minsupport "s".

```
          for each sequence S ∈ C_k do Begin
               1.5.1.1 If sequence S is I-step sequence of the
form (a b) then
 Call function I-step Pruning of Algorithm 2
               1.5.1.2 else if Sequence S is I-step sequence of
the form (a)(b) then
                Call function S-step Pruning of Algorithm 3
          end //for each sequence S loop
          end //while loop 1.5
     1.6 if (F_k ≠ ∅) then go to step 1.3
     1.7 F_p = F_1 ∪ F_2 ∪ ... ∪ F_k .
end //for each database 1
end // of TidFSeq //
```

The process of mining the two databases are presented formally in Algorithm 1 and discussed in the following steps. For each input database, mining the database includes:

**Step 1:** Get candidate 1-item ($C_1$) sequence set. Scan the database table once to gather the transaction ids for each candidate 1-item as $C_1$ tuples in the form of <1-Sequenceid, Position_idlist> tuples for each 1-item in $C_1$ set.

**Step 2:** Find the frequent 1-items $F_1$ from the $C_1$ sequence as those with more than minimum support count transaction ids (e.g., two for the example databases). The occurrence count of each one-item is easily obtained from the $C_1$ by counting the number of transaction ids in its Tid list (e.g., of $d_1, d_2,$ and $d_4$ for side effect 1 means 1 has support 3).

**Step 3:** Set the next iteration k = k + 1 to compute higher level frequent k-item patterns.

**Step 4:** Generate the candidate k-sequence $C_k$ using an adapted version of the GSP-gen join function. Here, candidate sequences are generated using the GSP's join function as: $C_k = F_{k-1} \bowtie_{GSP-gen} F_{k-1}$ (Srikant & Agrawal 1996) with our newly defined I-step and S-step join conditions for sequences from MDB related through foreign key attribute based on definitions 6 and 7. If the computed $C_k$ is an empty set, the iteration ends by going to step 7.

**Step 5:** Compute frequent k-sequences $F_k$ from the $C_k$ from Step 4 using two different functions for counting the supports of the two types of sequences called I-step and S-step sequences with I-step pruning and S-step pruning functions, respectively. After the candidate generation step, if the candidate sequence is of the form (a b) (i.e., itemset together), then it is an I-step sequence and I-step pruning algorithm is called to count its support. Otherwise, if the sequence is of the form (a)(b) (items a and b separately purchased), then it is an S-step sequence and S-step pruning algorithm is called to count its support. The basic rule for computing support count of an I-step sequence of the form (ab) is: "If items $a$ and $b$ have the same sequence ids and occupy the same column (subsequence) positions (i.e., have the same position ids), then support count of sequence (ab) is incremented by 1". The basic rule for computing support count of an S-step sequence of the form (a)(b) is: "If items $a$ and $b$ have the same sequence ids and $a$ occupies an earlier (subsequence) column position than that of $b$ (i.e., position id of $a$ is less than position id of $b$), then support count of sequence (a)(b) is incremented by 1." These two rules are implemented by the two called functions of I-step pruning and S-step pruning respectively. If the computed $F_k$ is an empty set with no sequences, the iteration ends by going to Step 7.

**Step 6:** Find the higher order frequent k-sequence after the I-step and S-step by pruning functions return to the main algorithm. To compute the next candidate (k+1)-sequence and frequent (k+1)-

sequence, the algorithm goes back to Step 3, in which k is set to k+1 and the remaining steps are run iteratively until either a $C_k$ or $F_k$ generation step yields an empty set of sequences.

**Step 7:** Find the final set of frequent sequential patterns. The final output is the frequent sequential patterns consisting of the union of all the frequent n-sequences for $1 \le n \le k$. This is like $FP = F_1 \cup F_2 \cup \ldots F_k$.

This result set of two or multiple tables answers user queries for the input-related sequence tables, including "what are frequent sequences of side effects affecting patients $p_1$ and $p_3$?"

### Algorithm 2: I-step Pruning (counting support of I-step sequences)

The I-step pruning algorithm computes the support count of I-step sequences after each iteration of generating the extended candidate (k+1)-sequences from the frequent k-sequences. Thus, this algorithm takes the (k+1)-sequences and counts the support of I-step sequences of the form (a, b) so that it will return only those I-step sequences that are frequent with support greater than or equal to the given minimum support count using Definition 6. It goes through the two steps.

**Step 1: Compute the Support Count for the I-step Candidate Sequences of Form (a, b).** This is done using Definition 6. For example, assume there is a candidate I-step sequence (1, 2) and the $<$ Sequence id, Position_id list $>$ tuples for item 1 is $\left\langle \left(sid1, pos1\right)\left(sid2, pos1\right)\right\rangle$ (i.e., item 1 occurs in sid1 at position id: pos1 and also occurs in sid2 at position id: pos1) and tuples for item 2 are $\left\langle \left(sid1, pos1\right)\left(sid2, \left(pos1, pos4\right)\right)\right\rangle$ (i.e., item 2 occurs in sid1 at position id: pos1 and also occurs in sid2 at position ids: pos1, pos4). From the tuples, we can see that item 1 and item 2 have two matching sequence ids (i.e. sid 1 and sid2). The corresponding position ids also match (i.e., items 1 and 2 occur at pos 1 in sid1 and again occur at pos1 in sid2). The $\left\langle Sequenceid, Positionidlist\right\rangle$ tuple for I-step sequence (1, 2) is $< \left(sid1, pos1\right)\left(sid2, pos1\right) >$. The support count of the I-step sequence is 2.

**Step 2: Checking Whether the I-Step Sequence is Frequent.** The support count computed in the previous step is checked for whether it satisfies the minimum support count (i.e., if the support count of I-step sequence is greater than or equal to minimum support count). If the minimum support count is 2, then the I-step sequence (1 2), whose support count calculated in previous step, is 2, satisfies the minimum support count and is frequent. Output of I-step pruning (): The frequent sequence and its < Sequence id, Position_id > list tuples will be added to the result set that will be returned to the main program.

### Algorithm 3: S-step Pruning (counting support of S-step sequences)

The S-step pruning algorithm (Algorithm 3) accepts as its input the S-step sequences of the form (a) (b). It returns the S-step sequences that are frequent and have support counts greater than or equal to the given minimum support count. In counting the support of each S-step sequence, it applies the S-step support count rule defined in the definition section. S-step input data are the S-step candidate sequences of the form (a) (b) generated by candidate generation method of the main algorithm (Algorithm 1) and min-support count ("s"). Its output consists of the S-step frequent sequences with the two steps below:

**Step 1: Compute Support Count for the S-Step Candidate Sequence of the Form (a)(b).**

This is done using the rule of Definition 7. For example, assume there is an S-step candidate sequence (1) (2) and the $\left\langle Sequenceid, Position\_idlist\right\rangle$ tuples for item 1 is $\left\langle \left(sid_1, pos_1\right)\left(sid_2, pos_2\right)\right\rangle$ (i.e., item 1 occurs in $sid_1$ at sub sequence position id: $pos_1$ and also occurs in $sid_2$ at position id: $pos_2$) and tuples for item 2 are $< \left(sid_1, pos_3\right)\left(sid_2, \left(pos_1, pos_4\right)\right) >$ (i.e., item 2 occurs in $sid_1$ at sub

sequence position id = $pos_3$ and also occurs in $sid_2$ at position ids= $pos_1$, $pos_4$). From the tuples, we can see that item 1 and item 2 have a total of two matching sequence ids (i.e., $sid_1$ and $sid_2$). The corresponding position ids of item 1 are less than the corresponding position ids of item 2 (i.e., the position id ($pos_1$) of item 1 in $sid_1$ is less than position id ($pos_3$) of item 2 in $sid_1$ and the position id ($pos_2$) of item 1 in $sid_2$ is less than position id ($pos_4$) of item 2 in $sid_2$). The tuples for S-step sequence (1) (2) are $<\left(sid_1,\left(pos_1,pos_3\right)\right)(\left(sid_2,\left(pos_2,pos_4\right)\right)>$ and the support count of the S-step for the sequence (1) (2) is 2.

**Step 2: Checking Whether the S-Step Sequence is Frequent.** The support count computed in the previous step is checked for whether it satisfies the minimum support count. If the minimum support count is 2, then the S-step sequence (1) (2), whose support count calculated in previous step, is 2, satisfies the minimum support count and is frequent.

## EXAMPLE APPLICATION OF THE TIDFSEQ ALGORITHM

**Input.** Multiple related sequence tables $db_1$ = Drug/sequence of side-effects (see Table 3), $db_2$ = patient/drugs sequence (given in Table 4, user-defined minimum support count=2, $C_1^{db1}$ for $db_1$ = {side effects ids} = {1, 2, 3, 4}, $C_1^{db2}$ for $db_2$ = {drug ids} = { $d_1, d_2, d_3, d_4$ }.

**Output of the Algorithm.** Frequent sequences of each database $FS^{db_1}$, $FS^{db_2}$, …, $FS^{db_n}$, with the associated subsequence ids where they occurred listed in the form of { $F_{n-sequence}$ :< Ssids that $F_{n-sequence} occurred >$ }. Other data: Initial iteration k = 1.

The process of mining with these two database tables using the TidFseq algorithm goes through the following steps for each database.

**Step 1:** Generate the < Sequence id, Position id (Pid) list > tuple for each 1-item in the $C_1$ to get the $C_1$ sequence set in vertical format for the drugs/side effect database of Table 3 shown in Table 7, and of the patient/drugs database of Table 4 shown in Table 8.

For example, in Table 3 (shown here in Table 7), we can see that item "1" has a tuple of sequence ids (i.e., $d_1, d_2,$ and $d_4$) and the corresponding position ids for each sequence id (i.e., pid $e_1, e_2$, in sid $d_1$, pid $e_1$ in sid $d_2$, pid $e_1$ in sid $d_4$).

**Step 2:** Find frequent ($F_1$) as $F_1^{DB_1}$ = {side effects} = {1, 2, 3}. $F_1^{DB_2} = \left\{drugs\right\} = \left\{d_1, d_2, d_3\right\}$.

**Step 3:** Since $F_1$ are not empty set, we have k = 2.

**Step 4:** Finding candidate 2-sequences $C_2$: $C_2^{DB_1} = F_1^{DB_1} \bowtie_{GSP-gen} F_1^{DB_1}$. $C_2^{DB_2} = F_1^{DB_2} \bowtie_{GSP-gen} F_1^{DB_2}$. Using the rules for GSP join, the candidate 2-sequences for the input $DB_1$ sequence Drug/Side effects table, Table 3 are: $C_2^{DB1}$ = {(1)(1), (1)(2), (1)(3), (1,1), (1, 2), (1, 3), (2)(1), (2)(2), (2)(3), (2, 1), (2, 2), (2, 3), (3)(1), (3)(2), (3)(3), (3, 1), (3, 2), (3, 3) }. The candidate 2-sequences for the input $DB_2$ sequence table, Table 4 are: $C_2^{DB2}$ = {($d_1$)($d_1$), ($d_1$)($d_2$), ($d_1$)($d_3$), ($d_1$, $d_1$), ($d_1$, $d_2$), ($d_1$, $d_3$), ($d_2$)($d_1$), ($d_2$)($d_2$), ($d_2$)($d_3$), ($d_2$, $d_1$), ($d_2$, $d_2$), ($d_2$, $d_3$), ($d_3$)($d_1$), ($d_3$)($d_2$), ($d_3$)($d_3$), ($d_3$, $d_1$), ($d_3$, $d_2$), ($d_3$, $d_3$)}.

**Step 5/6:** Computing frequent 2-sequences $F_2$ by finding frequent I-step (of the form (1 2)) and S-step (of the form (1)(2)) sequences that are frequent. In the example, the candidate I-step sequence (1, 2) has the following < Sequence id, Position_id list > tuples for item 1 and item 2: Item 1 as: $\left\langle\left(d_1,\left(e_1,e_2\right)\right),\left(d_2,e_1\right),\left(d_4,e_1\right)\right\rangle$ (i.e., item 1 occurs in sid: $d_1$ at position id: $e_1, e_2$ and also occurs in sid: $d_2$ and sid: $d_4$ at position id: $e_1$) and Item 2: $\left\langle\left(d_1,e_1\right),\left(d_2,e_1\right)\right\rangle$ (i.e., item 2 occurs in sid: $d_1$ at

**Table 7. Transformed $C_1$ item sequences for drug/side effects db1**

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| Sid Pid_list | Sid Pid_list | Sid Pid_list | Sid Pid_list |
| $d_1$ $e_1$ $e_2$ | $d_1$ $e_1$ | $d_1$ $e_1$ | $d_3$ $e_2$ |
| $d_2$ $e_1$ | $d_2$ $e_1$ | $d_2$ $e_1$ | |
| $d_4$ $e_1$ | | $d_3$ $e_1$ | |
| | | $d_4$ $e_2$ | |

**Table 8. Transformed item sequences for patient/drugs db2**

| $d_1$ | $d_2$ | $d_3$ | $d_4$ |
|---|---|---|---|
| Sid Pid_list | Sid Pid_list | Sid Pid_list | Sid Pid_list |
| $p_1$ $e_1$ | $p_1$ $e_1$ | $p_1$ $e_1$ | $p_2$ $e_2$ |
| $p_3$ $e_1$ $e_2$ | $p_3$ $e_1$ | $p_2$ $e_1$ | |
| $p_4$ $e_1$ | | $p_3$ $e_3$ | |
| | | $p_4$ $e_2$ | |

position id: $e_1$ and also occurs in sid: $d_2$ at position id: $e_1$). From the tuples, it can be seen that item 1 and item 2 have a total of two matching sequence ids and position ids (i.e., item 1 and item 2 occur at position id: $e_1$ in sid: $d_1$ and occur at position id: $e_1$ in sid: $d_2$). The condition for I-step sequence is met twice. Therefore, the support count of sequence (1, 2) is 2, which satisfies the input support count (2). Hence, I-step sequence (1, 2) is a frequent sequence. Running the I-step sequence prune algorithm on the $C_2^{DB_1}$ yields the following frequent $F_2$ I-step sequences for DB1. $F_2^{DB1} = \{(1,2),(1,3),(2,3)\}$ for only I-step sequences. Running the I-step sequence prune algorithm on the $C_2^{DB_2}$ yields the following frequent $F_2$ I-step sequence for $DB_2$. $F_2^{DB_2} = \{(d_1,d_2)\}$ for only I-step sequences. If items of a candidate sequence occur separately then this is an S-step sequence of the form (a) (b). If "a" and "b" have same sequence ids and "a" occupies earlier position than that of "b," in greater than or equal to min-support number of sequences, then (a)(b) is frequent. In the example $db_2$, candidate S-step sequence $(d_1)(d_3)$ has the following $<$ Sequence id, Position_id list $>$ tuples for item $d_1$ and item $d_3$ in the same S-step sequence: Item $d_1$ occurred in transactions $p_1, p_3$ and $p_4$ with the given position id lists, $\langle(p_1,e_1),(p_3,(e_1,e_2)),(p_4,e_1)\rangle$ (i.e., item $d_1$ occurs in

sid: $p_1$ at position id: $e_1$ and occurs in sid: $p_3$ at position ids: $e_1, e_2$ and also occurs in sid: $p_4$ at position id: $e_1$). The pid list for item item $d_3$ is: $\langle (p_1,e_1),(p_2,e_1),(p_3,e_3),(p_4,e_2) \rangle$ (i.e., item $d_3$ occurs in sid: $p_1, p_2, p_3, p_4$ at position ids: $e_1, e_1, e_3, e_2$ respectively). From the tuples, items $d_1$ and $d_3$ have matching sequence ids $p_3$ and $p_4$. For sid $p_3$, corresponding pids of item $d_1$ (i.e., $e_1, e_2$) is less than the corresponding pid of item $d_3$ (i.e., $e_3$) and similarly for sid $p_4$: corresponding pid of item $d_1$ (i.e., $e_1$) is less than corresponding pid of $d_3$ (i.e., $e_2$). Since the condition for S-step sequence is met twice, the support count of sequence $(d_1)(d_3)$ is 2, which satisfies the input support count (2). Hence, the result of running the S-step sequence algorithm on $C_2^{Db_2}$ is $\{(d_1)(d_3)\}$. The result of running the S-step sequence algorithm on $C_2^{DB_1}$ is $\varnothing$. Thus, the complete set of frequent sequences for both I-step and S-step sequences is $F_2^{DB_1} = \{sideeffects\} = \{(1,2),(1,3),(2,3)\}$ and $F_2^{DB_2} = \{drugs\} = \{(d_1,d_2),(d_1)(d_3)\}$.

**Step 5 (again):** Find higher order 3-sequences iteratively. $C_3^{DB_1} = F_2^{DB_1} \bowtie_{GSP-gen} F_2^{DB_1} = C_3^{DB_2} = F_2^{DB_2} \bowtie_{GSP-gen} F_2^{DB_2}$. Using the rules for GSP join, the candidate 3-sequences for the input DB1 sequence Drug/Side effects table, Table 3 are: $C_3^{DB_1} = \{(1,2,3)\}$. Running the I-step and S-step prune algorithms on this set also confirms it frequent so that $F_3^{DB_1} = \{(1,2,3)\}$. The candidate three-sequences for the input DB2 sequence patient/drugs table, Table 4 are: $C_3^{DB_1} = \varnothing$ and the iteration terminates for this Db2.

**Step 6 (again):** Find higher order 4-sequences for DB1 iteratively. Since $F_3^{DB_1} = \{(1,2,3)\}$, $C_4^{DB_1} = F_3^{DB_1} \bowtie_{GSP-gen} F_3^{DB_1} = \varnothing$. Now the iteration for DB1 also terminates.

**Step 7:** The full mined sequential patterns from $DB_1$ of (drugs, sequence of side effects) data in the form of (frequent sequence of side effects: sequence ids (drugs) they occurred in are: $F^{DB1}$ = { (1: $d_1$, $d_2$, $d_4$), (2: $d_1$, $d_2$), (3: $d_1$, $d_2$, $d_3$, $d_4$), ((1, 2): $d_1$, $d_2$), ((1, 3): $d_1$, $d_2$), ((2, 3): $d_1$, $d_2$), ((1, 2, 3): $d_1$, $d_2$)}. The full mined sequential patterns from $DB_2$ are: $F^{DB2}$ = { ($d_1$: $p_1$, $p_3$, $p_4$), ($d_2$: $p_1$, $p_3$), ($d_3$: $p_1$, $p_2$, $p_3$, $p_4$), (($d_1$,$d_2$): $p_1$, $p_3$), (($d_1$,$d_3$): $p_3$, $p_4$)}. Sample query handled by the proposed algorithm for the given input tables: What are the possible frequent sequence of side effects that the patients $p_1$ and $p_3$ suffer from? Answer: Patients $p_1$ and $p_3$ have purchased drugs $d_1$ and $d_2$ together. And the drugs $d_1$ and $d_2$ cause side effects 1, 2, and 3 to occur together. This answer can be derived by intersecting the result sets of the input tables from $DB_2$ of (patients, sequence of drugs purchased) and frequent pattern results of $DB_1$ (drugs, sequence of side effects) as: $\langle (d_1\ d_2), p_1 p_3 \rangle \cap \langle (1\ 2\ 3), d_1\ d_2 \rangle = \langle (1\ 2\ 3), p_1\ p_3 \rangle$. We clearly see how the proposed algorithm is able to solve such complex queries for multiple related sequence tables that the existing systems are not able to achieve.

## PERFORMANCE AND EXPERIMENTAL EVALUATION

This section compares the experimental performance analysis of the TidFSeq algorithm with the ApproxMap algorithm and PrefixSpan algorithm. The three algorithms were implemented with Java language running under Eclipse environment. All experiments were performed on Intel(R) CORE i7-4700HQ CPU @ 2.40 Ghz. The operating system is Linux. Synthetic datasets are generated using the

publicly available synthetic data generation program of the SPMF library at www.philippe-fournier-viger.com/spmf/index.php. The results obtained by the TidFSeq algorithm, ApproxMap algorithm, and PrefixSpan algorithm for mining frequent sequential patterns on multiple database tables were compared. Table 9 shows the drug/side effects database 1 representing Table 3 for the drugs/side Effects sequence database. Table 10 shows the patient/drugs sequence database 2 representing Table 4. These small sequence datasets represent two multiple related sequence tables that have been used to describe the proposed technique.

These datasets show how the proposed algorithm mines the frequent sequences from both datasets to solve a sample query that the ApproxMap and PrefixSpan algorithms are not able to solve. Note that the datasets in the input data file are in the form of a text file. Each row represents a sequence and each itemset in sequence is separated by "-1" while "-2" represents the end of the sequence. For example, 1 2 3 -1 2 -1 -2 in the input data file represents the sequence (1, 2, 3) (2). This sequence contains the two itemsets (1, 2, 3) and (2). The standard sequence datasets do not contain transaction ids (i.e., sequence ids), in the experiment, we assume that each row represents a sequence and has an associated sequence id. For reference purposes, the sequence ids are included in both the datasets in Table 9 for the drug/side effects database 1 and Table 10 for patient/drugs database 2 before the start of each sequence. The programs TidFseq and PrefixSpan gave the exact mined frequent sequential patterns for the two databases as found in the example running of the TidFseq algorithm, but only the TidFseq generated sequential patterns carry their associated transaction or sequence ids that can be used to link up frequent patterns in other related tables through foreign key relationship. Also, the ApproXMap was only able to find approximate sequential patterns and not exact patterns and is not able to link patterns in tables with different attribute structures. Only the proposed TidFseq algorithm is able to answer complex queries involving patterns from more than one table such as: What are the possible frequent sequences of side effects that the patients $p_1$ and $p_3$ suffer from? This query can be answered as an intersection of the generated frequent sequential patterns using their foreign key transaction id link through an SQL SELECT instruction with a join of the patterns on the foreign key as: SELECT Output-db1.side-effects, Output-db2.patients FROM Output-db1, Output-db2 WHERE Output-db1.drugs = Output-db2.drugstaken;

**Table 9. Drug/side effects table**

| Drug | Sequence of Side Effects for Experiment |
|---|---|
| d1 | 1 2 3 -1 1 -1 -2 |
| d2 | 1 2 3 -1 -2 |
| d3 | 3 -1 4 -1 -2 |
| d4 | 1 -1 3 -1 -2 |

**Table 10. Patient/drugs table**

| Patient | Sequence of Drugs Purchased by Patient for Experiment |
|---|---|
| p1 | d1 d2 d3 -1 -2 |
| p2 | d3 -1 d4 -1 -2 |
| p3 | d1 d2 -1 d1 -1 -2 |
| p4 | d1 -1 d3 -1 -2 |

Answer to this query from the generated patterns is: (1, 2, 3) $p_1$ $p_2$ indicating that patients $p_1$ and $p_3$ suffer from frequent sequence of side effects (1, 2, 3).

The following experiments are carried out on synthetic datasets generated with the SPFM library (Fournier-Viger, 2016) containing 250K, 500K, 750K, 1M, and 2M sequences over different support counts 10%, 20%, 30%, 40%, 50%. Subsections provide the comparison of the three algorithms based on:

1. Execution speed (speed of processing) of the algorithms for datasets containing 250K, 500K, 750K, 1M, and 2M sequences over support counts of 10%, 20%, 30%, 40%, 50%.
2. Memory usage (in MB) for different data sizes at Minsupport of 40%.
3. Accuracy of the frequent sequences obtained for datasets containing 250K, 500K, 750K sequences at minsupport of 40%.
4. Execution speed for frequent sequences obtained for 250K sequence dataset having sequences of length greater than 10 elements.

## Comparison of Execution Speed (Speed of Processing) of the Algorithms

Table 11 provides the result of comparing the CPU execution times of the three algorithms, and it can be seen that TidFseq algorithm performs considerably better than ApproxMap and PrefixSpan algorithms in terms of runtime for increasing data sizes (i.e., increasing number of sequences). The ApproxMap has to scan the input database twice (first time to make all sequences of equal length and second time to scan the pre-processed database for frequent approximate sequences) compared to the TidFseq algorithm that scans database once to get the sequence id and position id list tuples for the items.

## Execution Times (in Secs) for Small Dataset (250K) at Different Supports

Table 12 shows the result of comparing the CPU execution times of the three algorithms for mining single table sequences for small-sized dataset (250K) for different support counts. It can be seen that with increasing support count, the run times for all the algorithms reduce. However, the TidFseq algorithm clearly has better run times than the ApproxMap and PrefixSpan algorithms, especially when support is less than or equal to 30%.

**Table 11. Execution times (in secs) for different datasets sizes at MinSupport of 40%**

| Algorithm | 250K | 500K | 750K | 1M | 2M |
|---|---|---|---|---|---|
| TidFSeq | 34.32 | 59.2 | 78.45 | 113.56 | 145.62 |
| ApproxMap | 66.2 | 72.1 | 98.7 | 156.9 | 220.15 |
| PrefixSpan | 70 88 | 107.83 | 331 | 560.45 | |

**Table 12. Execution times (in secs) for small datasets at different supports**

| *Algorithm* | 10% | 20% | 30% | 40% | 50% |
|---|---|---|---|---|---|
| TidFSeq | 2905.1 | 792 | 73.03 | 34.32 | 14.2 |
| ApproxMap | 3866.6 | 1072.1 | 128.7 | 66.2 | 41.5 |
| PrefixSpan | 3869.09 | 1073 | 141 | 78.4 | 46.66 |

### Execution Times (in Secs) for Large Dataset (2M) at Different Supports

Table 13 shows the results of comparing the run times of the three algorithms for large datasets of two million sequences for different support counts. When processing very large data and when support counts are less, the run times for all three algorithms are also bigger. According to Table 13, the TidFseq algorithm runs faster than the ApproxMap and PrefixSpan algorithms, including higher support counts, when data is large.

## CONCLUSION AND FUTURE WORK

This article proposes the TidFSeq algorithm, which extends the techniques of the TidFP algorithm for mining itemsets to the more challenging problem of mining frequent sequences from MDBs to answer more complex queries from multiple related sequence tables. A new technique of using < sequence ids, position id_list > tuples was derived from the < transaction id, itemsets > tuples used in the TidFp algorithm to mine frequent sequences from multiple tables. It proposes new algorithms for implementing the support count of general sequences consisting of I-step sequence types of the form (a, b) and S-step sequence types of the form (a)(b) which would efficiently use the subsequence occurrence ids (called column element id and position id lists) of each 1-item to correctly count support for any extension of item sequences. The I-step prune and S-step prune algorithms for counting supports of sequences with their position ids are contributions. An adaptation of the GSP-gen join for extending higher-order frequent sequences carrying their position id lists from the only one database scan was used. Using the frequent sequences and their corresponding sequence ids mined from multiple related sequence tables, valuable user queries for multiple related sequence tables can be answered.

Future work should consider other ways of storing sequences, including hashing methods. This can be used rather than memory consuming data tuples. A parallel processing framework, including a Map-Reduce framework, can process multiple tables in parallel and data can be stored in a distributed storage system like Hadoop distributed file system to handle big data and growing sequences. The proposed algorithm can be extended to mine frequent sequences using compact tree-based approaches and condensed sequences, including maximal frequent sequence patterns.

## ACKNOWLEDGMENT

This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

**Table 13. Execution times for large dataset (2M) for different support count**

| Algorithm | 10% | 20% | 30% | 40% | 50% |
|---|---|---|---|---|---|
| TidFSeq | 30216.66 | 4876.23 | 701.01 | 145.62 | 70.89 |
| ApproxMap | 39994.96 | 5172.41 | 1008.5 | 220.15 | 142.3 |
| PrefixSpan | 41101 | 9908.4 | 5640.11 | 1590 | 896.32 |

## REFERENCES

Adhikari, A. (2013). Clustering local frequency items in multiple databases. *Journal of Information Science*, *237*, 221–241. doi:10.1016/j.ins.2013.02.043

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, & C. Zaniolo (Eds.), *Proceedings of the 20th International Conference on very Large Databases* (pp. 487-499). San Francisco, CA: Morgan Kaufmann.

Aravindan, V. (2016). *Mining frequent sequential patterns from MDBs using transaction Ids* [MSc thesis]. University of Windsor, Canada.

Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002). Sequential pattern mining using a bitmap representation. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 429-435). New York, NY: ACM. doi:10.1145/775047.775109

Ezeife, C. I., & Lu, Y. (2005). Mining Web log sequential patterns with position coded pre-order linked WAP-tree. *International Journal of Data Mining and Knowledge Discovery*, *10*(1), 5–38. doi:10.1007/s10618-005-0248-3

Ezeife, C. I., Lu, Y., & Liu, Y. (2005). PLWAP sequential mining: Open source code. *Proceedings of the Open Source Data Mining Workshop on Frequent Pattern Mining Implementations* (pp. 26-29). ACM. doi:10.1145/1133905.1133910

Ezeife, C. I., & Zhang, D. (2009). TidFP: Mining frequent patterns in different databases with transaction ID. *Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery (DAWAK)* (pp. 125-137). Springer Verlag.

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *International Journal of Data Mining and Knowledge Discovery, Kluwer Academic Publishers*, *8*(1), 53–87. doi:10.1023/B:DAMI.0000005258.31418.83

Kum, H.-C., Chang, H. J., & Wang, W. (2006). Sequential pattern mining in multi-databases via multiple alignment. *Data Mining and Knowledge Discovery*, *12*(2), 151–180. doi:10.1007/s10618-005-0017-3

Lin, Y., Hu, X., Li, X., & Wu, X. (2013). Mining stable patterns in multiple correlated databases. *Decision Support Systems*, *56*, 202–210. doi:10.1016/j.dss.2013.06.003

Liu, H., Lu, H., & Yao, J. (2001). Toward multidatabase mining: Identifying relevant databases. *IEEE Transactions on Knowledge and Data Engineering*, *13*(4), 541–553. doi:10.1109/69.940731

Mehenni, T., & Moussaoui, A. (2012). Data mining from multiple heterogeneous relational databases using decision tree classification. *Pattern Recognition Letters*, *33*(13), 1768–1775. doi:10.1016/j.patrec.2012.05.014

Nanopoulos, A., & Manolopoulos, Y. (2000). Finding generalized path patterns for Web log data mining. *Data & Knowledge Engineering*, *37*(3), 243–266. doi:10.1016/S0169-023X(01)00008-8

Park, J. S., Chen, M. S., & Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. *Proceeding of the 1995 ACM-SIGMOD International Conference on Management of Data (SIGMOD'95)* (pp. 175-186). ACM. doi:10.1145/223784.223813

Pei, J., Han, H., Pinto, H., Chen, Q., & Dayal, U. (2004). Prefix span: Mining sequential patterns efficiently by prefix-projected pattern growth. *IEEE Transactions on Knowledge and Data Engineering*, *16*(1), 1424–1440.

Pei, J., Han, J., Mortazavi-asl, B., & Zhu, H. (2000). Mining access patterns efficiently from Web logs. *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00)* (pp. 396-407). doi:10.1007/3-540-45571-X_47

Peng, W. C., & Liao, Z. X. (2009). Mining sequential patterns across multiple sequence databases. *Data & Knowledge Engineering*, *68*(10), 1014–1033. doi:10.1016/j.datak.2009.04.009

Savasere, A., Omiecinski, E., & Navathe, S. (1995). An efficient algorithm for mining association rules in large databases. *Proceeding of the 1995 International Conference on Very Large Data Bases (VLDB'95)* (pp. 432-443). San Francisco, CA: Morgan Kaufmann.

Srikant, R., & Agrawal, R. (1995). Mining generalized association rules. *Proceedings of the 21st International Conference on Very Large Databases (VLDB)* (pp. 407-419).

Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. *Proceedings of the 5th International Conference Extending Database Technology* (pp. 3-17). doi:10.1007/BFb0014140

Taniar, D., Clement, H., Leung, C., Rahayu, W., & Goel, S. (2008). *High performance parallel database processing and grid databases*. Hoboken, NJ: John Wiley & Sons. doi:10.1002/9780470391365

Zhang, S., Wu, X., & Zhang, C. (2003). Mining multi-database mining. *IEEE Computational Intelligence Bulletin*, *2*(1), 5–13.

## KEY TERMS AND DEFINITIONS

**Candidate Generation**: is the process of listing all possible combinations of itemsets/sequences of length k from frequent itemsets/sequences of length k-1.
**Complex Queries**: queries derived from patterns from more than one related table or database.
**Foreign Key**: is an attribute that appears in two tables and connects data in the tables.
**Frequent Patterns**: are the set of candidate itemsets/sequences which have occurred in the database more or equal times than the given minimum support count.
**Frequent Itemsets**: are frequent patterns that consist of only itemsets which have occurred in the database more or equal times than the given minimum support count.
**Frequent Sequences**: are frequent patterns that consist of only sequences which have occurred in the database more or equal times than the given minimum support count.
**Multiple Databases**: are composed of more than one itemset or sequence database tables.
**Sequence Database**: is a database where the value of each tuple or row of data is a sequence of values with some implied historical time order of occurrence.
**Transaction Id**: is a common foreign key attribute that links more than one database table data.

*Christie I. Ezeife received her M.Sc in Computer Science from Simon Fraser University, Canada in 1988, a Ph.D in Computer Science from the University of Manitoba, Canada in 1995 following a First class B.Sc. Honors degree in Computer Science in 1982. She has held academic positions in a number of universities including her current University of Windsor where she has been since 1996. She has been a Full Professor of Computer Science at the University of Windsor, Canada since 2009. She has graduated over 40 PhD and MSc students. Her research interests focus on exploring efficient techniques for physical design, mining, recommendation and application of database systems including data warehouses, object-oriented databases and web data. She has authored several technical publications including over 18 comprehensive journal articles in journals such as ACM Computing Surveys, Kluwer's International journals of Distributed and Parallel Databases, Springer's International Journal of Data Mining and Knowledge Discovery, Elsevier's journal of Data and Knowledge Engineering and IGI Global's International Journal of Data Warehousing and Mining, as well as two books on Problem Solving and Programs with C by Thomson Learning Publishers, which had been successfully used for teaching hundreds of first year Computer Science students for many years.*

*Vignesh Aravindan completed his MSc in Computers Science from the University of Windsor, Canada, in December 2016 under the supervision of Dr. Christie Ezeife. Prior to joining Windsor, he completed a BSc in Computer Science in 2015, from Anna University, Chennai, India. He is currently working as a Big data developer in RBC. He was also a full stack developer in a startup that is responsible for creating the updated MTO drivetest (web/mobile/Internal System) applications.*

*Ritu Chaturvedi completed her Ph.D. in May 2016 from the school of Computer Science, University of Windsor. Ever since she graduated, she has held academic positions in reputable Canadian Universities such as University of Toronto and University of Guelph, where she currently teaches as an Assistant Professor. Prior to 2012, before she began her tenure as a Ph.D student, she held various teaching-focused positions in the School of Computer Science, University of Windsor, from 2001 - 2011. Ritu's research interests are in Data Mining and Predictive modeling, especially in educational data mining that caters to web-based tutoring systems such as Intelligent Tutoring Systems. She did her master's in computer applications in 1988 from NIT, Rourkela, India. Soon after, Ritu started teaching in Banaras University (BHU), India as a Lecturer in Computer Science and was employed with them until 1998, when she moved to Canada. She completed her B.Sc. in Physics Honors from Uktal University, India in 1985.*