Distribution-Based Synthetic Database Generation Techniques for Itemset Mining

Ganesh Ramesh
University of British Columbia
ramesh@cs.ubc.ca

Mohammed J. Zaki *
Rensselaer Polytechnic Institute
zaki@cs.rpi.edu

William A. Maniatty
University at Albany, SUNY
maniatty@cs.albany.edu

Abstract

The resource requirements of frequent pattern mining algorithms depend mainly on the length distribution of the mined patterns in the database. Synthetic databases, which are used to benchmark performance of algorithms, tend to have distributions far different from those observed in real datasets. In this paper we focus on the problem of synthetic database generation and propose algorithms to effectively embed within the database, any given set of maximal pattern collections, and make the following contributions:

1. A database generation technique is presented which takes k *maximal itemset collections as input, and constructs a database which produces these maximal collections as output, when mined at* k *levels of support. To analyze the efficiency of the procedure, upper bounds are provided on the number of transactions output in the generated database.*

2. A compression method is used and extended to reduce the size of the output database. An optimization to the generation procedure is provided which could potentially reduce the number of transactions generated.

3. Preliminary experimental results are presented to demonstrate the feasibility of using the generation technique.

1. Introduction

Frequent and maximal frequent itemset mining are classic data mining problems that are widely used in practice. Informally, given a transaction database where each transaction is a set of items, frequent itemset mining looks for all possible itemsets occurring in at least a certain fraction of the database. This fraction which is commonly called the minimum support threshold is specified by the user. All existing frequent and maximal itemset mining algorithms have exponential worst case complexity as they could potentially output all the possible itemsets. However, the methods perform well in practice: partly due to the statistical distributions of real datasets and partly because they employ heuristics which exploit statistical information from the database to prune the output space.

Itemset mining algorithms are developed with an intention to perform efficiently on real datasets. However, copyright violations and intellectual property rights have limited the availability of real datasets. Hence, the performance analysis of itemset mining methods has heavily relied on benchmark datasets. Specific synthetic datasets such as those generated by IBM's synthetic dataset generator [6], that are widely used for benchmarking, tend to have substantially different distributions from those found in real world datasets. This fact has been recently established in [19]. This makes the quality of benchmarks obtained from such datasets questionable, when the algorithms are indeed intended for use on real datasets. Zheng et. al. [19] and our earlier work [15] also observed that the resource requirements of the mining algorithm relied on the length distribution of the mined patterns.

As a reasonable assumption to disclosure, let us suppose that organizations that are concerned with privacy are willing to disclose some data characteristics (say distributions) without disclosing the actual contents of the data. A natural question arises:

Can we generate synthetic datasets that mimic real datasets with respect to these disclosed characteristics?

For this work, we assume that the organizations are willing to disclose distributions of patterns at various support levels in the database. We focus on the problem of generating a synthetic dataset in which the pattern distributions match the specified distributions. The problem of privacy preservation with respect to disclosure of these distributions is interesting but orthogonal to this work and hence we do not consider it here. Such a generation technique could then be used in a setting where organizations could reveal distributions of patterns without actually revealing the patterns or the transactions in the database, for developing benchmarks for itemset mining algorithms. Specifically, we make the following technical contributions.

*This work was supported in part by NSF CAREER Award IIS-0092978, DOE Career Award DE-FG02-02ER25538, and NSF grants EIA-0103708 and EMT-0432098.

- A database generation technique is presented which takes k maximal itemset collections as input and constructs a database which produces these maximal collections as output when mined at k levels of support. To analyze the efficiency of the procedure, upper bounds are provided on the number of transactions output in the generated database.
- A compression method (earlier defined in [16]) is used and suitably extended to provide a technique for reducing the size of the output database. An optimization to the generation procedure is provided which could potentially reduce the number of transactions generated.
- We provide preliminary experimental results to demonstrate the feasibility of using the generation technique in practice.

1.1. Related Work

A variety of methods have been proposed for frequent and maximal itemset mining [6, 2, 11, 18, 10]. They span from methods which employ various types of search through the lattice space of frequent itemsets to studying these patterns with constraints [14]. Privacy issues in data mining have currently received attention and several issues in this direction have been explored [8, 12]. Theoretical results have been obtained in the general context of relating hypergraph transversal and mining interesting patterns by Gunopulos et al. [5]. They relate these mining problems to the model of exact learning in computational learning theory and provide complexity results and lower bounds. The complexity of mining frequent and maximal frequent itemsets have been studied by Yang in [17]. The author proves that counting the number of distinct maximal frequent itemsets in a database is #P-Complete and concludes that the problem of mining maximal frequent itemsets is NP-Hard. In a seminal work on application of combinatorial results, Geerts et. al. [9] provided a tight upper bound on candidate patterns that are computed by breadth-wise bottom up algorithm like *Apriori* [1]. In a previous study [16], we used combinatorial techniques to formally characterize the feasible frequent and maximal itemset distributions and gave preliminary techniques to embed distributions in a database. Work has also been done by Toon Calders et al. [3] on mining non-derivable itemsets and providing bounds on candidates. The concept of transaction maps/bags was motivated by the notion of bags which exist in multiset terminology and which were also used in [7]. General condensed representations for finding frequent patterns were also studied in [4]. Almost all synthetic datasets are generated using the IBM synthetic dataset generator [6]. This generator uses a Poisson frequent pattern distribution to generate datasets, about a user specified mean. To the best of our knowledge, our technique in [16] is the first in terms of distribution based dataset generation.

2. Preliminaries and Problem Statement

[Itemset, Transaction, Database]: Let $\mathcal{I}$ be a non-empty set of elements. Each $x \in \mathcal{I}$ is called an *item*. A non-empty subset $X \subseteq \mathcal{I}$ is called an *itemset*. The *size* or *length* of an itemset X, denoted by $|X|$, is the number of items in X. An itemset of size k is also called a k-itemset. The power set of $\mathcal{I}$, denoted by $\mathcal{P}(\mathcal{I})$, is the set of all possible itemsets of $\mathcal{I}$. The set of all possible k-itemsets of $\mathcal{I}$ is denoted by $\mathcal{I}^{(k)}$. For $X, Y \in \mathcal{P}(\mathcal{I})$, X is said to *contain* Y, if $Y \subseteq X$. A transaction T, is an itemset with a unique identifier, called the *transaction identifier* or *tid*. A database $\mathsf{DB} = \{T_1, T_2, \ldots, T_N\}$ is a finite, non-empty multi-set of transactions, with size $|\mathsf{DB}| = N$. A transaction T_i is said to contain an itemset X if $X \subseteq T_i$. A transaction T_i is said to be a *maximal transaction* if $\nexists T_j \in \mathsf{DB}$, such that $T_i \subset T_j$.

Assumptions: For the rest of the paper, we assume that items are drawn from the set of positive integers. Even though in the domain of sets, the actual elements are unordered, we implicitly assume an ordering $<$ (less than) on the items.

[Itemset Collections, D-sequence]: An itemset collection is a set of itemsets. More formally, a set $\mathcal{F} \subseteq \mathcal{P}(\mathcal{I})$ (with $\emptyset \notin \mathcal{F}$) is called an *itemset collection* over $\mathcal{I}$. We distinguish two types of k-itemsets that are associated with an itemset collection $\mathcal{F}$. The first type of k-itemsets are those that are members of $\mathcal{F}$. The second type of k-itemsets are those that are contained in members of $\mathcal{F}$. These are formally defined as follows. The k-*collection* of $\mathcal{F}$, denoted $\mathcal{F}_k$, is the collection of all k-itemsets in $\mathcal{F}$, i.e., $\mathcal{F}_k = \mathcal{F} \cap \mathcal{I}^{(k)} = \{X \in \mathcal{I}^{(k)} \mid X \in \mathcal{F}\}$. On the other hand, the *induced* $k-$collection of $\mathcal{F}$, denoted $[\mathcal{F}]_k$, is the set of k-itemsets contained in some element of $\mathcal{F}$, defined as, $[\mathcal{F}]_k = \{X \in \mathcal{I}^{(k)} \mid X \subseteq Y \text{ for some } Y \in \mathcal{F}\}$. Note that $[\mathcal{F}] = \bigcup_k [\mathcal{F}]_k$. Let $f_k = |\mathcal{F}_k|$ denote the size of $\mathcal{F}_k$. Let $l \leq n$ be the length of the longest itemset in $\mathcal{F}$. Then, the length distribution of itemsets in $\mathcal{F}$ given by the sequence $\langle \mathcal{F} \rangle = \langle f_1, f_2, \cdots, f_l \rangle$, is called the *distribution sequence* or the **D-sequence** of $\mathcal{F}$. A *maximal itemset collection* or *maximal collection* is an itemset collection $\mathcal{MF}$ where $(X \in \mathcal{MF}) \Rightarrow \nexists Y \in \mathcal{MF}, \supset X \subset Y$.

[Containment, Cover]: Let C_1 and C_2 be two itemset collections over $\mathcal{I}$. C_1 is said to be *contained* in C_2 (or

C_2 contains C_1) iff $C_1 \subseteq C_2$. C_1 is said to be *properly contained* in C_2 iff $C_1 \subset C_2$. C_1 is said to be *covered by* C_2, denoted as $C_1 \sqsubseteq C_2$ (or C_2 covers C_1), iff $\forall(X \in C_1) \Rightarrow (\exists Y \in C_2 | X \subseteq Y)$. That is, $C_1 \sqsubseteq C_2 \Leftrightarrow C_1 \subseteq [C_2]$. C_1 is *properly covered by* C_2, denoted $C_1 \sqsubset C_2$ if in addition $C_1 \neq C_2$.

[Support, Frequent/Maximal Frequent Itemset]: The notion of support captures how often an itemset occurs in a database and comes in two flavors. The *absolute support* of an itemset X in DB, defined as $\pi^A(X, \mathsf{DB}) = |\{T_i \in \mathsf{DB} | X \subseteq T_i\}|$, is the number of transactions in DB that contain X. The *(relative) support* of an itemset X in DB is the fraction of transactions in DB that contain X, and is defined as, $\pi(X, \mathsf{DB}) = \frac{\pi^A(X,\mathsf{DB})}{|\mathsf{DB}|}$. An itemset X is said to be *frequent* if $\pi(X, \mathsf{DB}) \geq \pi^{\min}$, where $0 < \pi^{\min} \leq 1$ is a user-specified minimum support threshold. A frequent itemset $X \in \mathbf{F}$ is *maximal* if it has no frequent superset, i.e., $(\nexists Y \mid (X \subset Y) \wedge (Y \in \mathbf{F}))$. A collection of frequent (or maximal) itemsets is denoted as $\mathbf{F}(\mathsf{DB}, \pi^{\min})$ or $\mathbf{F}$ (resply. $\mathbf{MF}(\mathsf{DB}, \pi^{\min})$ or $\mathbf{MF}$).

Definition 2.1 *[Containment Property] Let $C_1, C_2, \ldots C_k$ be k itemset collections over $\mathcal{I}$. $C_1, \ldots, C_k$ satisfy the* containment property *iff $C_k \sqsubseteq C_{k-1} \sqsubseteq \ldots \sqsubseteq C_2 \sqsubseteq C_1$. In such a case, $C_1, \ldots, C_k$ are said to form a* chain*.*

Frequent itemsets are closed under the $\subseteq$ operation and is widely used in itemset mining algorithms to generate candidates and pruning. More formally,

Proposition 2.2 *[6] Any subset of a frequent itemset is frequent: $X \in \mathbf{F}$ and $Y \subseteq X$ implies $Y \in \mathbf{F}$.* □

Frequent and maximal frequent itemsets are related in terms of induced subsets and the maximal itemsets of a given frequent itemset collection, is the smallest set of itemsets from which the frequent itemsets can be identified (but not their support values).

Proposition 2.3 *[16] Given a database DB and a minimum support level $\pi^{\min}$, $\mathbf{F}(\mathsf{DB}, \pi^{\min}) = [\mathbf{MF}(\mathsf{DB}, \pi^{\min})]$. Furthermore, $\mathbf{MF}$ is the smallest collection of frequent itemsets from which the elements of $\mathbf{F}$ (not their support values) can be derived.* □

We now define the notion of *isomorphic itemset collections* which is used later in our paper.

Definition 2.4 *[Isomorphic Itemset Collections] Let I_1 and I_2 be two itemset collections over $\mathcal{I}$. I_1 and I_2 are said to be* isomorphic itemset collections *iff there exists a bijection $\lambda : \mathcal{I} \to \mathcal{I}$ such that $\beta : I_1 \to I_2$ is a bijection from I_1 to I_2 where for all $X = \{x_1, \ldots, x_k\} \in I_1$, $\beta(X) = \{\lambda(x_1), \ldots, \lambda(x_k)\}$.*

Intuitively, for two isomorphic itemset collections, we can transform one itemset collection into another by relabeling the individual items in the collection uniformly across the entire collection.

Ordering plays an important role in the performance of algorithms that mine for itemsets. Various types of orderings can be defined on itemset collections. Two of the popular orderings on itemsets are the *lexicographic* (or lex) and the *co-lexicographic* (or colex) orderings, which are defined as follows.

Definition 2.5 *[Lex and Colex Order] Let $X, Y \in \mathcal{F} \cap \mathcal{I}^{(k)}$ be any two distinct k-itemsets in $\mathcal{F}$, with $X = x_1 x_2 \cdots x_k$ and $Y = y_1 y_2 \cdots y_k$. The* lexicographic *(or* lex*) ordering $\preceq_l$ is given as: $X \preceq_l Y$ if and only if $\exists z < k$ such that $\forall i : 1 \leq i < z, x_i = y_i$ and $x_z < y_z$. In contrast the* colex [1] *or* squashed *ordering $\preceq_c$ is given as: $X \preceq_c Y$ if and only if $\exists z < k$ such that $\forall i : z < i \leq k, x_i = y_i$ and $x_z < y_z$. Both lex and colex ordering are total orders on k-itemsets. We define the* rank *of a k-itemset as its position in the ordering, the first element having a rank of 1. We denote by $\mathcal{C}^{(k)}(m)$ the first m itemsets in $\mathcal{I}^{(k)}$ in colex order.*

2.1. Problem Statement

The database generation problem has two variants depending on the type of input. The first problem takes D-sequences of k maximal collections as input while the second type, takes the actual maximal collections as input.

Problem 2.6 *[Database Generation for D-sequences] Let $D_1, D_2, \ldots, D_k$ be the D-sequences of k maximal itemset collections. Construct a database DB and k minimum support levels $\pi_1^{\min}, \pi_2^{\min}, \ldots, \pi_k^{\min}$ such that*

1. $0 < \pi_1^{\min} \leq \pi_2^{\min} \ldots \leq \pi_k^{\min}$
2. $\langle \mathbf{MF}(\mathsf{DB}, \pi_i^{\min}) \rangle = D_i, \forall 1 \leq i \leq k$

Problem 2.7 *[Database Generation for maximal collections] Let $M_1, M_2, \ldots, M_k$ be k maximal itemset collections. Construct a database DB (if it exists) and k minimum support levels $\pi_1^{\min}, \ldots, \pi_k^{\min}$ such that*

1. $0 < \pi_1^{\min} \leq \pi_2^{\min} \ldots \leq \pi_k^{\min} \leq 1$
2. $\mathbf{MF}(\mathsf{DB}, \pi_j^{\min}) = M_j$ *for all* $1 \leq j \leq k$.

These two problems address the task of embedding k different distributions (D-sequences) or actual itemset collections into the generated database, such that mining at $\pi_i^{\min}$ yields M_i as the maximal frequent itemsets. The

[1] An alternate definition is given as: $X \preceq_c Y$ if and only if the largest item in symmetric difference of X and Y is in Y only.

corresponding *optimization problems* require the generation of a database with the smallest number of transactions satisfying the conditions of the problem.

It is important to differentiate between the two variants of the database generation problem. The generation of a database which takes as input a set of D-sequences, may require an additional procedure to generate the maximal itemset collections corresponding to the D-sequences. With respect to optimization results, any optimizations that hold for the case when D-sequences are provided as input also hold for the case when the maximal itemset collections are provided as input. The converse is not necessarily true. Any optimizations that are applied to a specific set of maximal itemset collections that are explicitly provided as input, need not necessarily generalize to all instances which correspond to a given set of D-sequences. However, there are some general optimizations for maximal itemset collections which may hold for D-sequences as well.

Note 2.8 *A given sequence of* D*-sequences may not be a feasible sequence of distributions. To check for feasibility, one could use and directly extend the methods in [16]. For this work, we assume that the input comes from the distribution of maximal collections mined from a database at various (increasing or decreasing) levels of minimum support and is hence, feasible.*

3. Some Basic Observations

We first summarize some basic results about databases and itemsets which are used later.

Proposition 3.1 [6] *Let* DB *be a database and let* X *and* Y *be two itemsets over* $\mathcal{I}$ *such that* $X \subseteq Y$*. Then* $\pi^A(X, \mathsf{DB}) \geq \pi^A(Y, \mathsf{DB})$. □

The following proposition states that at any given support level, the maximal collection obtained at that support level is contained in the frequent itemset collection at that support level.

Proposition 3.2 $\mathbf{MF}(\mathsf{DB}, \sigma) \subseteq \mathbf{F}(\mathsf{DB}, \sigma)$. □

Proposition 3.3 *Let* DB *be a database and let* σ_1 *and* σ_2 *be two values of support. If* $\sigma_1 \leq \sigma_2$ *then:*
1) $\mathbf{F}(\mathsf{DB}, \sigma_2) \subseteq \mathbf{F}(\mathsf{DB}, \sigma_1)$*, and*
2) $\mathbf{MF}(\mathsf{DB}, \sigma_2) \sqsubseteq \mathbf{MF}(\mathsf{DB}, \sigma_1)$.

Proof: 1) Let $\sigma_1 \leq \sigma_2$. $X \in \mathbf{F}(\mathsf{DB}, \sigma_2)$ implies that X occurs in at least a fraction σ_2 of DB which implies that it occurs in at least a fraction $\sigma_1 \leq \sigma_2$ of DB. Hence, $X \in \mathbf{F}(\mathsf{DB}, \sigma_1)$, and $\mathbf{F}(\mathsf{DB}, \sigma_2) \subseteq \mathbf{F}(\mathsf{DB}, \sigma_1)$.
2) By proposition 3.2 $\mathbf{MF}(\mathsf{DB}, \sigma_2) \subseteq \mathbf{F}(\mathrm{DB}, \sigma_2)$, and $\mathbf{F}(\mathrm{DB}, \sigma_2) \subseteq \mathbf{F}(\mathsf{DB}, \sigma_1)$. So $X \in \mathbf{MF}(\mathsf{DB}, \sigma_2)$ implies $X \in \mathbf{F}(\mathsf{DB}, \sigma_1)$. By definition of **MF**, this implies $\exists Y \in \mathbf{MF}(\mathsf{DB}, \sigma_1)$ such that $X \subseteq Y$. Thus $\mathbf{MF}(\mathsf{DB}, \sigma_2) \sqsubseteq \mathbf{MF}(\mathsf{DB}, \sigma_1)$. □

Proposition 3.3 can be directly generalized to more than two support values as follows.

Lemma 3.4 *Let* DB *be a database and let* $0 \leq \sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_k \leq 1$ *be* k *values of support. Then for* $1 \leq i \leq j \leq k$*: 1)* $\mathbf{F}(\mathsf{DB}, \sigma_j) \subseteq \mathbf{F}(\mathsf{DB}, \sigma_i)$*, and 2)* $\mathbf{MF}(\mathsf{DB}, \sigma_j) \sqsubseteq \mathbf{MF}(\mathsf{DB}, \sigma_i)$. □

From definition 2.1 and lemma 3.4, we have,

Corollary 3.5 *Let* DB *be a database and* $\sigma_1, \ldots, \sigma_k$ *be* k *values of support such that* $\sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_k$*. Then* $\mathbf{MF}(\mathsf{DB}, \sigma_1), \ldots, \mathbf{MF}(\mathsf{DB}, \sigma_k)$ *satisfies the containment property.* □

The following lemma shows that the maximal transactions are the maximal itemsets obtained from any database at the lowest level of support (absolute support of 1 or relative support of $\frac{1}{|\mathsf{DB}|}$).

Lemma 3.6 *Let* DB *be a database. Let* M_τ *be the set of all the maximal transactions in* DB*. Then, at support* $\sigma = \frac{1}{|\mathsf{DB}|}$, $\mathbf{MF}(\mathsf{DB}, \sigma) = M_\tau$.

Proof: If $T_i \in M_\tau$ then be definition, $\nexists T_j \in M_\tau, \ni T_i \subset T_j$, and $\pi^A(T_i) = 1$ implies $T_i \in \mathbf{MF}(\mathsf{DB}, \sigma)$. If $T_i \in \mathbf{MF}(\mathsf{DB}, \sigma)$, then $\nexists X \subseteq \mathcal{I}, \ni T_i \subset X$. Since every transaction is also an itemset, this implies $T_i \in M_\tau$. □

4. Database Generation in Stages

In [16], we used a generation procedure to demonstrate the existence of a database when feasible D-sequences are provided as input. We generated the collections in colex order and showed the procedure to work for such collections. In this section, we prove a stronger result by showing that the generation procedure works not only for colex ordered collections but for *any* sequence of maximal itemset collections satisfying the containment property (Definition 2.1). We analyze the procedure and provide an upper bound on the number of transactions in the database generated by the procedure.

Definition 4.1 *Let* DB *be a database. Define* $\hat{\mathcal{M}}(\mathsf{DB})$ *as* $\hat{\mathcal{M}}(\mathsf{DB}) = \max\{\pi^A(x, \mathsf{DB}) : x \in \mathcal{I}\}$*. In other words,* $\hat{\mathcal{M}}(\mathsf{DB})$ *is the maximum of the absolute support values of all the singleton items that occur in* DB*.*

To facilitate understanding, we define the following database generation operator.

Algorithm DB-Gen
Given: k maximal collections $M_1, \ldots, M_k$ satisfying the containment property
Output: DB and k real numbers $\pi_1^{min}, \ldots, \pi_k^{min}$ such that $\mathbf{MF}(DB, \pi_i^{min}) = M_i$, $1 \leq i \leq k$
Steps:
1. $DB_1 = \mathcal{D}(M_1)$; $c_1 = 1$;
2. $j = 2$; $DB = DB_1$;
3. $c_2 = \hat{\mathcal{M}}(DB) + 1$;
4. **while**$(j <= k)$
   (a) $DB_j = \mathcal{D}^{c_j}(M_j)$; (Use $DB_j = \mathcal{B}^{c_j}(M_j)$ for Bags)
   (b) $DB = DB \bigcup DB_j$
   (c) $c_{j+1} = \hat{\mathcal{M}}(DB) + 1$
   (d) $j = j + 1$;
5. **for**$(i = 1; i <= k; i = i + 1)$ $\pi_i^{min} = \frac{c_i}{|DB|}$;

Figure 1. Construct DB from k maximal collections

Definition 4.2 *[Database Generation Operator] Let $\mathcal{F}$ be a non-empty itemset collection, and let $n \geq 1$ be an integer. Let $\{X\}^n$ denote the multiset with n copies of X, i.e., $\{X\}^n = \{X_1, \ldots, X_n\}$, such that $X_i = X, \forall i \in [1, n]$. We define $\mathcal{D}^n(\mathcal{F})$ to be an operation that generates a database that consists of n copies of each itemset in $\mathcal{F}$, i.e., $\mathcal{D}^n(\mathcal{F}) = \bigcup_{X \in \mathcal{F}} \{X\}^n$. We denote $\mathcal{D}^1(\mathcal{F})$ by $\mathcal{D}(\mathcal{F})$. Let DB be an existing database, we assume that the operator ensures that every transaction in $\mathcal{D}^n(\mathcal{F})$ or $DB' = DB \cup \mathcal{D}^n(\mathcal{F})$ has a unique identifier.*

The following proposition states that $\hat{\mathcal{M}}(DB)$ is an upper bound for the support of *any* itemset in DB and is a direct consequence of Proposition 3.1 and definition 4.1.

Proposition 4.3 *Let DB be a database. For any $X \subseteq \mathcal{I}$, $\pi^A(X, DB) \leq \hat{\mathcal{M}}(DB)$.* □

Proposition 4.3 forms the central fact in the database generation algorithm **DB-Gen** given in figure 1. Algorithm **DB-Gen** generates the databases in k stages. In the first stage, it adds the elements of M_1 as transactions of the database and computes $\hat{\mathcal{M}}(DB)$ for the next stage of generation. At stage j in the generation process, it dynamically maintains the value of support for the singleton items and computes the $\hat{\mathcal{M}}$ value. It uses this value to replicate the itemsets in M_j and applies the database generation operator to add the replicated collection to the database. The procedure remembers the replication factor computed at each stage to compute the support values to be output at the end of the procedure. The central features of Algorithm DB-Gen are summarized below.

- The algorithm outputs the database in k stages.
- During the j^{th} stage, itemsets from M_j are added as transactions in the database. Each itemset in M_j is replicated the *same* number of times in the database.
- At the end of the j^{th} stage, $M_1 \ldots M_j$ are maximal collections embedded in DB at absolute support levels $c_1, \ldots, c_j$ respectively.

Lemma 4.4 *Let DB be the database generated by algorithm DB-Gen of figure 1. For any transaction $T \in DB$, $T \in \bigcup_{i=1}^{k} M_i$. Furthermore, the maximal transactions of DB is precisely the set M_1.* □

The following theorem proves the correctness of algorithm DB-Gen.

Theorem 4.5 *Given k maximal collections $M_1, \ldots, M_k$ satisfying the containment property, the database DB generated and the support values $\pi_1^{min}, \ldots, \pi_k^{min}$ computed by algorithm DB-gen satisfy the property that $\mathbf{MF}(DB, \pi_i^{min}) = M_i$, for $1 \leq i \leq k$.*

Proof: Consider any stage j of the generation procedure. Clearly, if $j = 1$, then the procedure produces M_1 as the maximal collection at absolute support 1 (and hence at relative support $\frac{1}{DB}$) by lemma 4.4. Let $j > 1$. Let DB_{j-1} be the database generated up to the beginning of the j^{th} stage. By proposition 4.3, no itemset has a minimum support of c_j calculated by the algorithm. Hence, adding c_j transactions for each element of M_j to the database will make the element frequent at absolute support level c_j in DB. Thus, M_j is the maximal frequent itemset collection at support level. Note that the containment property is essential to ensure that when M_j is added, it does not affect the frequencies of M_{j-1}, that was already added to the database. □

Proposition 4.6 *Let DB be a database and let $\hat{\mathcal{M}}(DB)$ be as defined in definition 4.1. Then $\hat{\mathcal{M}}(DB) \leq |DB|$.*

Proof: No itemset can have absolute support greater than the number of transactions in the database. The inequality follows directly. □

Using repeated applications of proposition 4.6, we obtain an upper bound on the number of transactions in the database after any stage in the generation procedure that is given by the following theorem.

Theorem 4.7 *Let m_j denote the $|M_j|$ for $1 \leq j \leq k$ and let DB_j be the database generated at the end of the j^{th} stage of algorithm DB-Gen. Then, $|DB_j| \leq m_j(1 + m_{j-1} * (1 + \ldots * (1 + m_2 * (1 + m_1)))) \leq \sum_{i=1}^{j} \prod_{l=i}^{j} m_l$.*

Proof: $|DB_1| = |M_1| = m_1$ is true by step 1 of the algorithm. In stage 2, each of the m_2 maximal itemsets in M_2 is replicated $c_2 = 1 + \hat{\mathcal{M}}(DB_1) \leq 1 + |DB_1| \leq 1 + m_1$ times and hence $|DB_2| \leq m_2 * (1 + m_1) \leq m_2 + m_1 * m_2$. Proceeding all the way up to stage j, we get the equation in the theorem. □

From the above theorem, we have the following upper bound for the number of transactions in the database generated by algorithm DB-Gen.

Theorem 4.8 *Let $M_1, \ldots, M_k$ be k maximal itemset collections satisfying the containment property and let m_j denote $|M_j|$ for $1 \leq j \leq k$. Then, the algorithm DB-Gen given in figure 1 generates a database DB with at most $O(k \cdot \prod_{i=1}^{k} m_i)$ transactions.*

Proof: From theorem 4.7 the maximum value of summation is $\prod_{i=1}^{k} m_i$, and there are k terms, giving the desired bounds. □

Thus, algorithm DB-Gen produces as output, a database where the number of transactions is proportional to the product of the size of the input maximal collections. Note that this number can be prohibitively large for practical collections of maximal itemsets. For example, for three D-sequences or maximal collections, each containing 1000 maximal itemsets ($M_i = 1000$ and $k = 3$), the procedure can potentially generate a *billion* transactions!!!

4.1. Compression using Transaction Bags

In this section, we study an alternative to generating replicated transactions to avoid generating such large databases. One of the primary reasons for the explosion in the number of transactions is the replication of itemsets during each stage of database generation. During any stage j, $j > 1$, every itemset in M_j is repeatedly written into the database c_j times. This causes a waste of storage space. To overcome the disk space consumption by repeated duplication of transactions, we proposed the idea of a transaction map in [16].

Definition 4.9 *[Transaction Map/Bag] A **transaction map** or **transaction bag** is a triple $\langle C, i, T \rangle$, where $C > 0$ denotes the count of itemset T, $i \geq 0$ is a unique map/bag identifier, and $T \subseteq \mathcal{I}$ is an itemset. A transaction bag database is a set of transaction bags.*

The database generation operator can be appropriately generalized to generate transaction bags as follows.

Definition 4.10 *[Transaction Bag Generator] Given an itemset collection $\mathcal{F}$ and an integer $n \geq 1$, we define $\mathcal{B}^n(\mathcal{F})$ to be an operation that generates a database that consists of one transaction bag with count n for each $X \in \mathcal{F}$, i.e., $\mathcal{B}^n(\mathcal{F}) = \{\langle n, i_X, X \rangle | X \in \mathcal{F}\}$. We denote $\mathcal{B}^1(\mathcal{F})$ by $\mathcal{B}(\mathcal{F})$. Let DB be an existing database, we assume that the operator ensures that every transaction bag in $\mathcal{B}^n(\mathcal{F})$ or $DB' = DB \cup \mathcal{B}^n(\mathcal{F})$ has a unique bag identifier.*

Using transaction bags, we can achieve significant compression as follows. We use $DB_j = \mathcal{B}^{c_j}(M_j)$ instead of the statement $DB_j = \mathcal{D}^{c_j}(M_j)$ in step 4 of algorithm DB-Gen to generate transaction bags.

Lemma 4.11 *In stage j of algorithm DB-Gen, the number of transaction bags that are generated by the procedure is $|M_j|$.*

Proof: Stage j adds one transaction bag for every itemset in M_j and hence the lemma follows. □

From lemma 4.11, we can compute the exact number of transaction bags in the database generated by algorithm DB-Gen as follows.

Theorem 4.12 *Let DB be a transaction bag database generated by algorithm DB-Gen for the input set of maximal collections $M_1, \ldots, M_k$. Let $|M_j| = m_j$, for $1 \leq j \leq k$. Then $|DB| = \sum_{i=1}^{k} m_i$.* □

Theorem 4.12 shows that by using transaction bags, we can obtain significant compression over the corresponding transaction databases in terms of the transactions that are actually stored in the database. For the same example discussed earlier, when there are three maximal itemset collections each consisting of 1000 maximal itemsets, the transaction bag database will consist of at most 3000 transaction bags as opposed to the at most 1 billion transactions in the original procedure, achieving an order of compression of 10^6. Transaction bags may be a good alternative to transactions especially when a lot of repetitions happen. However, the limitations of DB-Gen can also constrain the use of transaction bag databases, as shown empirically in section 6.

5. An Optimization to DB-Gen

The database generation algorithm DB-Gen assumes a pessimistic repetition factor for each stage of generation. The main reason for using such a conservative repetition factor is the ease with which it can be computed. For determining the value of $\hat{\mathcal{M}}(DB)$ at each stage, the algorithm only needs to maintain the counts of individual items in a vector of size $|\mathcal{I}|$, where $\mathcal{I}$ is the set of items over which the itemsets are constructed. The trade-off between the ease of computation and the size of output imposes limitations to the algorithm which constrains its usage in practice (see section 6). A second limitation in the generation procedure is that all the itemsets in a maximal collection are repeated the same number of times at any stage (c_j in the j^{th} stage). The procedure is not flexible enough to add itemsets on an *as needed* basis to the database. In this section, we present an optimization that can potentially reduce the number of transactions (the count value in case of transaction bags) in the output database (but with an overhead in time and space caused by the intermediate structures).

The main motivation behind the optimization to algorithm DB-Gen is based on the following intuition. At stage i, Algorithm DB-Gen repeatedly adds the itemsets in M_i to make them frequent at a *high enough* value of absolute support. This *high enough* value of support is computed using the naive upper bound given by $\hat{\mathcal{M}}$ (which is easy to compute). However, this repetition factor ignores the support of each itemset in M_i in the

database already generated so far (i.e., before the addition). For instance, in stage 2, the support of itemsets in M_2 in the current database, are not taken into account when adding those itemsets to the database. Hence, if itemsets in M_2 need to have a certain value of support say h, they are replicated h times, even though they occur with a certain support in the database generated thus far. Secondly, the replication value of $\hat{\mathcal{M}}(DB)$ is too conservative. Is there a better replication value we can use? To address this issue, we first extend the definition of $\hat{\mathcal{M}}(DB)$ to a given itemset collection as follows.

Definition 5.1 *Let* DB *be a database and* $\mathcal{F}$ *be an itemset collection. Then* $\hat{\mathcal{M}}(\mathcal{F}, DB)$ *is defined as follows:* $\hat{\mathcal{M}}(\mathcal{F}, DB) = \max\{\pi^A(X, DB) : X \in \mathcal{F}\}$. *In other words,* $\hat{\mathcal{M}}(\mathcal{F}, DB)$ *is the maximum of the absolute support values of all the itemsets in* $\mathcal{F}$.

The concept of *negative border* for a collection of itemsets is well defined [5, 13]. The negative border gives the collection of minimal itemsets that are infrequent with respect to the item universe. We modify this definition to suit our current purpose, by defining the negative border of an itemset collection with respect to another itemset collection as follows.

Definition 5.2 *Let* M_1 *and* M_2 *be two maximal collections such that* $M_2 \sqsubseteq M_1$. *Then the* relative negative border *of* M_2 *with respect to* M_1 *is defined as* $\delta(M_1, M_2) = \{X \in [M_1] - [M_2] | (\nexists Y \in [M_1] - [M_2] : Y \subset X)\}$.

The set $\delta(M_1, M_2)$ consists of precisely the minimal itemsets in $[M_1] - [M_2]$ (recall that $[M_i]$ denotes the set of itemsets induced by M_i). An example is shown is figure 2 for two maximal collections M_1 and M_2. In the sequel, by negative border, we mean the relative negative border defined above. Intuitively, if a database DB has been generated up to stage $j-1$, then the minimum support for the j^{th} stage can be determined by using the count values of itemsets in M_j and $\delta(M_{j-1}, M_j)$, instead of the $\hat{\mathcal{M}}$ value.

Definition 5.3 *Let* DB *be a database and let* M_1 *and* M_2 *be two maximal collections such that* $M_2 \sqsubseteq M_1$. *Let* $\eta = \hat{\mathcal{M}}(\delta(M_1, M_2), DB)$. *For a given integer* $\sigma \geq 0$, *define* $\gamma(M_1, M_2, DB, \sigma)$ *as the smallest value in the set* $\{\pi^A(X, DB) | X \in M_2\}$ *that is greater than or equal to* $M_\delta = \max\{\sigma, \eta\}$, *if it exists and* M_δ, *otherwise.*

The above definition of $\gamma(M_1, M_2, DB, \sigma)$ gives a value of minimum support at which the following two conditions are satisfied:

1. It is the smallest value of absolute support which itemsets in M_j need to have in the database.
2. No item in $\delta(M_1, M_2)$ has an absolute support of γ.

Hence, using the γ values computed at every stage, one could embed the itemsets in M_2 without making any itemset in the border frequent at this value of absolute support.

M_1

1234	1235	1456	2567
157	167	467	
36	37		

M_2

123	146	125
56	37	67

$[M_1] - [M_2]$

1234	1235	1456	2567				
124	134	234	135	235	145	156	
256	456	157	257	167	267	467	567
24	34	35	45	26	36	17	27	47	57

$\delta(M_1, M_2)$

156	24	34	35	45	26
36	17	34	27	47	

Figure 2. Example of $\delta(M_1, M_2)$ for two collections

Lemma 5.4 *Let* M_1 *and* M_2 *be two maximal collections such that* $M_2 \sqsubseteq M_1$. *Let* DB *be a database of transactions and let* π_1^A *be a value of absolute support at which* M_1 *is maximally frequent. Let* γ *be as defined in definition 5.3. Then,* $\forall Y \in \delta(M_1, M_2)$, $\pi^A(Y, DB) < \gamma(M_1, M_2, DB, \pi_1^A) + 1$.

Proof: Follows from definitions 5.2 and 5.3. □

A consequence of lemma 5.4 is algorithm γ-DB-Gen shown in figure 3 for database generation, which uses the γ values at each stage instead of the $\hat{\mathcal{M}}$ values. This algorithm has the additional overhead of computing $\delta(M_{j-1}, M_j)$ and computing $\gamma(M_{j-1}, M_j, DB, c_{j-1})$ at each stage of generation. Computing $\delta(M_{j-1}, M_j)$ can be a costly step at every stage. The task of computing the γ values, essentially boils down to computing the absolute support of itemsets in M_j and $\delta(M_{j-1}, M_j)$ and this can be done in one pass over the database generated thus far at every stage. For some specific types of maximal collections, if we know that the absolute support of itemsets in M_j is greater than those in $\delta(M_{j-1}, M_j)$, then the computation of $\delta(M_{j-1}, M_j)$ can be avoided altogether. It should also be noted that the replication factor is not the same for *all* the itemsets during any given stage. Instead, the algorithm replicates itemsets on an as needed basis. From lemma 5.4, we have the following theorem.

Theorem 5.5 *Let* $M = \langle M_1, \ldots, M_k \rangle$ *be* k *maximal collections satisfying the containment property. Let* DB *be the transaction database generated by DB-Gen for* M *and* DB_γ *be the transaction database generated by* γ*-DB-Gen. Then,* $|DB_\gamma| \leq |DB|$.

Algorithm γ-DB-Gen
Given: k maximal collections $M_1, \ldots, M_k$ satisfying the containment property
Output: DB and k real numbers $\pi_1^{min}, \ldots, \pi_k^{min}$ such that $\mathbf{MF}(DB, \pi_i^{min}) = M_i$, $1 \leq i \leq k$
Steps:
1. $DB_1 = \mathcal{D}(M_1)$; $c_1 = 1$;
2. $j = 2$; $DB = DB_1$;
3. $c_2 = \gamma(M_1, M_2, DB, c_1) + 1$;
4. **while**$(j <= k)$
   (a) **for** $X \in M_j$,
      i. **if**$\{\pi^A(X, DB) < c_j\}$ **then**
         $l = c_j - \pi^A(X, DB)$ **else** $l = 0$;
      ii. $DB_j = DB_j \bigcup \mathcal{D}^l(X, M_j)$;
   (b) $DB = DB \bigcup DB_j$;
   (c) $c_{j+1} = \gamma(M_j, M_{j+1}, DB, c_j) + 1$; //(for $j < k$)
   (d) $j = j + 1$;
5. **for**$(j = 1; j <= k; j = j + 1)$ $\pi_i^{min} = \frac{c_i}{|DB|}$;

Figure 3. Optimized γ-DB-Gen Algorithm

Proof: Immediately follows from the fact that $\gamma(M_j, M_{j+1}, DB, c_j)$ computed by γ-DB-Gen at every stage is at most $\hat{\mathcal{M}}(DB)$ computed by DB-Gen at the corresponding stage. □

5.1. Closely Packed Maximal Collections

Closely packed maximal collections are generated using the *colex* ordering of itemsets. The nice property of these collections which makes them attractive for study is that these collections can be generated efficiently. Given a D-sequence, a closely packed maximal collection satisfying this D-sequence can be generated in linear time (*linear in the output size*). More formally, closely packed collections are defined as below.

Definition 5.6 *[Closely Packed Collection]*
Let $D = \langle d_1, \ldots, d_k \rangle$ *be any D-sequence. Let* M *be the maximal collection constructed by algorithm Construct-Closely-Packed shown in figure 4 for* D. *A maximal collection* W *with* $\langle W \rangle = D$ *is said to be a* ***closely packed collection*** *iff* W *is isomorphic to* M.

Intuitively, a closely packed maximal collection is one where the items can be renumbered to give a *colex ordered* maximal collection generated by the algorithm in figure 4. For a given D-sequence $D = \langle d_1, \ldots, d_k \rangle$, algorithm Construct-Closely-Packed given in figure 4 generates a *colex ordered* closely packed maximal collection M, with $\langle M \rangle = D$, in time proportional to $|M|$. In [16], the authors show that the time is linear in the size of the output. Generating arbitrary maximal collections with distributions specified by a given D-sequence may be time consuming. This is the main motivation behind considering D-sequences in which the itemsets are in some predefined order.

In the following section, the D-sequences obtained from the real datasets are used to generate closely packed maximal collections which are used as input for the database generation procedure.

Algorithm Construct-Closely-Packed
Given: A D−sequence $D = \langle d_1, \ldots, d_k \rangle$
Output: A closely packed maximal collection M over $\mathcal{I} = \{1, 2, \ldots\}$ such that $\langle M \rangle = D$.
Steps:
1. $M = \mathcal{C}^{(k)}(d_k)$ // first d_k k−itemsets in colex order
2. $j = k - 1$
3. **while**$(j >= 1)$
   (a) r_j = largest rank of the itemsets in $[M]_j$
   (b) Add the j−itemsets of rank $r_j + 1, \ldots, r_j + d_j$ in colex ordering of j−itemsets to M
   (c) $j = j - 1$;

Figure 4. Constructing Closely Packed Collections

6. Experimental Study

We present preliminary experimental results to analyze how applicable the generation procedures are in practice. We use three real datasets chess, mushroom and connect [2], with database cardinalities as shown in figure 5. For each dataset, we use GenMax [10] to obtain the maximal frequent itemset distributions (i.e., the D-sequences) at various increasing values of minimum support. We used $k = 6$, D-sequences from chess, $k = 8$ from mushroom and $k = 14$ from connect datasets respectively. Figure 6 shows the sizes of the maximal sets for each D-sequence, i.e., the k values, $\{|M_1|, \cdots, |M_k|\}$. We then use Construct-Closely-Packed to generate closely packed maximal itemset collections satisfying these input distributions [3]. The maximal collections thus generated are used as input for the database generation procedure.

Source	# trans. in dataset
chess	3196
connect	67557
mushroom	8124

Figure 5. Real Databases Used

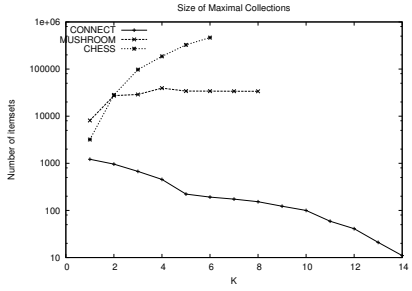

Figure 6. Sizes of $M_i, i \in [1, n]$ for Real Datasets

[2] The UCI KDD Archive available at kdd.ics.uci.edu

[3] In general, the generation procedure works for ANY input set of maximal collections that satisfy the containment property

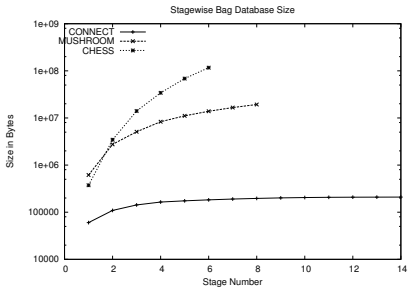

Figure 7. Stage-wise Database Size in Bytes

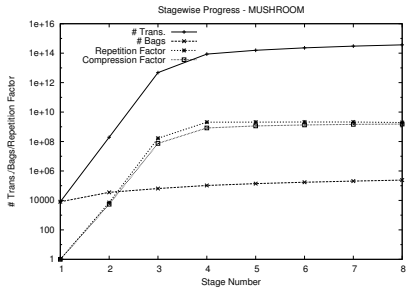

Figure 10. Generation for mushroom dataset

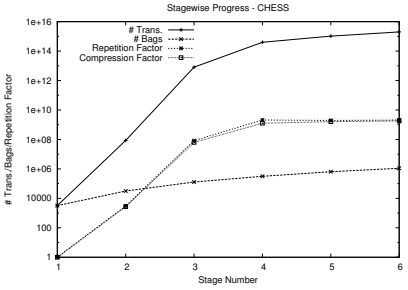

Figure 8. Generation for chess dataset

Figures 8, 9 and 10 show the stage-wise progress of results of algorithm DB-Gen. The plots show the number of transactions, the repetition factor, the number of transaction bags and the compression factor of using bag databases over transaction databases for the different stages; j^{th} stage corresponds to embedding $M_1 \cdots M_j$ in DB. From the figures, it can be observed that generating transaction bag databases can provide significant compression ranging between a factor of a few thousands in the early stages to up to a factor of a billion in the later stages. It is also interesting that for connect, the database that embeds M_i is in fact smaller than even the original database.

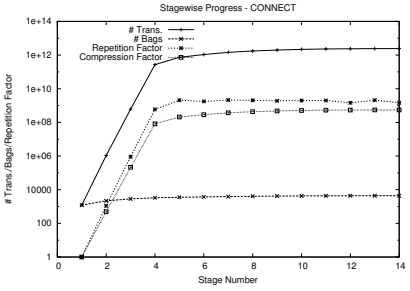

Figure 9. Generation for connect dataset

Figure 6 shows the bag database sizes corresponding to every stage of generation. For the input sequences corresponding to the connect dataset, the generated bag databases during each stage were hundreds of kilobytes in size. For the inputs corresponding to chess and mushroom datasets, the database sizes ranged from a few megabytes to a few hundred megabytes, which is quite reasonable in terms of size. A naive implementation of algorithm γ-Db-Gen can slow down the generation procedure significantly. The computation of negative border at each stage causes significant overhead in run time. Our ongoing work is addressing the problem of developing efficient approximations that can avoid this computation. Specifically, we are studying the existence of repetition factors that are more optimistic than $\hat{\mathcal{M}}$ but avoid the computation of the entire relative negative border.

7. Discussion

Although the results presented in the earlier section demonstrate the practical feasibility of using algorithm DB-Gen to generate bag databases of realistic size, the procedure faces some serious limitations. These limitations are discussed in this section and elaborated in more detail to illustrate the requirements of database generation techniques.

Firstly, the bag databases compress transactions by providing a count value with each bag to illustrate the repetition factor of the itemset associated with that bag. The counts associated with the bags in earlier stages are reasonable in terms of the repetition. But this value can grow large very quickly. For instance, in the instance of the input from the mushroom dataset, the repetition factor grows to a billion at stage 7. This might cause a problem when the input is a large set of maximal collections each of high cardinality.

Secondly, the constraint of adding only maximal itemsets from maximal collections during each stage, is primitive. One can take advantage of computing covers for groups of itemsets and using these covers to account for multiple itemsets in terms of counts. Such an approach can also reduce the large count values that

are evident in later stages. An interesting direction of work is to allow the addition of any arbitrary itemset as a transaction (either a subset or superset of a maximal itemset). Also the generation process is sensitive to the initial maximal collections provided as input. The instances taken from the chess and mushroom datasets in the examples in the earlier section, were all obtained at the lowest levels of absolute support from these datasets.

Thirdly, the complexity of the optimization problem needs further investigation. Earlier work on hypergraph transversal by Gunopulos et al [5] has great relevance to identifying the complexity of the optimization problem. In parallel to probing better generation procedures, we are also investigating the complexity of the optimization problem. Unlike the stage-wise generation procedures, the global optimization problem considers the maximal collections as a whole instead of one at a time.

Finally, the generation procedures considered here are sensitive to the size of the maximal collections/D-sequences provided as input. Since the generation proceeds stage-wise, the maximal collections are also embedded in a stepwise manner in the database. This means that M_i is the maximal collection that would be frequent in the database for any value of support in the range $[\pi_i^{min}, \pi_{i+1}^{min})$. The database size, repetition factor and compression factors are all very sensitive to the size of the maximal collections.

8. Conclusions and Future Work

In this study, we formalized the synthetic database generation problem for maximal collections and D-sequences. We presented a stage-wise generation procedure that constructs a database and k levels of minimum support at which a given set of k maximal collections satisfying the closure property are obtained as output. We analyzed the generation procedure by providing an upper bound on the number of transactions output. We developed an alternative approach using transaction bags, which significantly compresses the size of the output database and we gave an upper bound on the number of transaction bags generated in the output database. We presented an optimization that can potentially reduce the number of transactions/bags generated by the stage-wise generation procedure. Finally, experimental results are provided to demonstrate the feasibility of the generation procedure in practice.

Several future directions naturally present themselves. An ongoing study is investigating existing literature on complexity results in hypergraph transversal and their relation to itemset mining and lattices for optimizations of the database generation problem. Some results in complexity has already been explored by Yang in [17]. A second direction is to study the variation in the two problems defined in the study. One needs to explore what kind of general results can be obtained for D-sequences and how they relate to maximal collections. We used closely packed maximal collections generated from D-sequences; it would be interesting to explore other efficient transformation schemes from D-sequences to the actual maximal collections generated. The question, what is the minimal transaction database embedding a given itemset collection, remains open.

Acknowledgments

The authors would like to thank Dr. Laks V.S. Lakshmanan for his valuable suggestions and comments.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB Conference*, 1994.

[2] R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD Conference*, 1998.

[3] Toon Calders and Bart Goethals. Mining all non-derivable itemsets. In *Principles of Data Mining and Knowledge Discovery*, 2002.

[4] Artur Bykowski et al. A condensed representation to find frequent patterns. In *ACM PODS Conference*, 2001.

[5] Dimitrios Gunopulos et al. Discovering all most specific sentences. *ACM TODS*, 2003.

[6] R. Agrawal et al. Fast discovery of association rules. In U. Fayyad and et al, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.

[7] Richard M. Karp et al. A simple algorithm for finding frequent elements in streams and bags. *ACM TODS*, 2003.

[8] A. Evfimievski, R. Srikant, R. Agrawal, and J. E. Gehrke. Privacy preserving mining of association rules. In *ACM SIGKDD Conference*, 2002.

[9] Bart Goethals, Floris Geerts, and Jan Van den Bussche. A tight upper bound on the number of candidate patterns. In *IEEE ICDM Conference*, 2001.

[10] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *IEEE ICDM Conference*, 2001.

[11] J. Han and M. Kamber. Data mining: concepts and techniques. Morgan Kaufman. 2000.

[12] Murat Kantarcioglu and Chris Clifton. Privacy preserving data mining of association rules on vertically partitioned data. In *ACM KDD Conference*, 2003.

[13] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1997.

[14] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained association rules. In *ACM SIGMOD Conference*, 1998.

[15] Ganesh Ramesh, William A. Maniatty, and Mohammed J. Zaki. Indexing and Data Access Methods for Database Mining. In *ACM SIGMOD Workshop - DMKD*, 2002.

[16] Ganesh Ramesh, William A. Maniatty, and Mohammed J. Zaki. Feasible itemset distributions in data mining: Theory and application. In *ACM PODS Conference*, 2003.

[17] Yang G. The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns. In *ACM KDD Conference*, 2004.

[18] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD Conference*, 1997.

[19] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *ACM KDD Conference*, 2001.