Discovering all Most Specific Sentences by Randomized Algorithms Extended Abstract

Dimitrios Gunopulos[1] and Heikki Mannila[2] and Sanjeev Saluja[3]

[1] Max-Planck-Insitut Informatik, Im Stadtwald, 66123 Saarbrücken, Germany.
gunopulo@mpi-sb.mpg.de
[2] University of Helsinki, Dept. of Computer Science, FIN-00014 Helsinki, Finland.
Heikki.Mannila@cs.helsinki.fi.
Work supported by Alexander von Humbold-Stiftung and the Academy of Finland.
[3] Max-Planck-Institut Informatik, Im Stadtwald, 66123 Saarbrücken, Germany.
saluja@mpi-sb.mpg.de

Abstract. Data mining can in many instances be viewed as the task of computing a representation of a theory of a model or a database. In this paper we present a randomized algorithm that can be used to compute the representation of a theory in terms of the most specific sentences of that theory. In addition to randomization, the algorithm uses a generalization of the concept of hypergraph transversal. We apply the general algorithm, for discovering maximal frequent sets in 0/1 data, and for computing minimal keys in relations. We present some empirical results on the performance of these methods on real data. We also show some complexity theoretic evidence of the hardness of these problems.

1 Introduction

Data mining has recently emerged as an important application area [11]. The goal of data mining can briefly be stated as "finding useful high-level knowledge from large masses of data". The area combines methods and tools from databases, machine learning, and statistics.

A large part of current research in the area can be viewed as addressing instances of the following problem: given a language, a frequency criterion, and a database, find all sentences from the language that are true in the database and satisfy the frequency criterion. Typically, the frequency criterion states that there are sufficiently many instances in the database satisfying the sentence. Examples where this formulation works include the discovery of association rules, strong rules, episodes, and keys. Using this *theory extraction formulation* [19, 20, 23] one can formulate general results about the complexity of algorithms for various data mining tasks.

The algorithms for the above mentioned data mining tasks typically operate in a bottom-up fashion: first the truth and frequency of the simplest, most general, sentences from the language are evaluated against the database, and then the process continues for more specific sentences. While this approach works reasonably well in several cases, problems can arise when the theory to be extracted

is large: there are lots of sentences to be output. An alternative method is to try to search for the most specific sentences from the language that satisfy the requirements: theses sentences determine the theory uniquely.

In this paper we present a randomized algorithm for locating the most specific true sentences satisfying the frequency criterion. Although our algorithm is randomized, it is complete, in the sense that it returns all most specific sentences. Briefly the method works as follows. We apply a randomized greedy search to locate some maximal elements from the language. We then use the simple fact that if some sets from an anti-chain are known, then every unknown set in the anti-chain must contain a *minimal transversal* of the *complements* of the known sets. The algorithm alternates between finding new random most specific true sentences and finding minimal transversals of the complements of the already discovered most specific true sentences, until no new most specific true sentence can be found.

We demonstrate this method by applying it to the problem of computing all maximal frequent sets of a 0/1 matrix for a given threshold, and the problem of computing of all minimal keys, or functional dependences, in a relational database. The computation of maximal frequent sets is a fundamental data mining problem which is required in discovering association rules [1, 2]. Minimal keys can be used for semantic query optimization, which leads to fast query processing in database systems [22, 18, 5, 26]. Here we refer to possible keys that exist in a specific instance of a relational database and are not designed as such.

The computation of sentences of a theory is an enumeration problem. The computation involves listing combinatorial substructures related with the input. For enumeration problems one of the definitions of efficiency is that the running time of algorithm be bounded by a polynomial function of input and output sizes. Such an algorithm is called as an output-polynomial time algorithm. For the problems that we discuss in this paper, output-polynomial algorithms are not known. In the absence of such provably efficient algorithms, we view our algorithm as an efficient alternative which can perform well in practice.

The rest of this paper is organized as follows. In Section 2 we present a model of data mining which formally defines the theory extraction problem. In Section 3 we formulate our general algorithm in this setting. Section 4 adapts our algorithm to the well-studied problem of finding maximal frequent sets, and also gives some complexity-theoretic evidence of hardness of this problem. Empirical results on the behavior of the algorithm are also given. In Section 5 we adapt the general algorithm of Section 3 to the problem of finding keys of relations, and present some empirical results. Finally, in Section 6 we give a short conclusion.

2 Data mining as theory extraction

The model of knowledge discovery that we consider is the following [19, 23, 20]. Given a database $\mathbf{r}$, a language $\mathcal{L}$ for expressing properties or defining subgroups of the data, and a frequency criterion q for evaluating whether a sentence $\varphi \in \mathcal{L}$ defines a sufficiently large subclass of $\mathbf{r}$. The computational task is to find

the theory of $\mathbf{r}$ with respect to $\mathcal{L}$ and q, i.e., the set $Th(\mathcal{L}, \mathbf{r}, q) = \{\varphi \in \mathcal{L} \mid q(\mathbf{r}, \varphi) \text{ is true}\}$.

We are not specifying any satisfaction relation for the sentences of $\mathcal{L}$ in $\mathbf{r}$: this task is taken care of by the frequency criterion q. For some applications, $q(\mathbf{r}, \varphi)$ could mean that φ is true or almost true in $\mathbf{r}$, or that φ defines (in some way) a sufficiently large or otherwise interesting subgroup of $\mathbf{r}$. The roots of this approach are in the use of *diagrams* of models in model theory (see, e.g., [7]). The approach has been used in various forms for example in [2, 8, 9, 16, 17, 21]. One should note that in contrast with, e.g., [8] our emphasis is on very simple representation languages.

Obviously, if $\mathcal{L}$ is infinite and $q(\mathbf{r}, \varphi)$ is satisfied for infinitely many sentences, (an explicit representation of) $Th(\mathcal{L}, \mathbf{r}, q)$ cannot be computed feasibly. Therefore for the above formulation to make sense, the language $\mathcal{L}$ has to be defined carefully. In case $\mathcal{L}$ is infinite, there are alternative ways of meaningfully defining feasible computations in terms of dynamic output size, but we do not concern ourselves with these scenarios. In this paper we assume that $\mathcal{L}$ is finite.

3 A randomized algorithm for computing $Th(\mathcal{L}, \mathbf{r}, q)$

We make the following assumption about language $\mathcal{L}$. There is a partial order $\preceq$ on the set of sentences of $\mathcal{L}$ such that q is monotone with respect to $\preceq$, that is, for all $\psi, \theta \in \mathcal{L}$ with $\theta \preceq \psi$ we have: if $q(\mathbf{r}, \psi)$, then $q(\mathbf{r}, \theta)$. [4] Denote by $rank(\psi)$ the *rank* of a sentence $\psi \in \mathcal{L}$, defined as follows. If for no $\theta \in \mathcal{L}$ we have $\theta \prec \psi$, then $rank(\psi) = 0$, otherwise $rank(\psi) = 1 + \max\{rank(\theta) \mid \theta \prec \psi\}$. For a sentence ψ, $rank(\psi)$ can be arbitrary large but finite. For $T \subset \mathcal{L}$, let T_i denote the set of the sentences of $\mathcal{L}$ with rank i.

The level-wise algorithm [23] for computing $Th = Th(\mathcal{L}, \mathbf{r}, q)$ proceeds by first computing the set Th_0 consisting of the sentences of rank 0 that are in Th. Then, assuming Th_i is known, it computes a set of *candidates*: sentences ψ with rank $i+1$ such that all θ with $\theta \prec \psi$ are in Th. For each one of these candidates ψ, the algorithm calls the function q to check whether ψ really belongs to Th. This iterative procedure is performed until no more sentences in Th are found. This level-wise algorithm has been used in various forms in finding association rules, episodes, sequential rules, etc. [2, 3, 24, 23, 1, 13, 14, 25]. The drawback with this algorithm is that it always computes the whole set $Th(\mathcal{L}, \mathbf{r}, q)$, even in the cases where a condensed representation of Th using *most specific sentences* would be useful.

Given Th, a sentence $\psi \in Th$ is a *most specific* sentence of Th, if for no $\theta \in Th$ we have $\psi \prec \theta$. Denote by $MTh = MTh(\mathcal{L}, \mathbf{r}, q, \preceq)$ the set of most specific sentences of $Th(\mathcal{L}, \mathbf{r}, q)$ with respect to $\preceq$.

Our general algorithm for computing MTh is based on repeatedly computing new most specific sentences in Th. To compute one most specific sequence, we use the following randomized algorithm:

[4] Note that this description is a fairly severe one. For example, a q defined in terms of statistical significance does not satisfy this condition.

Algorithm A_Random_MSS Find a random most specific sentence from Th.

1. $i := 0$.
2. $\psi :=$ **true**.
3. While (there is an immediate specialization θ of ψ
   such that $q(\mathbf{r}, \theta)$ holds) do: select such a θ randomly and let $\psi := \theta$.
4. Output ψ.

The algorithm assumes that **true** $\in \mathcal{L}$, and proceeds to specialize it successively until a most specific sentence is found. In Step 2, if ψ is initialized with an arbitrary sentence $s \in Th$ instead of "true", then the algorithm will find a random most specific sentence s' such that $s \preceq s'$.

After the computation of one or more new most specific sentences in MTh, we compute the *minimal-orthogonal-elements* (abbr. *min-ortho-element*) with respect to the collection of most specific sentences found so far. Let S be the set of most specific sentences computed. A *minimal orthogonal element* with respect to S is a sentence ψ in $\mathcal{L}$, such that ψ is unrelated to all the sentences in S under $\preceq$ and for no sentence $\phi \prec \psi$, this property holds. The definition of minimal orthogonal elements is a natural extension of the minimal hypergraph transversal [6, 10]. Then we can show the following lemma:

Lemma 1 *Let S be a set of most specific sentences in MTh, and let T be the set of all minimal orthogonal elements with respect to S. Then, if there is a sentence γ in MTh which is not in S, then there is a sentence β in T such that $\beta \preceq \gamma$.*

Proof: Any new most specific sentence γ cannot be either above or below (with respect to $\preceq$) another most specific sentence. γ must therefore be unrelated to all the sentences in S. This means that either $\gamma \in T$ or there is a sentence $\gamma' \preceq \gamma$ that is also unrelated to all sentences in S. Since the rank of γ is finite the lemma follows. □

This property allows us to limit the search to these sentences which are above the min-ortho-elements with respect to $\preceq$. Denote by Algorithm *A_Random_MSS(ψ_{init})* the parameterized version of the Algorithm *A_Random_MSS*, which starts by initializing ψ with the sentence ψ_{init}. We can now give the general algorithm for finding all most specific sentences.

Algorithm All_MSS Finding all most specific sentences in Th.
k_1 and k_2 are input parameters.

1. Run Algorithm *A_Random_MSS("true")* k_1 times and let S be the set of most specific sentences found.
2. While new most specific sentences are found:
   (a) Compute the set X of all *min-ortho-elements* with respect to S.
   (b) For each sentence $x \in X$:
   Run Algorithm *A_Random_MSS(x)*, k_2 times and add any new most specific sentence found to S.
3. Output S.

Theorem 2 *Given a database* $\mathbf{r}$*, a language* $\mathcal{L}$*, a partial order* $\preceq$*, and a frequency criterion* q*, algorithm All_MSS computes* $MTh(\mathcal{L}, \mathbf{r}, q, \preceq)$*.*

Proof: If there exist one most specific sentence, Step 1 results in a nonempty collection of most specific sentences. Let S be the collection of sentences, which are output in Step 3 at the end of the algorithm. Since the algorithm exited the while loop, it must be that in the last iteration of the while loop, S remained unchanged. By Lemma 1, if there is a most specific sentence γ not in S, there exist $x \in X$ such that $x \preceq \gamma$. So the algorithm *A_Random_MFS(x)* must have found at least one new maximal frequent set in the last iteration, a contradiction. □

Since the running time of the algorithm depends on k_1, k_2, these values must be chosen suitably depending on the application of the algorithm.

Computing the min-ortho-elements is in general computationally non-trivial. In the problems we consider the problem reduces to the problem of computing the minimal traversals of a hypergraph, a problem for which no output polynomial algorithm is known [15]. We use a simple enumeration heuristic that attempts to cut down the number of sets that cannot be transversals. This algorithm is exponential in the worst case. In our experiments the randomized algorithm usually produces a good approximation to the collection MTh, so only a few iterations of the expensive transversal computation are needed. Also, note that the transversal computation does not involve the input data, but only elements of $\mathcal{L}$ instead; if the size of the input data is large, a complicated computation on $\mathcal{L}$ can still be cheaper than just reading the data once.

4 Finding frequent sets using the randomized algorithm

In this section, we discuss how to adapt the algorithms of the previous section to find maximal frequent sets of a {0,1} matrix and threshold value σ. We first discuss association rules and how frequent sets arise in computation of association rules.

Given a 0/1 relation $\mathbf{r}$ with attributes R (that is, a {0,1} matrix with $|R|$ columns), an association rule is an expression $X \Rightarrow B$, where $X \subseteq R$ and $B \in R$. The intuitive meaning of such a rule is that if a row has a 1 in all attributes of X, then it tends also to have a 1 in column B.

An association rule has two values *support* and *confidence* associated with it, which are defined as follows. Given a set X of attributes of a relation $\mathbf{r}$, *frequency* f(X,r) of X in $\mathbf{r}$ is the number of rows in $\mathbf{r}$ for which all attributes in X have a 1. The *support* of X in $\mathbf{r}$ is the fraction of these rows among all the rows of $\mathbf{r}$. Given a rule $X \Rightarrow B$, the support of the rule is defined to be the support of $X \cup \{B\}$. The confidence of the rule is the fraction $f(X \cup \{B\}, \mathbf{r})/f(X, \mathbf{r})$.

The problem of mining association rules is to compute all association rules in a 0-1 relation such that the support of a rule is at least σ and the confidence at least γ. The first step in computing such association rules is to find all subsets of attributes (columns), whose support is at least σ. Such subsets are called the

frequent sets of the relation $\mathbf{r}$ with threshold σ (or σ-frequent sets). A *maximal σ-frequent set* X of relation $\mathbf{r}$ is an σ-frequent set of $\mathbf{r}$ such that no proper superset of X is an σ-frequent set of $\mathbf{r}$. The collection of all σ-frequent sets (resp. maximal σ-frequent sets) of relation $\mathbf{r}$ is denoted by $Fr(\mathbf{r}, \sigma)$ (resp. $MFr(\mathbf{r}, \sigma)$). Note that to identify all σ-frequent sets, it is enough to compute $MFr(\mathbf{r}, \sigma)$ because every frequent set is a subset of some maximal frequent set and conversely every subset of a maximal frequent set is a frequent set.

The computational problem that we study in this section is the following.

Problem 3 *Given a 0/1 relation $\mathbf{r}$ over attributes R, and a support value $\sigma \in [0, 1]$, find all maximal σ-frequent sets of $\mathbf{r}$.*

We start by presenting two results which show the computational hardness of the above problem.

Theorem 4 *The problem of finding the number of σ-frequent sets of a given 0-1 relation $\mathbf{r}$ and a threshold $\sigma \in [0, 1]$ is #P-hard. In addition, the problem of deciding if there is a maximal σ-frequent set with at least t attributes for a given 0/1 relation $\mathbf{r}$, and a threshold $\sigma \in [0, 1]$, is NP-complete.*

Proof: Finding the number of σ-frequent sets is shown to be #P-hard by reducing the problem of computing the number of satisfying assignments of a monotone-2CNF (shown to be #P-hard by [28]) to it.

Deciding if there is a maximal σ-frequent set with at least t attributes can be shown to be NP-complete by reducing the Balanced Bipartite Clique problem (known to be NP-hard, [12]), to it. □

This theorem rules out the possibility of an efficient algorithm which outputs the maximal frequent sets in the decreasing order of their size.

We now discuss a refinement of the algorithm of Section 3 for computing all maximal frequent sets. To use the framework of Section 3, we define $\mathcal{L} = \{X \mid X \subseteq R\}$, and let $q(\mathbf{r}, X)$ be true iff $s(X, \mathbf{r}) \geq \sigma$. Next, the relation $\preceq$ is defined by $X \preceq Y$ iff $X \subseteq Y$; it is easy to see that the monotonicity condition holds. We also have $X \preceq_1 Y$ iff $Y = X \cup \{A\}$ for some $A \in R$. A most specific sentence corresponds to a maximal frequent set.

A useful way to think about the maximal σ-frequent sets problem is the lattice that is formed by the subsets of R. The level i of the lattice includes all subsets of size i, and two subsets are connected if they are on consecutive levels and one is the subset of the other (see Figure 1.) Note that the collection of all maximal σ-frequent sets of a matrix and threshold σ, is an ideal of the boolean lattice over the set of columns (attributes) of the matrix. The lattice view makes also the drawbacks of the level-wise algorithm evident: it can be that Fr is large, but MFr is quite small.

To apply the general algorithm *A_Random_MSS*, we can use the lattice structure efficiently: the process can be seen as a random walk in the lattice. Given X, in order to select a random sentence/set Y such that $X \preceq_1 Y$ the only thing we have to do is to get a random element $A \in R \setminus X$ and let $Y = X \cup \{A\}$.

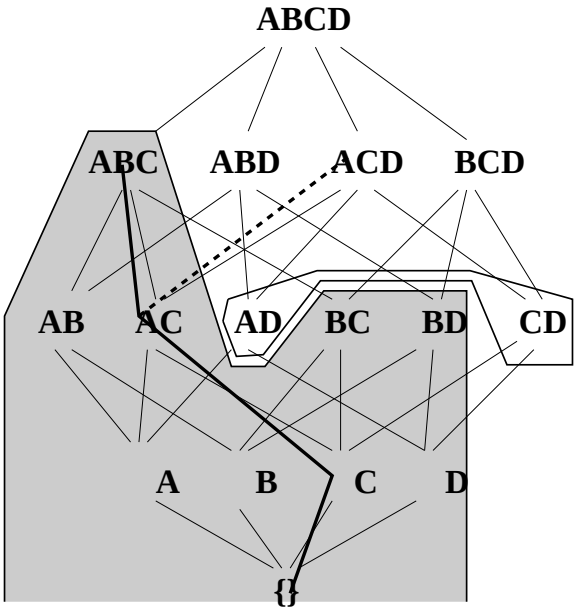

Fig. 1. A relation with four attributes. The shaded area represents the maximal σ-frequent sets found so far ($\{A, B, C\}, \{B, D\}$) and their subsets. The solid line represents a run of the algorithm for the permutation C, A, D, B. Any new maximal σ-frequent set must be a superset of $\{A, D\}$ or $\{C, D\}$.

Once a collection of maximal σ-frequent sets is found, any new maximal σ-frequent set cannot be subset or superset of a known maximal σ-frequent set. It follows that any new maximal σ-frequent set must include a set of attributes that is not a subset of any of the maximal σ-frequent sets found so far. Similarly for any new maximal σ-frequent set, there must be set of attributes such that it intersects every known maximal σ-frequent set and does not intersect the the new maximal σ-frequent set. We can express these conditions more succinctly using the concept of *minimal transversal* of a hypergraph. A *minimal* transversal of a hypergraph is a transversal of the hypergraph with the property that no proper subset of it is also a transversal. Therefore, if we view a given collection C of maximal σ-frequent sets as a hypergraph, then any new maximal σ-frequent set F is a transversal of the hypergraph whose edges are the complements of the σ-frequent sets in C. In fact the following lemma is the equivalent of Lemma 1.

Lemma 5 *Let C be a collection of maximal σ-frequent sets of a relation, and F be a maximal σ-frequent set not in C. Then there exists a minimal transversal T of the hypergraph defined by the complements of the sets in C such that $T \subseteq F$.*

Therefore to discover a new maximal σ-frequent set, we have to start with a minimal transversal of the above type and extend it to a maximal σ-frequent set. We can now present the algorithm in detail. First we give the algorithm

A_Random_MFS, which finds a single random maximal σ-frequent set containing a given set S of attributes. This algorithm corresponds to the parameterized version of the algorithm *A_Random_MSS*.

Algorithm A_Random_MFS(S) Given a {0,1} matrix M with attributes $R = \{A_1, \ldots, A_{|R|}\}$ and n tuples (rows), a threshold σ and the set S of attributes $\{A_{S_1}, ..., A_{S_l}\}$; find a maximal σ-frequent set F containing all the attributes in S.

1. Find a permutation p of $(1, \ldots, |R|)$ such that for $i \leq |S|$, $p(i) = S_i$, and for $i > |S|$, p is a random permutation of the attributes in the set $R \setminus S$.
2. Set $X = \emptyset$.
3. For $i = 1$ to $|R|$:
   (a) If $X \cup \{A_{p(i)}\}$ is a σ-frequent set, add $A_{p(i)}$ to X.
4. Return X

The following theorem shows the basic properties of the algorithm.

Theorem 6 *Let S be a σ-frequent set of relation $\mathbf{r}$. Then the algorithm A_Random_MFS(S) finds the lexicographically first (according to the ordering given by p) maximal σ-frequent set that contains the attributes in S. Its running time complexity is $O(|\mathbf{r}|)$.*

Proof: The basic operation of the algorithm is to add a new attribute in the σ-frequent set X. We keep the set of rows $\alpha(X, \mathbf{r})$ that support X as a vector $s = (s_1, \ldots, s_m)$. When attribute R_i is considered, we take the intersection of s and the i-th column of $\mathbf{r}$. This is the support of the set $X \cup R_i$. This process takes $O(m)$ time, so the total running time of the algorithm is $O(m|R|) = O(|\mathbf{r}|)$, linear to the size of the relation $\mathbf{r}$.

Note that with respect to a given permutation, a maximal frequent set F_1 is lexicographically smaller than another maximal frequent set F_2, if the smallest attribute (w.r.t. the order of attributes defined using permutation) in the symmetric difference of F_1 and F_2 is in F_1.

It is clear that the output set X is a maximal σ-frequent set. Assume that it is not the lexicographically first maximal σ-frequent set with respect to the ordering p that contains S. S has to be a frequent set itself and all the attributes of S are in the beginning of p. So all will be included in X in Step 3. Thereafter, the algorithm will add greedily into X attributes in the order given by p. Let $LF = \{R_{LF_1}, \ldots, R_{LF_k}\}$ be the lexicographically first maximal σ-frequent set with the attributes sorted according to p, and let P_i be the first attribute that is included to LF but not X. But the set $\{R_{LF_1}, \ldots, R_i\}$ is a frequent set, and therefore the algorithm would add P_i to X when it was considered. It follows that at the end of the algorithm F will represent the lexicographically smallest maximal σ-frequent set containing S. □

The complete algorithm uses algorithm *A_Random_MFS*, and finds all maximal frequent sets. After finding some of the maximal frequent sets, it computes all the minimal transversals of the hypergraph as defined in the lemma, to focus

the search on undiscovered maximal frequent sets. In the algorithm below, we omit the details of how transversals are computed.

Algorithm All_MFS Given a {0,1} relation $\mathbf{r}$ in the form of an $n \times m$ matrix M and a threshold σ, find all maximal σ-frequent sets. Parameters k_1, k_2 are positive integers.

1. Preprocess the matrix to remove all the columns (i.e. attributes) in which the number of 1's is less than σn.
2. Run algorithm *A_Random_MFS(ϕ)* k_1 times and let C be the set of maximal σ-frequent sets discovered in these runs.
3. While new maximal frequent sets are found:
   (a) Compute the set X of all minimal transversals of the hypergraph defined by complements of sets in C.
   (b) For each $x \in X$:
   Run algorithm *A_Random_MFS(x)* k_2 times and add any new maximal frequent set found to C.
4. Output S.

The following theorem is a corollary of Algorithm *All_MFS* and Theorem 2.

Theorem 7 *Given a 0/1 matrix M, and a threshold value σ, algorithm All_MFS finds all maximal σ-frequent sets of the input matrix M.*

We have implemented the algorithm *A_Random_MFS*, and we used the implementation to find maximal frequent sets in real data sets taken from the University of Helsinki. In these data sets each column represents a course offered, and the rows represent students. A given column has a 1 for each student that took this course and 0 for the rest. We have used two different threshold values, and we try to determine the rate at which the probabilistic algorithm finds new maximal frequent sets. We compare our results with the output of the level-wise algorithm ([1, 2]). These data sets are quite small for any serious data mining application. Our aim is not so much to claim that the randomized method would be a good way of computing maximal σ-frequent sets; rather we want to check the feasibility of the approach.

The preliminary results of our experiments are summarized in the following tables. In the first table we present the runs of the randomized algorithm. The two datasets have sizes of 2670×20 and 2836×129 respectively, and the threshold value σ was set to 100 and 400 rows. We run the algorithm for 500 to 4000 times before collecting the different maximal σ-frequent sets that had been found so far. The number of different maximal σ-frequent sets found is shown in the column $MFSs$ found. The next column shows the total number of maximal σ-frequent sets, as reported by the level-wise algorithm. In the second table we tabulate the results of the level-wise algorithm runs on the same datasets.

Matrix Size	σ	Runs	$MFSs$ found	$MFSs$ present	Time (sec)
2670×20	100	500	78	93	15
2670×20	100	1000	86	93	26
2670×20	100	2000	88	93	50
2670×20	100	4000	89	93	99
2670×20	400	100	23	23	5
2670×20	400	500	23	23	14
2836×129	100	500	178	315	56
2836×129	100	1000	244	315	108
2836×129	100	2000	283	315	208
2836×129	100	4000	303	315	409
2836×129	400	100	27	27	18
2836×129	400	500	27	27	64

Matrix Size	σ	$MFSs$ found	Time (sec)
2670×20	100	93	355
2670×20	400	23	6
2836×129	100	315	1512
2836×129	400	27	10

The implementation of our algorithm is in C++, and the running times for both algorithms were measured on a SPARCstation 5.

The experiments show that the randomized algorithm finds a big fraction of all the maximal frequent sets while the number of iterations is only about 5 times the total number of maximal frequent sets. In addition the randomized algorithm clearly outperforms the level-wise algorithm as long as the size of the maximal σ-frequent sets is relatively large. In our datasets this happens for a threshold value of 100.

However, as the number of iterations increase the number of discovered maximal frequent sets does not increase in proportion, but "levels off". By observing the datasets, we also noticed that the level-wise algorithm performs equivalently or slightly better when the size of maximal frequent sets are small.

Our observations suggest that by increasing the number of runs to a very large number, the advantage of finding more MFS is lost in the increase of the running time. So a better alternative can be to run the randomized algorithm a fixed number of times and then use the transversal computation to focus the search. We have implemented the algorithm for transversal computation and included it in an implementation of the algorithm *All_MFS*. We are currently testing it with more examples to see how much the computation of transversals help in speeding up the computation of all maximal frequent sets.

5 Finding Minimal Keys in Databases

In this section we discuss the computational problem of finding all minimal keys of a relational database and propose an algorithm for the problem based on the general algorithm of Section 3. We begin by defining what we mean by keys and describe an application in which it is useful to find all minimal keys. Let R be a relation schema, and $\mathbf{r}$ a relation over R. Then a set $X \subseteq R$ is a *key* of $\mathbf{r}$, if no two rows of $\mathbf{r}$ agree on every attribute in X. A *minimal key* is a key such that no proper subset of it is a key. Note that every key must contain some minimal key

and conversely every superset of a minimal key is a key. Therefore the collection of all minimal keys of a relational database is a succinct representation of the set of all keys of the database. The computational problem that we consider here is the following:

Problem 8 *Given a relational database, compute all minimal keys that exist currently.*

The knowledge of all minimal keys in the relation instance can help in semantic query optimization. In this process, a database manager substitutes a computationally expensive query by a semantically equivalent query which can be processed much faster ([4]).

However, the problem of finding the minimal keys turns out to be a difficult one.

Theorem 9 *The problem of finding the number of minimal keys of a relation* **r** *is #P-hard.*

Proof: We present two polynomial time reductions. The first reduction is from the problem of computing the number of minimal vertex covers of a graph to the problem of computing the number of minimal set covers of a family of sets. The second reduction is from the problem of computing the number of minimal set covers of a family of sets to the problem of computing the number of minimal keys if the database. Since the problem of computing the number of minimal vertex covers of a graph is known to be #P hard [28], the theorem follows.

Recall that a vertex cover of a graph G is a set of vertices of G such that every edge of G is incident on at least one vertex in the set. Given a graph G with n vertices and m edges, define a family of sets $S_1, ..., S_n$ each of which is a subset of the set $\{1, 2, ..., m\}$, as follows. The set S_i has element j iff the j^{th} edge of the graph is incident on the i^{th} vertex. Note that a collection of sets $S_{i_1}, ..., S_{i_c}$ (for some c) from the family is a minimal set cover iff the set of vertices $\{i_1, ..., i_c\}$ is a minimal vertex cover of G. Therefore the number of minimal vertex covers of G is same as the number of minimal set covers of the family. This completes the first reduction.

We now discuss the second reduction. Given a family of sets $S_1, ..., S_n$ each of which is a subset of the universe set $\{1, 2, ..., m\}$, construct a relational database as follows. The database has m fields $f_1, ..., f_m$ and $n+1$ records $R_0, ..., R_n$. The record R_0 will have value 0 in every field. For $1 \leq i \leq n$ and $1 \leq j \leq m$, the field f_j of record R_i will have value i if element i is present in the set S_j otherwise it will have value 0. A collection $S_{i_1}, ..., S_{i_c}$ is a *minimal* set cover of the family iff the set of fields $\{f_{i_1}, ..., f_{i_c}\}$ is a *minimal* key of the database. Therefore the number of minimal set covers of the family is same as the number of minimal keys of the database. □

We now discuss our algorithm for discovering all minimal keys of a relation. To keep an analogy with the problem of discovering maximal frequent sets, we will use the notion of an *anti-key*. An *anti-key* in a relation is a set of fields which is complement of some key of the relational database. A maximal anti-key

is an anti-key such that no proper superset of it is an anti-key. Note that a set of fields is a maximal anti-key iff its complement is a minimal key. Therefore the problem of finding all minimal keys of a relation is equivalent to the problem of finding all maximal anti-keys of the relation. Note that the collection of all maximal anti-keys of a relation forms an ideal of the boolean lattice over the fields of the relation.

We first present an algorithm for finding a random maximal anti-key containing a given set of fields.

Algorithm A_Random_MAS(S) Given a relation $\mathbf{r}$ in the form of an $n \times m$ matrix M and a set $S = \{f_{i_1}, ..., f_{i_s}\}$ of s fields of the relation, find a random maximal anti-key which contains all the fields in the set, provided there exists one.

1. Find a permutation p of $(1, \ldots, m)$ such that for $i \leq s$, $p(i) = i_s$, and for $i > s$, p is a random permutation of the numbers $\{s+1, s+2, \ldots m\}$.
2. Set $A = \emptyset$.
3. For $i = 1$ to m:
   (a) If $A \cup \{f_{p(i)}\}$ is an anti-key, add $f_{p(i)}$ to A.
4. Return A

Theorem 10 *Given a relation* $\mathbf{r}$ *and a set* S *of fields, suppose there exist an anti-key which contains all the fields in* S*. Then the algorithm A_Random_MAK outputs the lex-smallest maximal anti-key with respect to the random permutation and which contains all the fields in* S*. The running time of the algorithm is* $O(nm)$*, assuming that the time to access any field of any record is constant.*

We now give the complete algorithm for finding all maximal anti-keys, which is analogous to the algorithm for finding all maximal frequent sets. First we point out that an analogue of the Lemma 5 holds also for the case of maximal anti-keys.

Lemma 11 *Given a collection* C *of maximal anti-keys of a relation, let* K *be a maximal anti-key not in* C*. Then there exists a minimal transversal* T *of the hypergraph defined by the complements of the sets in* C *such that* $T \subseteq K$*.*

In the algorithm *All_MAK*, once again, we ignore the details of how to find all minimal transversals of a hypergraph.

Algorithm All_MAK Given a relational database in the form of $n \times m$ matrix M and find all maximal anti-keys. Parameters k_1, k_2, which are positive integers.

1. Run algorithm *A_Random_MAK(ϕ)* k_1 times and let C be the set of maximal anti-keys discovered in these runs.
2. While new anti-keys are being found:
   (a) Compute the set X of all minimal transversals of the hypergraph defined by complements of subsets in C.
   (b) For each $x \in X$: Run algorithm *A_Random_MAK(X)* k_2 times and add any new maximal anti-key found to C.

3. Output C.

We can now claim the following theorem.

Theorem 12 *Given a relational database* $\mathbf{r}$*, algorithm All_MAK finds all current maximal anti-keys (and hence minimal keys) of* $\mathbf{r}$*.*

In this section we present some experimental results obtained from an earlier implementation of a somewhat different algorithm to compute all keys. This algorithm uses the same general scheme, but attempts to compute minimal keys directly, instead of computing maximal anti-keys first. We implemented this algorithm, *All_K* in C++. To test the algorithm, we used two different relations with extremely complicated real data, which happen have a lot of keys, and four artificial relations, three of which have few keys and 1 has many keys. The size of the keys ranges from one to ten attributes. Parameters k_1, k_2 where set to 100.

Relation size	Number of keys present	Time (sec)
58×12	58	30
128×22	1258	6375
100×10	1	1
100×11	2	2
10000×11	2	388
11×20	1024	604

We remark that the second test case is an exceptionally complex one in terms of the size and structure of minimal keys of the input relation. We are planning more experiments to see the performance for large inputs and with respect to the level-wise algorithm.

6 Discussion

We have given a randomized algorithm for computing the representation of a theory in terms of its most specific sentences. We have also proposed an approach based on transversal computation to focus the search of the randomized algorithm on undiscovered sentences. This can be combined with the randomized algorithm to give an algorithm which can (provably) find all the most specific sentences. We have illustrated the application of the algorithm in two important data mining scenarios: computing all maximal σ-frequent sets of a 0/1 relation with threshold σ and computing all minimal keys of a relation.

We have conducted some experiments with our algorithms which indicate the benefits of using randomization over the earlier known level-wise approach. Though the preliminary results of the experiments show our algorithms to be promising, a lot more remains to be done to substantiate these promises. These include the scalability analysis and the tradeoff of transversal computation for focusing search against the level wise approach.

An issue we have not explored yet is to find a more efficient way to use transversals to guide the search. Currently we compute all the minimal transversals from scratch every time some new sentences is discovered. Some alternative strategies may result in faster algorithms. One possibility is to compute only one transversal at a time instead of all of them. Another possibility is to avoid computing the minimal transversals from scratch every time. Since the hypergraph (whose minimal transversals are computed) in a given iteration contains the hypergraphs for all previous iterations, it may be possible to compute the new set of minimal transversals by scanning the old set and dropping those which are no longer transversals and then computing only the newly formed transversals.

Another interesting possibility we plan to explore is to use the randomized algorithm in combination with the level-wise algorithm. The randomized algorithm for maximal σ-frequent sets can be used to select the right range for σ, as a preprocessing step to the level-wise algorithm of Agrawal et. al. [2].

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*, pages 207 – 216, May 1993.
2. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.
3. R. Agrawal and R. Srikant. Mining sequential patterns. In *International Conference on Data Engineering*, Mar. 1995.
4. S. Bell. Deciding distinctness of query results by discovered constraints. *Manuscript.*
5. S. Bell and P. Brockhausen. Discovery of data dependencies in relational databases. Technical Report LS-8 14, Universität Dortmund, Fachbereich Informatik, Lehrstuhl VIII, Künstliche Intelligenz, 1995.
6. C. Berge. *Hypergraphs. Combinatorics of Finite Sets.* North-Holland Publishing Company, Amsterdam, 1989.
7. C. C. Chang and H. J. Keisler. *Model Theory.* North-Holland, Amsterdam, 1973. 3rd ed., 1990.
8. L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI–93)*, pages 1058 – 1053, Chambéry, France, 1993. Morgan Kaufmann.
9. L. De Raedt and S. Džeroski. First-order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375 – 392, 1994.
10. T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278 – 1304, Dec. 1995.
11. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, Menlo Park, CA, 1996.
12. M. Garey and D. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* W.H. Freeman, New York, 1979.

13. J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 420 – 431, Zurich, Swizerland, 1995.
14. M. Houtsma and A. Swami. Set-oriented mining of association rules. Research Report RJ 9567, IBM Almaden Research Center, San Jose, California, October 1993.
15. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.
16. J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*, pages 335 – 359. Academic Press, London, 1992.
17. W. Kloesgen. Efficient discovery of interesting statements in databases. *Journal of Intelligent Information Systems*, 4(1):53 – 69, 1995.
18. A. J. Knobbe and P. W. Adriaans. Discovering foreign key relations in relational databases. In *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, pages 94 – 99, Heraklion, Crete, Greece, Apr. 1995.
19. H. Mannila. Aspects of data mining. In *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, pages 1–6, Heraklion, Crete, Greece, Apr. 1995.
20. H. Mannila. Data mining: machine learning, statistics, and databases. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management, Stockholm*, 1996. To appear.
21. H. Mannila and K.-J. Räihä. Design by example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 33(2):126 – 141, 1986.
22. H. Mannila and K.-J. Räihä. Algorithms for inferring functional dependencies. *Data & Knowledge Engineering*, 12(1):83 – 99, Feb. 1994.
23. H. Mannila and H. Toivonen. On an algorithm for finding all interesting sentences. In *Cybernetics and Systems Research '96*, Vienna, Austria, Apr. 1996. To appear.
24. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210 – 215, Montreal, Canada, Aug. 1995.
25. A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 432 – 444, Zurich, Swizerland, 1995.
26. J. Schlimmer. Using learned dependencies to automatically construct sufficient and sensible editing views. In *Knowledge Discovery in Databases, Papers from the 1993 AAAI Workshop (KDD'93)*, pages 186 – 196, Washington, D.C., 1993.
27. J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville, MD, 1988.
28. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

This article was processed using the LaTeX macro package with LLNCS style