HOKKAIDO UNIVERSITY

Title	Frequentness-Transition Queries for Distinctive Pattern Mining from Time-Segmented Databases
Author(s)	Minato, Shin-ichi; Uno, Takeaki
Citation	Proceedings of the 10th SIAM International Conference on Data Mining (SDM2010) p. 339-349
Issue Date	2010-04
Doc URL	http://hdl.handle.net/2115/47334
Rights	Copyright ©2010 Society for Industrial and Applied Mathematics
Type	proceedings
Note	2010 SIAM International Conference on Data Mining (SDM2010). April 29 - May 1, 2010. Renaissance Columbus Downtown Hotel. Columbus, Ohio
File Information	dm10_030_minatos-2.pdf

Instructions for use

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

Frequentness-Transition Queries for Distinctive Pattern Mining from Time-Segmented Databases

Shin-ichi Minato [*] Takeaki Uno [†]

Abstract

We propose a new data mining method called frequentness-transitional pattern mining for finding patterns with interesting sequential behavior specified by a user's query. For a series of databases, we introduce the frequentness-sequence of a pattern that is a sequence of the two symbols 'H' and 'L,' which represent the frequency or infrequency in each segment of a database, respectively. The problem is finding patterns whose frequentness-sequences satisfy the query. The goal of this research is to develop an efficient algorithm and its implementation that accepts various models and that can be widely used in practice with large-scale data. Thus, we chose an itemset as a pattern, and regular expression for the query language to accept various models. To cope with the unavoidably large number of candidate patterns, we use Zero-suppressed Binary Decision Diagrams (ZDDs or ZBDDs) to store and operate a large number of candidate itemsets in a short time. Our algorithm performed quite well in our computational experiments, such that it is competitive with the standard itemset mining algorithms that can be used only to find frequent itemsets. To the best of our knowledge, this is the first study on detecting distinctive itemsets of user-specific models of sequential behaviors.

1 Introduction

Discovering useful knowledge from large-scale databases has attracted considerable attention during the last decade. *Frequent pattern mining* is one of the fundamental problems for data mining and knowledge discovery. The task is to find all frequent patterns included in at least σ records of the database where σ is the user specified threshold. Since the pioneering work by Agrawal *et al.* [1], various algorithms have been proposed to solve the *frequent itemset mining problem* (cf., [4, 15]). Recently, Minato et al. [9] proposed a fast algorithm "LCM over ZDDs" for generating very large-scale frequent itemsets using *Zero-suppressed BDDs (ZDDs)* [8], a compact graph-based data structure. Their method is based on *LCM algorithm* [12], one of the most efficient state-of-the-art techniques for itemset mining, and also based on the ZDD-based data structure, which directly generates compact output data structures on the main memory, to be efficiently post-processed by using algebraic operations.

We apply these efficient itemset mining techniques to sequential databases. There have also been many studies on finding patterns from sequential databases. One of these approaches is known as *sequence mining* or *episode mining*, to find patterns of sequential structures such as sequences, sequences with gap constraints, and directed acyclic graphs [10, 11, 14]. Another approach is finding combinatorial (non-sequential) patterns whose occurrence has an interesting sequential property. We considered this kind of problem. In practice, we often want to know such patterns. For example, if we obtain combinations of keywords, which were not frequently used in Web searches one month ago, but are frequently used now, then perhaps we can find some recent hot topics. Naturally, we understand that this kind of analysis is quite important in many areas such as marketing, web analysis, IT security, financial engineering, and natural science experimentation.

Emerging pattern mining [3] is an early work dealing with such problems. For two databases D_1 and D_2, a pattern P is called an *emerging pattern* if the ratio of P's frequencies in the two databases is greater than a given threshold value θ (i.e. $frq_{D_2}(P)/frq_{D_1}(P) \geq \theta$). This method can be used to capture the distinctive differences between the two databases, or between two different periods of one sequential database. There are many extensions or variations of emerging pattern mining. A recent proposal is *Transitional pattern mining* [13] for time-stamped sequential databases. A pattern P is transitional if there is a segmentation of the database into former and latter such that P's frequency in the former is considerably larger or considerably smaller than that of the latter. Transitional patterns can capture the time point (they call *milestone*) of the frequency changes, and do not need a given segmentation, while

[*]Graduate School of Information Science and Technology, Hokkaido University, Sapporo, 060-0814 Japan. minato@ist.hokudai.ac.jp, and ERATO MINATO Discrete Structure Manipulation System Project, Japan Science and Technology Agency.

[†]National Institute of Informatics, Tokyo 101–8430, Japan. uno@nii.ac.jp

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

the emerging patterns need a good segmentation before mining.

Those differential mining methods are aiming to find distinctive behaviors of itemsets that cannot be detected with ordinary frequent itemset mining, so they have to explore many more candidates with lower frequencies. In addition, the useful *Apriori property* no longer holds, and we cannot prune the search space. Thus, the computation cost is a serious problem in this kind of pattern mining. Recently, Loekit and Bailey proposed using ZDDs for emerging pattern mining to accelerate computation [7]. We expect that the ZDD-based post-processing approach has a large potential for solving these problems.

We propose a pattern mining algorithm which allows users' queries to specify the interesting frequency changes in a sequential database. This idea enables us to handle not only the difference in the two time periods but also more sophisticated models of sequential behavior. If we implement this method in a straightforward way, the computational complexity will become much higher than existing mining problems. To cope with this, we propose a new method, as shown below.

- Digitizing the time frame of sequential databases. We consider a *time-segmented database*, which is a sequence of non-sequential databases of the same type. For example, the sequence of POS database of a supermarket for each day can be considered as a time-segmented database.

- Quantization of the pattern frequencies. For a time-segmented database $\mathcal{D} = (D_1, \ldots, D_T)$, we consider the *frequentness-sequence* of a pattern, which is a sequence of length T, consisting of two symbols 'H' and 'L,' where 'H' ('L') on the i-th position means the pattern is frequent (infrequent) in D_i. (for example, "L...LH...H")

- Using ZDDs for post-processing a large number of patterns. LCM over ZDDs can efficiently extract a large number of frequent itemsets and directly generate compact representations of all solutions using ZDDs. After digitization and quantization of the database model, we can use this state-of-the-art technique.

- Using regular expression for the query language. We call this *frequentness-transition query*, which is a regular expression of 'H' and 'L.' For example, a query "LL*HH*" represents a set of sequences such that "the pattern is not frequent at the beginning, but at some point it becomes frequent and remains so until the end." Many users easily understand regular expressions, and it has a good trade-off between expressive power and complexity.

We call a pattern whose frequentness-sequence belongs to the model of the frequentness-transition query a *frequentness-transitional pattern.* We propose an efficient algorithm for solving the frequentness-transitional pattern mining problem.

This paper is organized as follows. In Section 2, we start with preliminaries and explain straightforward algorithms for understanding the difficulty of the problem. Section 3 describes our algorithm for solving the problem, and Section 4 describes the ZDD and LCM over ZDDs for understanding how to reduce the computational cost. Experimental results are shown in Section 5, followed by our conclusions.

2 Preliminaries

Let $\mathcal{E} = \{1, 2, \ldots, n\}$ be the set of *items.* A *transaction database* on $\mathcal{E}$ is a multiset $D = \{T_1, T_2, \ldots, T_m\}$ where each T_i is included in $\mathcal{E}$. Each T_i is called a *transaction* (or *tuple*). We denote the sum of all transaction sizes in D as $||D||$ that is, the size of database D. A set $P \subseteq \mathcal{E}$ is called an *itemset.*

For itemset P, a transaction including P is an *occurrence* of P. The *denotation* of P, which is denoted by $Occ(P)$, is the set of the occurrences of P. $|Occ(P)|$ is the *frequency* of P and is denoted as $frq(P)$. For a given constant ρ, called a *minimum support ratio*, itemset P is *frequent* if $frq(P) \geq m \times \rho$. Note that usually the threshold value is called minimum support and is given by the absolute number of transactions; however, to adopt the difference in the sizes of the databases in a time-segmented database, we use the ratio for the threshold. A frequent itemset not included in another frequent itemset is called *maximal.* An itemset not included in another itemset with the same frequency is called *closed.*

For a given ρ, the set of frequent itemsets in database D is denoted as $FI(D)$. The problem of frequent itemset mining is to enumerate all frequent itemsets for given database D and a minimum support ratio ρ. In other words, the problem is to generate a representation (such as a list of itemsets) of $FI(D)$.

A *time-segmented database* is a sequence of the same type of databases. We may consider any type of databases such as transaction, text, and graph, but hereafter, a time-segmented database $\mathcal{D}$ is a sequence of transaction databases, i.e., $\mathcal{D} = \{D_1, \ldots, D_T\}$ such that each D_i is a transaction database. Each D_i is called a *segment database.* The size of $\mathcal{D}$, denoted by $||\mathcal{D}||$ is the sum of the database sizes composing $\mathcal{D}$, i.e., $||\mathcal{D}|| = \sum_{i=1}^{T} ||D_i||$. For an itemset P, *frequentness-*

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

sequence $fq(P)$ for time-segmented database $\mathcal{D}$ is a sequence composed of 'H' and 'L' of length T such that the i-th position of the sequence is 'H' if and only if P is frequent in D_i. $FI(\mathcal{D})$ denotes the set of itemsets which are frequent in at least one $D_i \in \mathcal{D}$.

Let $\Sigma = \{a_1, a_2, \dots, a_k\}$ be a finite alphabet; then the class of *regular expressions over* Σ is defined recursively as follows:

1. Any $a_1, a_2, \dots, a_k$ alone is a regular expression, as are the null sequence λ and empty set ϕ.
2. If P_1 and P_2 are regular expressions, then so is their *concatenation* P_1P_2 and their *union* $P_1 \cup P_2$. If P is a regular expression, then so is its *closure* P^*.

$A = [\Sigma, Q, \delta, q_0, F]$ is said to be a *non-deterministic finite automaton (NFA)* if

1. Σ is an alphabet,
2. Q is a finite non-empty set, the set of states,
3. $\delta : Q \times \Sigma \to 2^Q$, the state transition relation,
4. $q_0 \in Q$, the initial state, and
5. $F \subseteq Q$, the set of final states.

A sequence $S \in \Sigma^*$ is said to be accepted by an NFA $A = [\Sigma, Q, \delta, q_0, F]$ if and only if $\delta^*(q_0, p) \cap F \neq \phi$, where $\delta^* : Q \times \Sigma^* \to 2^Q$ is defined as follows:

1. $\delta^*(q, \lambda) = \{q\}$ for any $q \in Q$ and
2. $\delta^*(q, aS) = \bigcup_{q' \in \delta(q,a)} \delta^*(q', S)$ for any $q \in Q$, $a \in \Sigma$, and $S \in \Sigma^*$.

A *state transition graph* is a directed graph for visualizing an NFA. Its vertex set is the set of states $q_0, q_1, \dots$, and a directed edges (q, q') with label a means $q' \in \delta(q, a)$ in the state transition relation.

It is a well-known theorem that any regular expression can be transformed into a state transition graph of an NFA accepting all sequences generated by the regular expression. The size of the state transition graph is linear to the size of the regular expression.

A *frequentness-transition query* is a regular expression which describes a set of sequences composed of 'H' and 'L'. For a frequentness-transition query X, a finite automaton accepting the sequences belonging to the set given by X is denoted as $A(X)$. We call an itemset whose frequentness-sequence is in X a *frequentness-transitional itemset.*

Examples of frequentness-sequences are "L...LH...H", "L...LHHHL...L", and "H...HL...LH...H", where 'L' means infrequent and 'H' means frequent. The regular expressions representing the sequences are "LL*HH*", "L*HHHL*", and "HH*LL*HH*". Using regular expression also enables us to model periodical patterns of being frequent, such as "frequent only one day a week".

We address the following problem.

Frequentness-Transitional Itemset Mining Problem:
Input: a time-segmented transaction database $\mathcal{D}$, a minimum support ratio ρ, and a frequentness-transition query X
Output: all itemsets P whose frequentness-sequences belong to the model given by X.

The frequentness-sequences and frequentness-transitional patterns can also be defined in the same way for other kinds of patterns such as sequences, graphs, and geometric objects. Thus, this problem can be easily considered in other kinds of pattern mining applications.

In this paper, we only consider the two symbols 'H' and 'L' for frequentness-sequences and frequentness-transition queries. However, we can easily extend the model using the three symbols 'H,' 'M,' and 'L' to represent a ternary quantization of high, middle, and low, respectively. More multi-valued quantization will be possible.

2.1 Straightforward Algorithms

To understand the difficulty of this problem and non-triviality of the algorithm, this subsection discusses straightforward algorithms. The simplest way to solve the problem is to compute the frequentness-sequences of all itemsets and input the obtained sequences to the finite automaton $A(X)$. It takes, roughly speaking, $O(2^{|\mathcal{E}|} \times ||\mathcal{D}||)$ time, where $2^{|\mathcal{E}|}$ is the number of itemsets and $O(||\mathcal{D}||)$ is the time to compute the frequentness-sequence of an itemset. Note that the evaluation of a frequentness-sequence using the finite automaton $A(X)$ can be done in $O(T)$ time.

With this method, the frequentness-sequence "$L \dots L$" should not be in the model of X; otherwise, many itemsets will be output. Thus, we can restrict the itemsets to $FI(\mathcal{D})$. Let $T(D_i)$ be the time to enumerate all frequent itemsets in D_i. Then, the computation time will be $O(\sum_{i=1}^{T} T(D_i) + |\bigcup_{i=1}^{T} FI(D_i)| \times ||\mathcal{D}||)$.

This can be further reduced by computing all the frequentness-sequences in a breadth-first manner. First, we compute $FI(\mathcal{D})$. Then, we compute $FI(D_1)$, and set the first position of the frequentness-sequence of each itemset according to whether the itemset is included in $FI(D_1)$. Iteratively, we compute $FI(D_i)$ and determine the i-th position of each frequentness-sequence. In

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

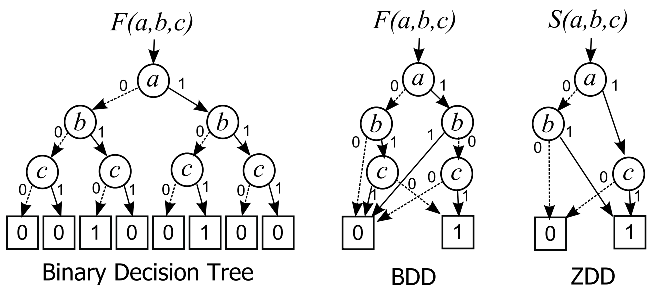

Figure 1: Binary Decision Tree, BDD and ZDD

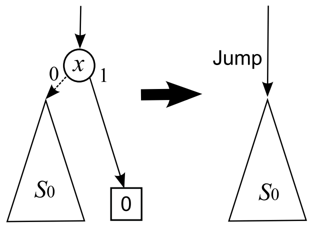

Figure 3: ZDD reduction rules

a	b	c	F	$\rightarrow S$
0	0	0	0	
0	0	1	0	
0	1	0	1	$\rightarrow b$
0	1	1	0	
1	0	0	0	
1	0	1	1	$\rightarrow ac$
1	1	0	0	
1	1	1	0	

As a Boolean function:
$F(a,b,c) = a\bar{b}c \vee \bar{a}b\bar{c}$

As a family of itemsets:
$S(a,b,c) = \{ac,\ b\}$

Figure 2: Correspondence of Boolean functions and sets of combinations

this way, we can compute all frequentness-sequences in $O(\sum_{i=1}^{T} T(D_i))$ time and $O(T \times |FI(\mathcal{D}|)$ memory. This is possibly a limit of straightforward algorithms derived using frequent itemset mining algorithms. However, since both $\sum_{i=1}^{T} T(D_i)$ and $|FI(\mathcal{D})|$ can be quite large in practice, this straightforward algorithm may not practically work on a large amount of data, or minimum support ratio involving a large number of frequent itemsets.

3 Zero-suppressed Binary Decision Diagrams (ZDDs)

A *Binary Decision Diagram (BDD)* is a graph representation for a Boolean function. An Example is shown in Fig. 1 for $F(a,b,c) = a\bar{b}c \vee \bar{a}b\bar{c}$. Given a variable ordering (in our example a, b, c), one can use Bryant's algorithm [2] to construct the BDD for any given Boolean function. For many Boolean functions appearing in practice this algorithm is quite efficient and the resulting BDDs are much more efficient representations than binary decision trees.

BDDs were originally invented to represent Boolean functions, but we can also map a family of itemsets into Boolean space of n variables, where n is the cardinality of $\mathcal{E}$ (see Fig. 2). Therefore, one could also use BDDs to represent families of itemsets. However, one can even obtain a more efficient representation using *Zero-suppressed* BDDs (ZDDs or ZBDDs) [8], especially if the itemsets are small compared to n.

If there are many similar itemsets then the subgraphs are shared resulting in a smaller representation. In addition, ZDDs have a special type of node deletion rule. As shown in Fig. 3, All nodes whose 1-edge directly points to the 0-terminal node are deleted. Because of this, the nodes of items that do not appear in any itemset are automatically deleted as shown in Fig. 1. This ZDD reduction rule is extremely effective if we handle a family of sparse itemsets. If the average appearance ratio of each item is 1%, ZDDs are possibly more compact than ordinary BDDs, up to 100 times.

ZDD representation has another good property in which each path from the root node to the 1-terminal node corresponds to each itemset in the family. Namely, the number of such paths in the ZDD exactly equals the cardinality of the family. This property indicates that, even if there are no equivalent nodes to be shared, the ZDD structure explicitly stores all itemsets as well as using an explicit linear linked list data structure. In other words, (the order of) ZDD size never exceeds the explicit representation. If more nodes are shared, the ZDD is more compact than the linear list.

Figure 4 summarizes the primitive operations of the ZDDs. In these operations, "$\emptyset$," "$\mathbf{1}$," and S.top can be obtained in a constant time. S.offset(k), S.onset(k), and S.change(k) operations require a constant time if item k is at the root node of S; otherwise, they consume linear time for the number of ZDD nodes located at a higher position than item k. The union, intersection, and difference operations can be performed in almost linear time to the size of the ZDDs. S.count is also linear to the ZDD size and does not depend on the cardinality.

Recently, Knuth [6] presented a new fascicle of his famous book series. This new fascicle has a section entirely devoted for BDDs with total 140 pages including 236 exercises, and ZDDs are also discussed minutely in 30 pages including 70 exercises. He re-arranged a set of primitive ZDD operations and named it "Family Algebra." His own BDD/ZDD package is available on his home page.

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

"$\emptyset$"	Returns empty family. (0-terminal node)
"**1**"	Returns singleton family of null-itemset. (1-terminal node)
S.top	Returns item-ID at root node of S.
S.offset(k)	Sub-family of itemsets not including item k.
S.onset(k)	Gets $S \setminus S$.offset(k) and then deletes item k from each itemset.
S.change(k)	Inverts existence of item k (add or delete) on each itemset.
$S_1 \cup S_2$	Returns union of the two families.
$S_1 \cap S_2$	Returns intersection of the two families.
$S_1 \setminus S_2$	Returns difference of the two families. (in S_1 but not in S_2.)
S.count	Counts cardinality of S.

Figure 4: Primitive ZDD operations

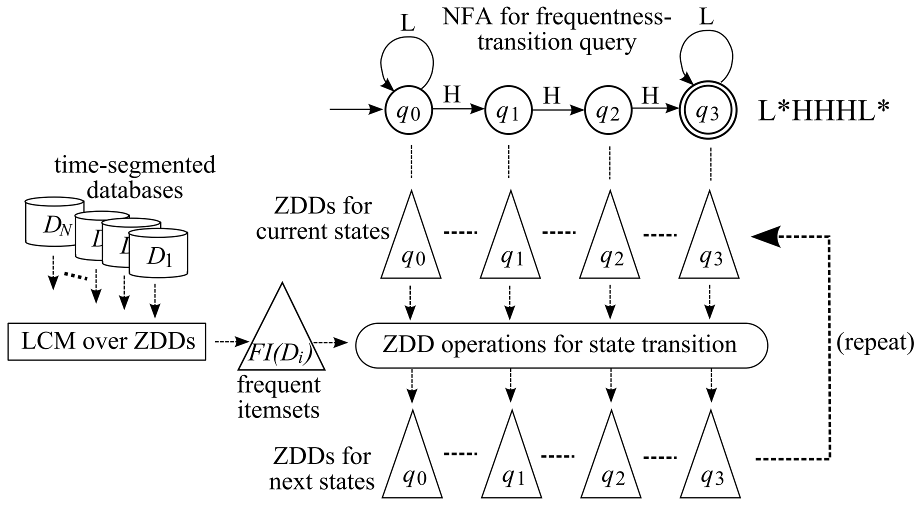

Figure 5: ZDD-based symbolic processing for finite automata

4 Frequentness-Transitional Itemset Mining from Time-Segmented Databases

The basic scheme of our method is to evaluate all the frequentness-sequences using the NFA of the given query. Suppose that we have an NFA A and are going to evaluate frequentness-sequences in the set S of frequentness-sequences using this NFA. We first place all frequentness-sequences in S on the start state q_0. We call an edge of an NFA *H-edge (L-edge)* if it is labeled 'H' ('L').

As the first iteration, we move the sequences whose first position is 'H' along the out-going H-edges, and the sequences whose first position is 'L' along the out-going L-edges. If there are k out-going edges, a sequence on q_0 will be duplicated and moved to k states. If there is no out-going edge, that is $k = 0$, we simply delete the sequences whose first position is 'H'. We do the same for sequences having 'L' on the first position. Similarly, in the i-th iteration, for each state q of A, we move the sequences on q such that

- sequences whose i-th position is 'H' are moved via out-going H-edges , and
- sequences whose i-th position is 'L' are moved via out-going L-edges.

After the process of T-th iteration, the sequences remaining on the final states belonging to the model given by X, that is, the itemsets with these frequentness-sequences are the frequentness-transitional patterns.

In fact, this computation can be done by inputting the sets of itemsets instead of frequentness sequences. At the beginning, we place all the itemsets in $FI(\mathcal{D})$, whose frequentness sequences will be evaluated, on the start state q_0. Then, as the first iteration, we move the itemsets that are frequent in D_1 along the out-going H-edges, and move those that are infrequent along the out-going L-edges. Let $I_i(q)$ be the set of itemsets on the state q after the i-th iteration, and $I_0(q)$ is an empty set for $q \neq q_0$ and is the set of all itemsets for $q = q_0$. In the i-th iteration, for each state q of A, we move

- itemsets in $I_{i-1}(q)$ being frequent in D_i along out-going H-edges, and

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

```
**ALGORITHM Mine-FrqTransPattern** ($\mathcal{D}$, $\sigma$, $A$)
**Input:** time-segmented database $\mathcal{D} = \{D_1, \ldots, D_T\}$, minimum support threshold $\sigma$,
frequentness-transition query (FNA) $A = [\Sigma, Q, \delta, q_0, F]$
**Output:** all frequentness-transitional patterns

1. determine the item ordering used by ZDDs
2. for each state $q \in Q, q \neq q_0$, assign an empty ZDD $Z(q)$
3. assign a ZDD representing all itemsets to $q_0$
4. for $i := 1$ to $T$
5.   compute a ZDD representing $FI(D_i)$
6.   for each state $q \in Q$, set $Z(q)$ to a ZDD representing $I_i(q)$
8. end for
9. output all itemsets in the ZDD on each accepted state
```

Figure 6: Algorithm for finding frequentness-transitional patterns

- itemsets in $I_{i-1}(q)$ being infrequent in D_i along out-going L-edges.

Then, at the end of the T-th iteration, the itemsets remaining on the final states are the frequentness-transitional itemsets. For a state q, $\delta^{-1}(q, H)$ denotes the set of states q' with an edge from q' to q labeled 'H,' $\delta^{-1}(q, L)$ denotes that for 'L.' In the term of set operations, the above process can be written as

$$I_i(q) = \bigcup_{q' \in \delta^{-1}(q,H)} (I_{i-1}(q') \cap FI(D_i))$$
$$\bigcup_{q' \in \delta^{-1}(q,L)} (I_{i-1}(q') \setminus FI(D_i)). \qquad (1)$$

LEMMA 4.1. *An itemset P is in $I_i(q)$ if and only if the frequentness-sequence of P is on q after the i-th transition.*

Proof. Since $I_0(q_0)$ is the set of all itemsets and $I_0(q), q \neq q_0$ is the emptyset, $I_1(q)$ satisfies the statement of the lemma. Suppose that $I_i(q)$ satisfies the statement of the lemma for any $i < k$. Then, from Eq. (1), $I_k(q)$ is the set of itemsets whose frequentness-sequences are on q after the i-th transition. Thus, by induction, the statement holds. ∎

We can see that the computation fits to ZDD quite well. We represent each $I_i(q)$ using a ZDD. The operation in each iteration is composed of computing the intersection, the difference, and the union of set families; thus, the computation time in an iteration is reduced by ZDD operations. In our experiments, we successfully represented quite many, up to 10^{43}, frequent itemsets by using a small ZDD, thus we could compute frequentness-transitional itemsets in a short time while saying that straightforward approaches cannot compute them. An intuitive image of the process with ZDDs is illustrated in Fig. 5.

4.1 Example of Execution of Algorithm

Figure 7 shows that the frequentness-transition query is "L*HHL*", and the time-segmented database $\mathcal{D}$ is composed of five transaction databases $D_1, \ldots, D_5$. The frequent itemsets are shown, by writing only maximal frequent itemsets to the conciseness. D_3 has maximal frequent itemsets $\{1, 2\}$ and $\{2, 3, 4\}$, thus the frequent itemsets of D_3 are $\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 3, 4\}$ and $\{2, 3, 4\}$.

First, we construct an NFA to accept the same sequence set as the given frequentness-transitional query. The state transition graph is shown in this figure. The NFA has three states, q_0, q_1 and q_2 where q_0 is the start state and q_2 is the final state, and the transition function is shown as 1-edges of (q_0, q_1) and (q_1, q_2) and 0-edges of (q_0, q_0) and (q_2, q_2).

Then, we place $FI(\mathcal{D})$ at q_0. For the first iteration, we compute $FI(\mathcal{D}) \cap FI(D_1)$ and put it on q_1, and compute $FI(\mathcal{D}) \setminus FI(D_1)$ and put it on q_0. In the second iteration, we put $FI(\mathcal{D}) \cap FI(D_1) \cap FI(D_2)$ on q_2, $(FI(\mathcal{D}) \setminus FI(D_1)) \cap FI(D_2)$ on q_1, and $(FI(\mathcal{D}) \setminus FI(D_1)) \setminus FI(D_2)$ on q_0. The itemsets in $(FI(\mathcal{D}) \cap FI(D_1)) \setminus FI(D_2)$ disappear from the NFA. Similarly, in the i-th iteration of the algorithm, we

- take the set I on q_0, put $I \cap FI(D_i)$ on q_1, and put $I \setminus FI(D_i)$ on q_0,
- remove the set I on q_1, put $I \cap FI(D_i)$ on q_2 (itemsets in $I \setminus FI(D_i)$ disappear from the NFA), and

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

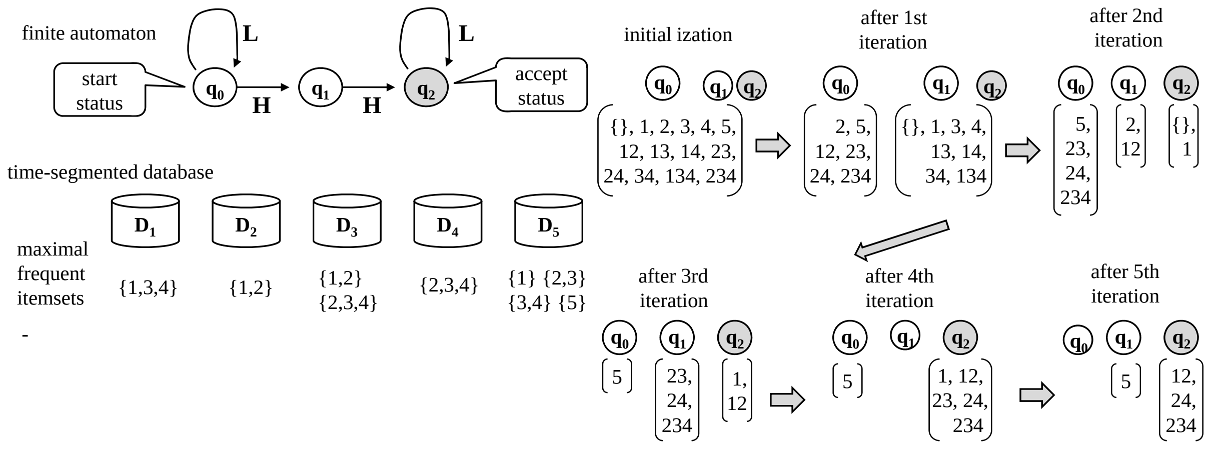

Figure 7: Example of finite automaton and frequent itemsets in time-segmented database; transition of itemsets on each state is shown below

- remove the set I on q_1, put $I \cap FI(D_i)$ on q_2 (itemsets in $I \setminus FI(D_i)$ disappear from the NFA)

The itemsets on each state is illustrated in Fig. 7 for each end of the iteration. These set family operations are done via ZDD arithmetic operations, and the ZDDs representing $FI(D_i)$'s are constructed by LCM over ZDDs. Note that in the i-th iteration, we do not take the itemsets on q_j that have moved from the other node in the same (i-th) iteration. Finally, at the end of the 5th iteration, we obtain $\{1,2\}$, $\{2,4\}$ and $\{2,3,4\}$ on q_2.

4.2 Further Improvement The algorithm we state above is a kind of 2-pass algorithm; first compute $FI(\mathcal{D})$ then evaluate using an NFA. This subsection shows the procedure for executing in 1-pass.

For 1-pass execution, we do not have $FI(\mathcal{D})$ at the initialization, and a trivial solution is to put the set of all itemsets $2^{\mathcal{E}}$ on the start state. The difficulty is that the set of itemset on each state is no longer sparse, thus the ZDDs will be quite large. To cope with this problem, we use a mark to represent the complement set.

After the initialization, the set of all the itemsets, $2^{\mathcal{E}}$, is on the start state. In the first iteration, we compute $FI(D_1)$, then put $FI(D_1)$ to some node, and $2^{\mathcal{E}} \setminus FI(D_1)$ to some node. Instead of computing the ZDD for $2^{\mathcal{E}} \setminus FI(D_1) = \overline{FI(D_1)}$, we put the ZDD representing $FI(D_1)$ and the mark of "complement". In the second iteration, we compute $FI(D_2)$ and compute $\overline{FI(D_1)} \cap FI(D_2)$ and $\overline{FI(D_1)} \setminus FI(D_2)$. They, in fact, hold

$$\overline{FI(D_1)} \cap FI(D_2) = FI(D_2) \setminus FI(D_1), \text{and}$$

$$\overline{FI(D_1)} \setminus FI(D_2) = \overline{FI(D_1) \cup FI(D_2)}.$$

Generally, for a set I of itemsets on a state with a "complement" mark,

$$\overline{I} \cap FI(D_2) = FI(D_2) \setminus I, \text{and}$$

$$\overline{I} \setminus FI(D_2) = \overline{I \cup FI(D_2)}.$$

This is not a heavy task since the set families we deal with are still sparse, thus we do not lose efficiency by omitting the computation of $FI(\mathcal{D})$.

In summary, an iteration involves a construction of a ZDD representing $FI(D_i)$ and performs $O(A)$ ZDD operations in an iteration where A is the number of edges in the NFA. Although in the worst case, the number of itemsets on each state can increase up to $|FI(\mathcal{D})|$ as the iterations proceed, in practice many itemsets will disappear; thereby, the ZDDs on the states are expected to be small on average. We can also observe this in the example. Therefore, roughly speaking, the computation time of the algorithm is $O(\sum_{i=1}^{T} |Z_i|)$ where $|Z_i|$ is the size of a ZDD representing $FI(D_i)$.

Theorem 4.1. *For a given time-segmented database composed of T transaction databases, a minimum support threshold σ, and a frequentness-transitional query A, a ZDD representing all frequentness-transitional itemsets can be obtained in $O(|A|FZT)$ time, where $|A|$ is the number of edges in the transition graph of A, and F (resp., Z) is the maximum size of ZDD to represent $FI(D_i)$ (resp., $I_i(q)$).*

4.3 LCM over ZDDs for Large-Scale Itemset Mining In this subsection, we briefly review the LCM over ZDDs algorithm proposed by Minato et al. [9] to efficiently generate and store a large number of frequent itemsets on the main memory. The algorithm is obtained by attaching ZDD operations in the inside of the LCM algorithm.

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

```
LCM_Backtrack($P$: itemset)
1. **Output** $P$
2. **For** $e = n$ **to** the maximum item in $P$+1 **step** $-1$ **do**
3.         **If** $P \cup \{e\}$ is frequent **then** LCM_Backtrack($P \cup \{e\}$)

ZDD LCMovZDD($P$: itemset)
1. ZDD $F \leftarrow P$
2. **For** $e = n$ **to** the maximum item in $P$+1 **step** $-1$ **do**
3.     **If** $P \cup \{e\}$ is frequent **then** $F \leftarrow F \cup$LCMovZDD($P \cup \{e\}$)
```

Figure 8: Basic structure of LCM and LCM over ZDDs

Table 1: Comparison of LCM over ZDDs with original LCM

Database name	#Item	#Trans-action	$\|\|D\|\|$ (size of D)	Min. support	#Frequent itemsets	LCM over ZDDs		LCM-count	LCM-dump
						|ZDD|	Time(s)	Time(s)	Time(s)
mushroom	119	8,124	186,852	1,000	123,287	760	0.50	0.49	0.64
				300	5,259,786	4,412	2.25	2.22	9.96
				100	66,076,586	11,584	5.06	4.87	114.21
				50	198,169,866	17,830	8.17	7.86	357.27
BMS-Web View-1	497	59,602	149,639	50	8,192	3,415	0.11	0.11	0.12
				40	48,544	10,755	0.18	0.18	0.22
				36	461,522	28,964	0.49	0.42	0.98
				34	4,849,466	49,377	1.30	1.07	8.58
				32	1,531,980,298	71,574	31.90	29.73	3,843.06
BMS-Web View-2	3,340	77,512	358,278	5	26,946,004	353,091	4.84	3.62	51.28
T10I4D100K	870	100,000	1,010,228	2	19,561,715	3,270,977	9.68	5.09	22.66
chess	75	3,196	118,252	1,000	29,442,849	53,338	197.58	197.10	248.18
connect	129	67,557	2,904,951	40,000	23,981,184	3,067	5.42	5.40	49.21
pumsb	2,113	49,046	3,629,404	32,000	7,733,322	5,443	60.65	60.42	75.29

(2.4GHz Core2Duo E6600 PC, 2 GB memory, SuSE Linux 10, GNU C++)

LCM over ZDDs does not touch the core algorithm of LCM, and just generates a ZDD for the solutions (or set of solutions, such as "$\{a, b\}$ plus any subset of $\{c, d, e\}$") obtained using LCM. Figure 8 shows the basic structure of the LCM. We omit the description of detailed techniques used in LCM for checking the frequency of each itemset. LCM explores all frequent itemsets in a backtracking (or depth-first) manner, and when a frequent itemset is found, the itemset is appended to the output file one by one. On the other hand, LCM over ZDDs constructs a ZDD, which is the union of all the itemsets found in the backtracking search, and finally returns a pointer to the root node of the ZDD. A basic modification is described in Fig. 8.

Recent itemset mining algorithms contain a technique called "equi-support", which is to find a set of frequent itemsets with the same frequency at once. Specifically, equi-support finds two itemsets Y and Z such that each itemset in $S = \{X \cup Z | X \subseteq Y\}$ has the same frequency. It reduces the computation of the frequency of each itemset in the set, thus we can decrease the computation time if Y is not small. Considerable improvements by equi-support are demonstrated with the experiments in FIMI repository [5]. However, if we output the frequent itemsets to file, we generate each itemset in S and output it to the file, spending a long time even if we find S in a short time. On the contrary, a ZDD allows the addition of such an exponential number of itemsets in one step, in precise with $|Y|$ operations, thus it can drastically reduce memory usage and the computation time. LCM over ZDDs usually takes quite a short time compared with usual itemset mining algorithms that output the frequent itemsets to a file.

Minato et al. demonstrated the performance of LCM over ZDDs [9]. The benchmark datasets, listed in Table 1, were chosen from the FIMI2003 repository [5]. |ZDD| represents the number of ZDD nodes representing all frequent itemsets. "LCM-count" is the computation time of LCM only for counting the number of itemsets (i.e., not output itemsets to a file), and "LCM-dump" is the time for outputting all frequent itemsets to a file (using /dev/null), that corresponds to usual itemset mining algorithms. "LCM over ZDDs" shows the time of LCM over ZDDs.

The computation time of LCM over ZDDs is almost the same as LCM-count, which does not output item-

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

Table 2: Number of itemsets and ZDD sizes for each state of NFA

i	$FI(D_i)$		$\overline{q_0}$		q_1		q_2		$q_3(solutions)$	
(seg.)	#Itemset	|ZDD|	#Itemset	|ZDD|	#Itemset	|ZDD|	#Itemset	|ZDD|	#Itemset	|ZDD|
0	–	–	0	0	0	0	0	0	0	0
1	345,095	1,483	345,095	1,483	345,095	1,483	0	0	0	0
2	131,908	447	476,592	1,660	131,417	316	491	314	0	0
3	340	228	476,557	1,688	45	50	23	31	233	172
4	701	374	476,774	1,792	217	190	8	12	39	52
5	1,393	562	477,528	2,005	754	368	59	70	18	29
6	2,230	797	478,867	2,444	1,339	611	109	96	37	53
7	1,668	666	479,695	2,741	828	479	199	150	33	55
8	$(1.49 \cdot 10^{20})$	12,147	$(1.49 \cdot 10^{20})$	15,082	$(1.49 \cdot 10^{20})$	15,862	500	322	186	136
9	328	260	$(1.49 \cdot 10^{20})$	15,096	10	16	29	43	174	150
	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯
55	1,598	737	$(7.14 \cdot 10^{44})$	26,499	361	248	109	82	878	313
56	425	336	$(7.14 \cdot 10^{44})$	26,516	19	25	3	4	877	320
57	443	322	$(7.14 \cdot 10^{44})$	26,560	17	26	0	0	872	318
58	455	342	$(7.14 \cdot 10^{44})$	26,607	57	63	5	8	866	312
59	1,610	723	$(7.14 \cdot 10^{44})$	26,817	676	287	17	18	852	302

sets. We can observe that LCM over ZDDs is much more efficient when large numbers of frequent itemsets are output. The original LCM-dump is known as an output linear time algorithm, but LCM over ZDDs is much faster than the original one, especially when the ZDD size is sub-linear to the number of frequent itemsets.

5 Experimental Results

To evaluate our algorithm, we conducted experiments for the benchmark datasets. For example, "BMS-WebView-1" is known as a set of click streams for an online shopping site. Each transaction shows a set of visited web pages from one customer's consecutive action. This dataset consists of 59,602 transactions without any timing information, but we assume that they are sorted by time, and we partitioned them with 1,000 transactions per segment, i.e., we had 59 segments in total (the last fragment was not used).

We then applied our algorithm to extract the itemsets whose frequentness-sequence belongs to X. Table 2 lists the results of the number of itemsets and ZDD sizes during the symbolic simulation for "BMS-WebView-1" with a minimum support ratio $\rho = 0.4\%$ and the frequentness-transition query $X =$ "L*HHHL*". We observed that 852 itemsets were finally extracted from 713623846352979940529143133451627984798184096 ($\approx .7.14 \cdot 10^{44}$) of itemsets in $FI(\mathcal{D})$. The total computation time was less than five seconds, including the execution of LCM over ZDDs. To check the correctness, we confirmed that the itemset $\{18631, 18643\}$ is frequent only in D_{50}, D_{51} and D_{52}. Similarly, $\{46293, 46285, 46281\}$ is frequent only in D_6, D_7 and D_8. This means that we discovered local events only seen in these time-segments.

Our algorithm can be applied flexibly for various settings. Table 3 shows the performance for different datasets, parameters, and frequentness-transition queries.

The next experiment was done for investigating the scalability of the algorithm. The scalability of the mining process is already known in [9], thus we look only at the scalability concerned with the increase in the number of segment databases. We used the click-stream data "kosarak", which has 990,000 transactions, for this experiment. We segmented the database so that each fragment has 1,500 transactions and executed our algorithm with the same setting as the previous experiment, for 10 to 640 generated databases. The frequentness-transition query was "L*HHHL*" and the minimum support ratio $\rho = 0.2\%$. The results are listed in Table 4. Against the increase in the number of segment databases, the computation time per segment database did not increase, thus we can say our algorithm scales to large-scale databases.

We remark that our algorithm repeats the same procedure for each time-segment, so the computation time is basically linear in T, but it depends on ZDD sizes for representing $FI(D_i)$'s and the itemsets on q_j's. We can apply this algorithm for large T, if we appropriately control the ZDD sizes by the minimum support ratio ρ.

6 Conclusions

We proposed a new mining problem called frequentness-transitional pattern mining by introducing time-segmented databases, which are sequences of databases. We modeled the problem by using a frequentness-transition query, which is a set of sequences composed of symbols meaning "frequent" and "infrequent", in the term of regular expressions. Frequentness-transition queries can represent many kinds of signifi-

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

Table 3: Experimental results for benchmark data

Dataset	ρ	Freq. trans.	#Itemset		Time
(#Segment)	(%)	query X	$FI(\mathcal{D})$	solutions	(sec)
BMS-WebView-1	0.5	L*HHHL*	$7.21 \cdot 10^{16}$	37	0.40
(59 segments)	0.4	L*HHHL*	$7.14 \cdot 10^{44}$	852	4.51
	0.3	L*HHHL*	$1.18 \cdot 10^{46}$	$3.57 \cdot 10^{44}$	42.00
	0.5	HH*LL*HH*	$7.21 \cdot 10^{16}$	7	0.40
	0.4	HH*LL*HH*	$7.14 \cdot 10^{44}$	6	4.41
	0.3	HH*LL*HH*	$1.18 \cdot 10^{46}$	19	42.90
BMS-WebView-2	0.4	L*HHHL*	666,654	300	1.75
(77 segments)	0.3	L*HHHL*	9,236,264	1,493	2.69
	0.2	L*HHHL*	$1.44 \cdot 10^{17}$	38,895	7.04

(2.4GHz Core2Duo PC, 2 GB mem., SuSE 10, GNU C++)

Table 4: Experimental results for scalability

#seg. T	$\lvert FI(\mathcal{D})\rvert$	ZDD size of $FI(\mathcal{D})$	#solution	time (sec)	time per #seg.
10	37,383,478,401,664	12,165	1,253	1.04	0.104
20	147,573,954,513,903,795,412	18,944	104	1.65	0.083
40	147,573,954,513,906,985,334	30,313	186	3.02	0.076
80	147,880,476,351,967,382,435	65,077	63	6.39	0.080
160	182,478,254,595,792,811,136	142,507	10	15.29	0.096
320	1311313142758880369213142447574	255,443	7	38.93	0.122
640	1311313147794829058480599311461	656,965	4	108.94	0.170

cant/periodical changes in natural ways, thus we hope it will help in the mining tasks of change detections in real-world problems. We also proposed an efficient algorithms obtained by combining ZDDs and the LCM by using ZDD-based symbolic processing of finite automata. Our algorithm performs quite well compared to straightforward algorithms and terminates almost at the same time as the usual itemset mining algorithms.

As we mentioned in the introduction, the Apriori property does not hold in this kind of problem, so it is hard to prune the search space. Also, closed/maximal itemset mining techniques are not effective for this problem because the frequentness-transitional itemsets are not always closed/maximal in each segment of databases. On the other hand, ZDDs provide automatic compressed graph representation for a large number of itemsets, and the compressed representation can be processed and analyzed efficiently using various set operations without decompression.

Future research may be to apply this algorithm to real-world problems and to other pattern mining problems, and consider classes of sequence sets that cannot be represented by regular expressions.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data, Vol. 22(2) of SIGMOD Record*, pages 207–216, 1993.

[2] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[3] G. Dong and J. Li. Efficient mining of emerging patterns: discovering trends and differences. In *Proc of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (SIGKDD'99)*, pages 43–52, 1999.

[4] B. Goethals. Survey on frequent pattern mining, 2003. http://www.cs.helsinki.fi/u/goethals/publications/survey.ps.

[5] B. Goethals and M. J. Zaki. Frequent itemset mining dataset repository, 2003. Frequent Itemset Mining Implementations (FIMI'03), http://fimi.cs.helsinki.fi/.

[6] D. E. Knuth. *The Art of Computer Programming: Bitwise Tricks & Techniques; Binary Decision Diagrams*, volume 4, fascicle 1. Addison-Wesley, 2009.

[7] E. Loekit and J. Bailey. Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. In *Proc. The Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2006)*, pages 307–316, 2006.

[8] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of 30th*

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.

ACM/IEEE Design Automation Conference, pages 272–277, 1993.

[9] S. Minato, T. Uno, and H. Arimura. LCM over ZBDDs: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. In *Proc. of 12-th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008), (LNAI 5012, Springer)*, pages 234–246, 5 2008.

[10] L. Parida, I. Rigoutsos, A. Floratos, D. E. Platt, and Y. Gao. Pattern discovery on character sets and real-valued data: linear-bound on irredandant motifs and efficient polynomial time algorithms. In *Proc. SODA'00*, 2000.

[11] N. Pisanti, M. Crochemore, R. Gross, and M. F. Sagot. A basis of tiling motifs for generating repeated patterns and its complexity of higher quorum. In *Proc. MFCS'03*, 2003.

[12] T. Uno, Y. Uchida, T. Asai, and H. Arimura. LCM: an efficient algorithm for enumerating frequent closed item sets. In *Proc. Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, 2003. http://fimi.cs.helsinki.fi/src/.

[13] Q. Wan and A. An. Transitional patterns and their significant milestones. In *Proc of Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 691–696, 2007.

[14] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proc. IEEE ICDE 2004*, pages 79–90, 2004.

[15] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(2):372–390, 2000.

Copyright © by SIAM.
Unauthorized reproduction of this article is prohibited.