Discovering All Most Specific Sentences

DIMITRIOS GUNOPULOS
Computer Science and Engineering Department, University of California, Riverside
RONI KHARDON
EECS Department, Tufts University, Medford, MA
HEIKKI MANNILA
Department of Computer Science, University of Helsinki, Helsinki, Finland
SANJEEV SALUJA
LSI Logic, Milpitas, CA
HANNU TOIVONEN
Department of Computer Science, University of Helsinki, Helsinki, Finland
and
RAM SEWAK SHARMA
Computer Science and Engineering Department, University of California, Riverside

Data mining can be viewed, in many instances, as the task of computing a representation of a theory of a model or a database, in particular by finding a set of maximally specific sentences satisfying some property. We prove some hardness results that rule out simple approaches to solving the problem.

The *a priori* algorithm is an algorithm that has been successfully applied to many instances of the problem. We analyze this algorithm, and prove that is optimal when the maximally specific sentences are "small". We also point out its limitations.

The work of D. Gunopulos was partially supported by National Science Foundation (NSF) CAREER Award 9984729, NSF grants IIS-9907477 and ITR 0220148, and the Department of Defense (DoD). The reseach of R. Khardon was supported by Office of Naval Research (ONR) grant N00014-95-1-0550 and ARO grant DAAL03-92-G-0115.

Authors' addresses: D. Gunopulos and R. Sewak Sharma, Computer Science and Engineering Department, University of California, Riverside, Riverside, CA 92507; email: {dg;rssharma}@cs.ucr.edu; R. Khardon, EECS Department, Tufts University, Medford, MA 02155, email: roni@eecs.tufts.edu; H. Mannila, HIIT Basic Research Unit, Department of Computer Science, University of Helsinki, Helsinki; Finland; email: Heikki.Mannila@cs.helsinki.fi; S. Saluja, LSI Logic, MS E 192, 1551 McCarthy Blvd., Milpitas, CA 95035; email: sanjeev@lsil.com; H. Toivonen, Department of Computer Science, University of Helsinki, Helsinki, Finland; email: Hannu.Toivonen@ca.helsinki.fi.

We then present a new algorithm, the Dualize and Advance algorithm, and prove worst-case complexity bounds that are favorable in the general case. Our results use the concept of hypergraph transversals. Our analysis shows that the *a priori* algorithm can solve the problem of enumerating the transversals of a hypergraph, improving on previously known results in a special case. On the other hand, using results for the general case of the hypergraph transversal enumeration problem, we can show that the Dualize and Advance algorithm has worst-case running time that is subexponential to the output size (i.e., the number of maximally specific sentences).

We further show that the problem of finding maximally specific sentences is closely related to the problem of exact learning with membership queries studied in computational learning theory.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Data mining, association rules, maximal frequent sets, learning with membership queries, minimal keys

1. INTRODUCTION

Data mining has recently emerged as an active area of investigation and applications [Fayyad et al. 1996]. The goal of data mining can briefly be stated as "development of efficient algorithms for finding useful high-level knowledge from large amounts of data." The area combines methods and tools from databases, machine learning, and statistics.

A large part of current research in data mining can be viewed as addressing instances of the following problem: given a language, an interestingness criterion, and a database, find all sentences from the language that are true in the database and satisfy the interestingness criterion. Typically, this criterion is a frequency criterion that states that there are sufficiently many instances in the database satisfying the sentence. Examples of scenarios where this formulation works include the discovery of frequent sets, association rules, strong rules, episodes, and keys. In this article, we show how the problems of finding frequent sets in relations and of finding minimal keys in databases can be reduced to this formulation. Using this *theory extraction formulation* [Mannila 1995, 1996; Mannila and Toivonen 1997], one can formulate general results about the complexity of algorithms for these data mining tasks.

The specific problem we are considering is the complexity of computing the most specific interesting sentences. This problem has known lower bound results, however existing algorithms have running times significantly worse than the best known lower-bounds. We analyze the running time of one of the most successful data mining algorithms, *a priori*, that has been applied to that problem. We then give a new algorithm, Dualize and Advance, that is designed to find the most specific sentences only.

Several variations of the *a priori* algorithm have been successfully applied to problems of data mining [Agrawal and Srikant 1994; Agrawal et al. 1996; Mannila and Toivonen 1997; Mannila et al. 1994, 1995]. The *a priori* algorithm computes the interesting sentences by walking up in the lattice of sentences, one level at a time. Thus, it operates 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, one level at a time. To concentrate on its operation, we refer to *a priori* algorithm as the *level-wise* algorithm. We show that as long as the number of levels in the search is small this algorithm is indeed optimal thus explaining its empirical success and shedding some light on when and why it is useful. Furthermore, we show that this algorithm can be used to efficiently solve a special case of the hypergraph transversal problem, improving on previous theoretical results.

On the other hand, the analysis indicates that when the number of levels in the search is large, the number of sentences of interest may become too large to handle. An alternative method is to try to search for the most specific sentences from the language that satisfy the requirements: these sentences determine the theory uniquely. The number of interesting sentences can be exponential to the number of most specific interesting sentences. It is therefore likely that an algorithm that computes the most specific sentences can offer significant improvements in computation time.

For this purpose, we present the *Dualize and Advance* algorithm (first introduced in Gunopulos et al. [1997]) for locating the most specific true sentences satisfying the frequency criterion. We prove upper bounds on the complexity of the algorithm for the general case, showing that it comes close to lower bounds for the problem. Our basic algorithm is deterministic and is sufficient to provide the worst case complexity bounds. We further apply a randomized heuristic in the algorithm that can improve its running time in practice considerably. While the algorithm is randomized, it is complete, in the sense that it returns all most specific sentences, and the worst case bounds hold for it as well.

Briefly, the method works as follows. We apply a greedy search to locate some maximal elements from the language. We then use the simple fact that if some most specific sentences are known, then every unknown one must contain a *minimal transversal* of the *complements* of the known sentences. The algorithm alternates between finding 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 sentences can be found.

We show that the running time of the algorithm is sub-exponential to the size of the output. This result also shows that the complexity of the problem of computing the most specific interesting sentences is lower than the complexity of finding all the interesting sentences, thus providing theoretical support for the experimental evidence that recent heuristic algorithms [Bayardo 1998; Burdick et al. 2001; Agrawal et al. 2000; Gouda and Zaki 2001], that have been designed to find maximal frequent sets directly, can significantly outperform *a priori*.

To demonstrate the utility of the algorithm, we apply it to the problem of computing of all minimal keys, or functional dependencies, in a relational database in addition to the problem of computing of all maximal frequent sets of a {0, 1} matrix for a given threshold. The computation of maximal frequent sets is a fundamental data mining problem which is required in discovering association rules [Agrawal et al. 1993, 1996; Agrawal and Srikant 1994]. Computation of minimal keys is important for semantic query optimization, which leads to fast

query processing in database systems [Mannila and Räihä 1994; Knobbe and Adriaans 1995; Bell and Brockhausen 1995; Schlimmer 1993]. Here, we refer to possible keys that exist in a specific instance of a relational database and are not designed as such. In both cases, we first prove some hardness results of related problems ruling out simple algorithmic approaches. We then show that the algorithm can be adapted to solve these problems.

The rest of this article is organized as follows: In Section 2, we present a model of data mining that formally defines the theory extraction problem. We also show how this model can be used to describe the problems of computing frequent sets and minimal keys. We also show the correspondence between this problem and problems studied in learning theory. In Section 3, we give hardness results that show that these two specific problems are difficult to solve. In Section 4, we formally define our computational model. Section 5 presents and analyses the level-wise algorithm. Section 6 presents the Dualize and Advance algorithm and analyses its complexity. Section 7 describes how the Dualize and Advance algorithm can be adapted to compute maximal frequent sets and minimal keys. We also apply this algorithm to the problem of learning Boolean monotone functions using membership queries. Section 8 presents an incremental algorithm for computing the transversals of a hypergraph. Section 9 presents related work. In Section 10, we discuss the scope of our algorithms and point out some directions of further work. Preliminary versions of the work presented here appeared previously in [Gunopulos et al. 1997a, 1997b].

2. DATA MINING AS THEORY EXTRACTION

The model of knowledge discovery that we consider is the following [Mannila 1995, 1996; Mannila and Toivonen 1997]. 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, that is, 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 $\mathbf{r}$. We therefore abstract this away by saying that φ *is interesting* when $q(\mathbf{r}, \varphi) = 1$, and discuss the problem of mining for interesting sentences.

Obviously, if $\mathcal{L}$ is infinite and $q(\mathbf{r}, \varphi)$ is satisfied for infinitely many sentences, (an explicit representation of) all 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 article, we assume that $\mathcal{L}$ is finite.

As already considered by Mitchell [1982], we use a specialization/generalization relation between sentences. (See, e.g., Langley [1995] for an

overview of approaches to related problems.) A *specialization relation* is a partial order $\preceq$ on the sentences in $\mathcal{L}$. We say that φ is more general than θ, if $\varphi \preceq \theta$; we also say that θ is more specific than φ. The relation $\preceq$ is a *monotone specialization relation* with respect to q if the quality predicate q is monotone with respect to $\preceq$, that is, for all $\mathbf{r}$ and φ we have the following: if $q(\mathbf{r}, \varphi)$ and $\varphi' \preceq \varphi$, then $q(\mathbf{r}, \varphi')$. In other words, if a sentence φ is interesting based on the quality predicate q, then also all less special (i.e., more general) sentences $\varphi' \preceq \varphi$ are interesting. We write $\sigma \prec \tau$ if $\sigma \preceq \tau$ and not $\tau \preceq \sigma$.

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 $\theta \preceq \psi$, then $rank(\psi) = 0$; otherwise, $rank(\psi) = 1 + \max\{rank(\theta) \mid \theta \preceq \psi\}$. For $T \subset \mathcal{L}$, let T_i denote the set of the sentences of $\mathcal{L}$ with rank i.

Typically, the relation $\preceq$ is (a restriction of) the semantic implication relation: if $\sigma \preceq \tau$, then $\tau \models \sigma$, that is, for all databases $\mathbf{r}$, if $\mathbf{r} \models \tau$, then $\mathbf{r} \models \sigma$. Note that if the interestingness predicate q is defined in terms of statistical significance or something similar, then the semantic implication relation is not a monotone specialization relation with respect to q: a more specific statement can be interesting, even when a general statement is not.

Given a specialization relation $\preceq$, the set $Th(\mathcal{L}, \mathbf{r}, q)$ can be represented by enumerating only its maximal elements, that is, the set

$$MTh(\mathcal{L}, \mathbf{r}, q) = \{\phi \in Th(\mathcal{L}, \mathbf{r}, q) \mid \text{for no } \theta \in Th(\mathcal{L}, \mathbf{r}, q)\ \phi \prec \theta\}.$$

Here again, one should be careful when working with infinite lattices. We assume throughout the article that the maximal elements exist and are well defined, and similarly for the minimal elements outside the theory $Th(\mathcal{L}, \mathbf{r}, q)$. This definitely holds in finite lattices, and can be useful in more general cases as well. The problem considered in this article is therefore the following:

Problem 1 [*MaxTh*]. Given $\mathcal{L}$, $\mathbf{r}$, and q, find $MTh(\mathcal{L}, \mathbf{r}, q)$.

It is easy to show ([Mannila and Toivonen 1997]) that finding frequent sets, episodes, keys, or inclusion dependencies are instances of the problem MaxTh. Especially for the problem of finding keys (or, more generally, functional dependencies) from relation instances the current framework has lots of connections to previous work.

2.1 Association Rules and Frequent Sets

To facilitate the presentation, we next discuss the problem of computing frequent sets that will serve to illustrate ideas in the next sections.

Given a 0/1 relation $\mathbf{r}$ with attributes R, an association rule is an expression $X \Rightarrow A$, where $X \subseteq R$ and $A \in R$. The intuitive meaning of such a rule is that, if a row has 1 in all attributes of X, then it tends also to have 1 in column A. Typically, in data mining, association rules are searched so that the set of rows having 1 in the attributes in $X \cup A$ is large enough; if we were to draw random rows from $\mathbf{r}$, it is required that such rows will be drawn with frequency at least σ, for some fixed σ. The actual frequency is called the *support* of the rule. The ratio of rows including 1 in $X \cup A$ to those including 1 in the set X is called the *confidence* of the rule.

Given the above description, a major subtask that is usually solved first is that of computing frequent sets. Namely, given a 0/1 relation $\mathbf{r}$, compute all subsets Z such that the frequency of rows having 1 in all attributes of Z is larger than σ. Clearly, this is an instance of the problem discussed above; $\mathcal{L}$ is the set of subsets of R, and q corresponds to having frequency higher than σ. The set $Th(\mathcal{L}, \mathbf{r}, q)$ corresponds to the set of frequent sets, and similarly we can talk of maximal frequent sets. This leads to the definition of the first specific instance of the problem we are considering:

Problem 2 (*Finding Maximal Frequent Sets*). Given a 0/1 relation, and a threshold σ compute all maximal frequent sets.

Once the frequent sets are found, the problem of computing association rules from them is straightforward. For each frequent set Z, and for each $A \in Z$, one can test the confidence of the rule $Z \setminus A \Rightarrow A$.

2.2 Finding Minimal Keys in Databases

In this section, we discuss the problem of finding all minimal keys of a database. We begin by defining what we mean by keys and describe an application in which it is useful to find all minimal keys. We view a relational database r as a matrix whose columns correspond to fields and rows correspond to records. Let R denote the set of all fields (i.e., columns of the matrix). Then, a set $X \subseteq R$ is a *key* of r, if no two rows of 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 database is a succinct representation of the set of all keys of the database. Note the distinction between our definition of key and the more standard definition of (primary) key of a database [Ullman 1988]. A (primary) key is a key (with respect to our definition) of the database throughout the life of the database and is maintained so by the database manager. However, an arbitrary key, by our definition, may be so at current state of the database and may not exist to be so after an update of the database. The problem we consider here is the following:

Problem 3 (*Finding Keys*). Given a relational database, compute all minimal keys that exist currently.

As has been discussed in Bell [2003], the knowledge of all minimal keys existing currently in the database can help in semantic query optimization that is, in the process by which a database manager substitutes a computationally expensive query by a semantically equivalent query that can be processed much faster.

2.3 Relation to Learning Theory

We now show that the problem discussed above is very closely related to problems in learning theory. One of the scenarios discussed in learning theory is

as follows: a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ is fixed by some adversary (modeling a concept in the world). A learner is given access to some oracle giving it partial information on the function f. The task of the learner is to find a representation for a Boolean function that is identical (or approximates) f. In particular, we consider the model of exact learning with membership queries [Angluin 1988].

A membership query oracle $MQ(f)$ allows the learner to ask for the value of f on a certain point. That is, given $x \in \{0,1\}^n$, $MQ(f)$ returns the value $f(x)$. The learning algorithm is given access to $MQ(f)$, and the algorithm is required to produce an expression that computes f exactly.

Definition 4. An algorithm is an exact learning algorithm with time complexity $T()$, query complexity $Q()$, and representation class $\mathcal{H}$, for a class of functions $\mathcal{F}$, if for all $f \in \mathcal{F}$, when given access to $MQ(f)$, the algorithm runs in time $T()$, asks MQ on at most $Q()$ points and then outputs a representation $h \in \mathcal{H}$ for a Boolean function such that h is equivalent to f.

In the above definition, we omitted the parameters of the functions $T()$ and $Q()$. Normally, the algorithm is allowed time polynomial in the number of variables n, and the size of a representation for f in some representation language.

In particular, we next consider the problem of learning *monotone* functions with membership queries. A function f is monotone if $f(x) = 1$, and $y \geq x$ implies $f(y) = 1$, where $\leq$ is the normal partial order on $\{0,1\}^n$. We also consider the standard CNF and DNF representations for such functions. A term is a conjunction of literals, for example, x_1x_2 is a term. A DNF expression is a disjunction of terms, for example, $x_1x_2 \vee x_2x_3$ is a DNF expression. Similarly, a CNF expression is a conjunction of disjunctions, for example, $(x_1 \vee x_2)(x_2 \vee x_3)$ is a CNF expression. It is well known that monotone functions have unique minimum size representations in both DNF and CNF, that include all minimal terms or clauses respectively of the function. (A minimal term, called a *prime implicant*, is a term that implies f and such that every subset of it does not imply f.)

In the scenario that follows, the learning algorithm is allowed time relative to the number of attributes n, and the sum of sizes of its DNF and CNF representations. That is, we consider $T(m)$, and $Q(m)$ where $m = n + |DNF(f)| + |CNF(f)|$.

The correspondence between learning monotone functions and computing interesting sets is thus straightforward. The elements of $\{0,1\}^n$ correspond to subsets of the variables so that a value 1 implies that the corresponding attribute is in the set. The value of the function on an assignment corresponds to the negation of the interestingness relation q. Since q is monotone, the function is monotone. Membership queries now naturally correspond to Is-interesting queries. We therefore get:

THEOREM 5. *The problem of computing interesting sentences for problems representable as sets is equivalent to the problem of learning monotone functions with membership queries, with representation class CNF* (*or DNF*).

3. HARDNESS RESULTS ON THE COMPUTATION OF FREQUENT SETS AND MINIMAL KEYS

Computing frequent sets or maximal frequent sets is an enumeration problem. The algorithm must enumerate all sets and in the end provide proof that no more sets exist. The results in this section indicate that it is difficult not only to find all maximal frequent sets and minimal keys, but it is also difficult to verify that all maximal sets or keys have already been found. The hardness results we present also show that it is difficult to to find the number of frequent sets or keys. As a result, algorithms that find frequent sets or keys are likely to be worst case exponential. Consequently, we use an output-size sensitive complexity model to evaluate the performance of the algorithms.

3.1 Hardness Results on the Computation of Frequent Sets

We first consider the problem of counting the number of σ-frequent sets.

THEOREM 6. *The problem of finding the number of σ-frequent sets of a given $0-1$ relation r and a threshold $\sigma \in [0, 1]$ is #P-hard.*

PROOF. We show a polynomial time reduction from the problem of computing the number of satisfying assignments of a monotone-2CNF formula to the problem of computing the number of frequent sets, that has a simple mapping between the number of solutions. This suffices, since the problem of computing the number of satisfying assignment of monotone-2CNF formulas is known to be #P-hard [Valiant 1979].

A monotone-2CNF formula is a Boolean formula in conjunctive normal form in which every clause has at most two literals and every literal is unnegated. Given a monotone-2CNF formula f with m clauses and n variables, construct an $m \times n$ $\{0, 1\}$ matrix M as follows: $M_{j,i}$ is 0 if the ith variable is present in the jth clause and 1 otherwise. An assignment of variables falsifies f, iff the set of columns corresponding to variables with value 1 forms a frequent set of M with threshold $\frac{1}{m}$. Therefore, the number of frequent sets of M with threshold $\frac{1}{m}$ is (2^n—the number of satisfying assignments of f). This completes the reduction. □

Note that the above result still does not rule out the possibility of an output polynomial algorithm for computing all maximal frequent sets, since in contrast with counting, for enumeration one is given time polynomial in the size of the output. The next theorem rules out the possibility of an efficient algorithm that outputs the maximal frequent sets in the decreasing order of their size.

THEOREM 7. *The problem of deciding if there is a maximal σ-frequent set with at least t attributes for a given $0-1$ relation r, and a threshold $\sigma \in [0, 1]$, is NP-complete.*

PROOF. It is easily seen that the problem is in NP. To show the NP-hardness, we show a polynomial time reduction from the Balanced Bipartite Clique problem to the above problem. Since the Balanced Bipartite Clique is known to be NP-hard, the result will follow [Garey and Johnson 1979].

Given a bipartite graph $G = (V_1, V_2, E)$, a balanced clique of size k is a complete bipartite graph with exactly k vertices from each of V_1 and V_2. The Balanced Bipartite Clique problem is, given a bipartite graph G and a positive integer k, check if there exists a balanced bipartite clique of size k.

Given a bipartite graph G and a positive integer k, let n_1 and n_2 be the number of vertices in V_1 and V_2 respectively. Define an $n_1 \times n_2$ {0, 1} matrix M as follows. $M_{i,j}$ is 1 iff ith vertex of V_1 is connected to the jth of V_2. Then, there is a bipartite clique of size k in G iff there is a frequent set of M of size at least k with threshold $\frac{k}{n_1}$. □

3.2 Hardness Results for Computing Minimal Keys

THEOREM 8. *The problem of finding the number of all keys of a given database is #P-hard.*

PROOF. We prove the result in two steps. First, we show a polynomial-time reduction from the problem of computing the number of satisfying assignments of a monotone-2CNF formula to the problem of computing the number of set-covers of a family of sets. Then, we show a polynomial time reduction from the problem of computing the number of set covers of family of sets to the problem of computing the number of keys of a database. Since the reductions maintain the same number of solutions, and the problem of computing the number of satisfying assignments of a monotone-2CNF formula is #P-hard [Valiant 1979], this proves the result.

Recall that, given a family of sets each of which is subset of a finite *universe* set, a *set cover* is a collection of sets from the family such that every element of the universe is in at least one of the sets in the collection. Given a monotone-2CNF formula with m clauses and n variables, construct a family of n sets $S_1, \ldots, S_n$ each of which is a subset of the set $\{1, 2, \ldots, m\}$, as follows. The set S_i contains j iff ith variable is present in jth clause. It is easily seen that a satisfying assignment of the monotone-2CNF formula corresponds to a unique set cover of the family of sets and vice-versa, by picking S_i in the set cover iff the ith variable has value 1 in the assignment. Therefore, the number of satisfying assignments of the monotone-2CNF formula is exactly the number of set covers of the family of sets. This completes the first reduction.

We now discuss the second reduction. Given a family of sets $S_1, \ldots, S_n$ each of which is a subset of the universe set $\{1, 2, \ldots, m\}$, construct a relational database as follows: The database has n fields $f_1, \ldots, f_n$ and $m + 1$ records $r_0, \ldots, r_m$. The record r_0 will have value 0 in every field. For $1 \leq j \leq n$ and $1 \leq i \leq m$, 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. Note that a collection $S_{i_1}, S_{i_2}, \ldots, S_{i_c}$ (for some c) of sets from the family will be a set cover iff the collection of fields $f_{i_1}, \ldots, f_{i_c}$ is a key of the database. Therefore, the number of set covers of the given family of sets is the same as the number of keys of the database. This completes the second reduction and the proof of the theorem. □

The following theorem shows that counting the number of minimal keys is not easier than counting the number of all keys.

THEOREM 9. *The problem of finding the number of minimal keys of a given database is #P-hard.*

PROOF. Once again, we show two polynomial time reductions that maintain the same number of solutions. 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 of the database. Since the problem of computing the number of minimal vertex covers of a graph is known to be #P hard [Valiant 1979], this implies the result.

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 $S_1, \ldots, S_n$ each of which is a subset of the set $\{1, 2, \ldots, m\}$, as follows. The set S_i has element j iff the jth edge of the graph is incident on the ith vertex. Note that a collection of sets $S_{i_1}, \ldots, S_{i_c}$ (for some c) from the family is a minimal set cover iff the set of vertices $\{i_1, \ldots, i_c\}$ 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.

For the second reduction, we use the same reduction that was used as second reduction in the proof of Theorem 8. With respect to the reduction, note that a collection $S_{i_1}, \ldots, S_{i_c}$ is a *minimal* set cover of the family iff the set of fields $\{f_{i_1}, \ldots, f_{i_c}\}$ 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. □

4. COMPLEXITY OF FINDING ALL INTERESTING SENTENCES

The hardness results we presented in the previous section show that algorithms that find maximal frequent sets or keys are likely to have exponential worst case running time. Consequently, we use an output-size sensitive complexity model to evaluate the performance of the algorithms.

To study the complexity of the generation problem, we introduce some notation and basic results that appeared previously in Mannila and Toivonen [1997].

Consider a set S of sentences from $\mathcal{L}$ such that S is closed downwards under the relation $\preceq$, that is, if $\theta \in S$ and $\varphi \preceq \theta$, then $\varphi \in S$. The *border* $Bd(S)$ of S consists of those sentences σ such that all generalizations of σ are in S and none of the specializations of σ is in S. Those sentences σ in $Bd(S)$ that are in S are called the *positive border*[1] $Bd^+(S)$, and those sentences σ in $Bd(S)$ that are not in S are the *negative border* $Bd^-(S)$. In other words,

$$Bd(S) = Bd^+(S) \cup Bd^-(S),$$

where

$$Bd^+(S) = \{\sigma \in S \mid \text{ for all } \gamma \text{ such that } \sigma \prec \gamma, \text{ we have } \gamma \notin S\}$$

[1]That is, the positive border corresponds to the set "S" of [Mitchell 1982].

and

$$Bd^-(S) = \{\sigma \in \mathcal{L} \setminus S \mid \text{ for all } \gamma \prec \sigma, \text{ we have } \gamma \in S\}.$$

The positive border of the theory is the set of its maximal elements, that is, $MTh(\mathcal{L}, \mathbf{r}, q) = Bd^+(Th(\mathcal{L}, \mathbf{r}, q))$. Note that $Bd(S)$ can be small even for large S.

Above, we assumed that the set S is closed downwards. We generalize the notation for sets S that are not closed downwards by simply defining that $Bd(S) = Bd(S')$ where S' is the downward closure of S. The generalization is similar for negative and positive borders.

Some straightforward lower bounds for the problem of finding all frequent sets are given in Agrawal et al. [1996] and Mannila et al. [1994]. Now we consider the problem of lower bounds in a more realistic model of computation.

The main effort in finding interesting sets is in the step where the interestingness of subgroups are evaluated against the database. Thus, we consider the following model of computation. Assume the only way of getting information from the database is by asking questions of the form

Is-interesting Is the sentence φ interesting, that is, does $q(\mathbf{r}, \varphi)$ hold?

THEOREM 10 [MANNILA AND TOIVONEN 1997]. *Any algorithm for computing $Th(\mathcal{L}, \mathbf{r}, q)$ that accesses the data using only Is-interesting queries must use at least $|Bd(Th(\mathcal{L}, \mathbf{r}, q))|$ queries.*

This result, simple as it seems, gives as a corollary a result about finding functional dependencies that in the more specific setting is not easy to find; cf. Mannila and Räihä [1992] and Mannila and Toivonen [1997]. Similarly, the corresponding verification problem requires at least this number of queries.

Problem 11 (*Verification*). Given $\mathcal{L}$, $\mathbf{r}$, q, and a set $S \subseteq \mathcal{L}$, verify that $S = MTh(\mathcal{L}, \mathbf{r}, q)$.

COROLLARY 12 [MANNILA AND TOIVONEN 1997]. *Given $\mathcal{L}$, $\mathbf{r}$, q, and a set $S \subseteq \mathcal{L}$, determining whether $S = MTh(\mathcal{L}, \mathbf{r}, q)$ requires in the worst case at least $|Bd(S)|$ evaluations of the predicate q, and it can be solved using exactly this number of evaluations of q.*

We now show that the verification problem is closely related to computing hypergraph transversals. A collection $\mathcal{H}$ of subsets of R is a (*simple*) *hypergraph*, if no element of $\mathcal{H}$ is empty and if $X, Y \in \mathcal{H}$ and $X \subseteq Y$ imply $X = Y$. The elements of $\mathcal{H}$ are called the *edges* of the hypergraph, and the elements of R are the *vertices* of the hypergraph. Given a simple hypergraph $\mathcal{H}$ on R, a *transversal* T of $\mathcal{H}$ is a subset of R intersecting all the edges of $\mathcal{H}$, that is, $T \cap E \neq \emptyset$ for all $E \in \mathcal{H}$.

Transversals are also called *hitting sets*. Here we consider *minimal transversals*: a transversal T of $\mathcal{H}$ is minimal if no $T' \subset T$ is a transversal (Figure 1). The collection of minimal transversals of $\mathcal{H}$ is denoted by $\mathrm{Tr}(\mathcal{H})$. It is a hypergraph on R.

Problem 13 (*HTR*). Given a hypergraph $\mathcal{H}$, construct $\mathrm{Tr}(\mathcal{H})$.

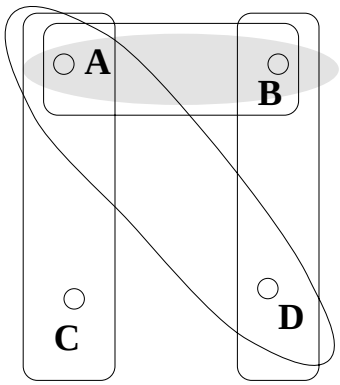

Fig. 1. The set $\{A, B\}$ is a transversal of the hypergraph with edges AB, AC, AD, BD.

For more information on hypergraphs, see [Berge 1973]. The problem of computing transversals appears in various branches of computer science; a comprehensive study of this problem is given by [Eiter and Gottlob 1995]. The HTR problem also appears in several forms in databases. In particular, the problem of translating between a set of functional dependencies and their corresponding Armstrong relation [Mannila and Räihä 1986, 1992] is at least as hard as this problem and equivalent to it in special cases [Eiter and Gottlob 1995]. Further discussion of these issues is given by Khardon [1995] and Mannila and Räihä [1994].

Notice that, in general, the output for this problem may be exponentially larger than its input, and thus the question is whether it can be solved in time polynomial in both its input size and output size. We say that an algorithm is *output* $T()$ *time* algorithm for the problem if it runs in time $T(I, O)$ where I is the input size, and O is the corresponding output size. A more strict condition, that we use here, requires that the output transversals be enumerated, and that the time to compute the ith transversal will be measured against the input size and i. That is, an algorithm solves the problem in *incremental* $T(I, i)$ *time* if the ith transversal is computed in time $T(I, i)$. For further discussion and other variations, see Eiter and Gottlob [1995].

The exact complexity of the HTR problem is yet unknown. A subexponential solution for the problem has been recently discovered [Fredman and Khachiyan 1996], and several special cases can be solved in polynomial time [Eiter and Gottlob 1995; Mishra and Pitt 1997].

Now we return to the verification problem. Given $S \subseteq \mathcal{L}$, we have to determine whether $S = MTh(\mathcal{L}, \mathbf{r}, q)$ holds using as few evaluations of the interestingness predicate as possible.

Definition 14 (*Representing as Sets*). Let $\mathcal{L}$ be the language, $\preceq$ a specialization relation, and R a set; denote by $\mathcal{P}(R)$ the powerset of R. A function $f : \mathcal{L} \to \mathcal{P}(R)$ is a *representation of* $\mathcal{L}$ *(and* $\preceq$*) as sets*, if f is one-to-one and surjective, f and its inverse are computable, and for all θ and φ we have $\theta \preceq \varphi$ if and only if $f(\theta) \subseteq f(\varphi)$.

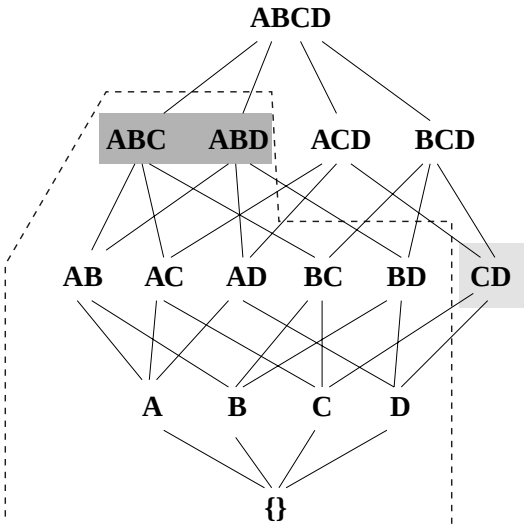

Fig. 2. The lower shaded area represents the downward closure of $S = \{ABC, ABD\}$, S is exactly the positive border and the negative border is the set CD.

Thus, representing as sets requires that the structure imposed on $\mathcal{L}$ by $\preceq$ is isomorphic to a subset lattice. In particular, the lattice must be finite, and its size must be a power of 2. Note that frequent sets, functional dependencies with fixed right-hand sides, and inclusion dependencies are easily representable as sets; the same holds for monotone Boolean functions. However, the language of Mannila et al. [1995] used for discovering episodes in sequences does not satisfy this condition.

Given S, we can compute $Bd^+(S)$ without looking at the data $\mathbf{r}$ at all: simply find the most special sentences in S. The negative border $Bd^-(S)$ is also determined by S, but finding the most general sentences in $\mathcal{L}\backslash S$ can be difficult. We now show how minimal transversals can be used in the task. Assume that (f, R) represents $\mathcal{L}$ as sets, and consider the hypergraph $\mathcal{H}(S)$ on R containing as edges the complements of sets $f(\varphi)$ for $\varphi \in Bd^+(S)$: $\mathcal{H}(S) = \{R\backslash f(\varphi) \mid \varphi \in Bd^+(S)\}$. Then $\mathrm{Tr}(\mathcal{H}(S))$ is a hypergraph on R, and hence we can apply f^{-1} to it: $f^{-1}(\mathrm{Tr}(\mathcal{H}(S))) = \{f^{-1}(H) \mid H \in \mathrm{Tr}(\mathcal{H}(S))\}$. We have the following.

THEOREM 15 [MANNILA AND TOIVONEN 1997]. $f^{-1}(Tr(\mathcal{H}(S))) = Bd^-(S)$.

Example 16. Consider the problem of computing frequent sets illustrated in Figure 2, with attributes $R = \{A, B, C, D\}$. Let $S = \{ABC, ABD\}$, where we use a shorthand notation for sets, for example, we represent $\{A, B, C\}$ by ABC. Then, the downward closure of S is equal to $\{ABC, ABD, AB, AC, AD, BC, BD, A, B, C, D\}$, and S includes the maximal elements. The negative border (that can be found by drawing the corresponding lattice) is $Bd^-(S) = \{CD\}$.

For this problem, we already have $\mathcal{L}$ represented as sets and thus use the identity mapping $f(X) = X$, thus $\mathcal{H}(S) = \{D, C\}$. It is easy to see that $\mathrm{Tr}(\{D, C\}) = \{CD\}$, and thus f^{-1} indeed yields the correct answer.

The requirement for representing as sets is quite strong. It is however necessary. In particular, the mapping f must be surjective, that is, cover all of $P(R)$. Otherwise, after computing the transversal, a set may not have an inverse mapping to be applied in the last transformation in the theorem. This is indeed the case when we consider the problem of finding sequential patterns [Agrawal and Srikant 1995] or episodes [Mannila et al. 1995]. The *a priori* (level-wise) algorithm can be extended to handle sequences of attributes rather than sets of attributes because we can define a monotone specialization relation between sequences. However, it is not clear how to represent sequences of attributes as sets so the Dualize and Advance algorithm cannot be applied in this case.

5. THE LEVEL-WISE ALGORITHM

The *a priori* algorithm [Agrawal and Srikant 1994; Mannila and Toivonen 1997] 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. [Agrawal and Srikant 1994, 1995; Agrawal et al. 1996; Mannila et al. 1995; Mannila and Toivonen 1997]. In Gunopulos et al. [1997a], it is shown to be optimal for the computation of the set $Th(\mathcal{L}, \mathbf{r}, q)$. The algorithm solves the problem MaxTh by simply finding all interesting statements, that is, the whole theory $Th(\mathcal{L}, \mathbf{r}, q)$ going bottom up. The method is as follows:

Algorithm 17. The *a priori* (*level-wise*) *algorithm* for finding all interesting statements.
Input: A database $\mathbf{r}$, a language $\mathcal{L}$ with specialization relation $\preceq$, and a quality predicate q.
Output: The set $Th(\mathcal{L}, \mathbf{r}, q)$.
Method:

1. $C_1 := \{\varphi \in \mathcal{L} \mid$ there is no φ' in $\mathcal{L}$ such that $\varphi' \prec \varphi\}$;
2. $i := 1$;
3. **While** $C_i \neq \emptyset$ **do**
4. $\quad L_i := \{\varphi \in C_i \mid q(\mathbf{r}, \varphi)\}$;
5. $\quad C_{i+1} := \{\varphi \in \mathcal{L} \mid$ for all $\varphi' \prec \varphi$ we have $\varphi' \in \bigcup_{j \leq i} L_j\} \setminus \bigcup_{j \leq i} C_j$;
6. $\quad i := i + 1$;
7. **Od**;
8. output $\bigcup_{j<i} L_j$;

The algorithm works iteratively, alternating between *candidate generation* and *evaluation* phases. First, in the generation phase of an iteration i, a collection C_i of new *candidate* sentences is generated, using the information available from more general sentences. Then the quality predicate is computed for these

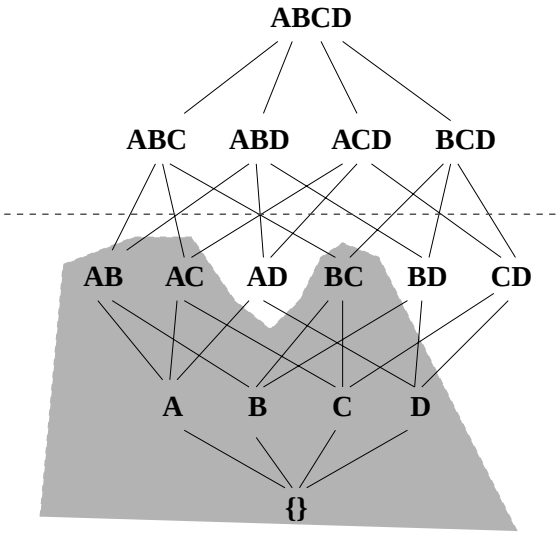

Fig. 3. The *a priori* algorithm operates at levels: After computing the frequent sets of size 2 ($L_2 = \{AB, AC, BC\}$), the only set of cardinality 3 that must be considered is ABC, which is a superset of all the sets in L_2.

candidate sentences. The collection L_i will consist of the interesting sentences in C_i. In the next iteration $i + 1$, candidate sentences in C_{i+1} are generated using the information about the interesting sentences in $\bigcup L_j$ (Figure 3). Note that using the notion of border, Step 5 of the algorithm can be written as $C_{i+1} := Bd^-(\bigcup_{j \le i} L_j) \setminus \bigcup_{j \le i} C_j$.

The algorithm aims at minimizing the amount of database processing, that is, the number of evaluations of q (Step 4). Note that the computation to determine the candidate collection does not involve the database (Step 5). For example, in Agrawal et al. [1996], when computing frequent sets Step 5 used only a negligible amount of time.

Clearly, by definition, the algorithm finds the maximal interesting sentences. Moreover, we show that under certain conditions the algorithm does not take too much time. The following theorem is immediate.

THEOREM 18. *The level-wise algorithm computes the set of interesting sentences correctly, and it evaluates the predicate q*

$$|Th(\mathcal{L}, \mathbf{r}, q) \cup Bd^-(Th(\mathcal{L}, \mathbf{r}, q))|$$

times.

Example 19. Consider, again, the problem of computing frequent sets where $R = \{A, B, C, D\}$ and MaxTh $= \{ABC, BD\}$, that is, the situation of Figure 2. The level-wise algorithm works its way up from the bottom. It starts by evaluating the singletons A, B, C, and D; all of these are frequent. In the second iteration, C_2 contains pairs of attributes such that both attributes are frequent, in this case all attribute pairs. Of them, AB, AC, BC, and BD are

frequent. C_3 then contains such sets of size three all of whose subsets are frequent, that is, the set ABC, which is actually frequent. Notice that the negative border corresponds exactly to the sets that have been found not interesting along the way, that is the sets AD and CD.

In order to further analyze the complexity, we use the following notation. First, recall the definition of rank, given in Section 2, capturing the "level" of a sentence. Denote by $dc(k)$ the maximal size of the downward closure of any sentence ϕ of rank $\leq k$. Also, by $width(\mathcal{L}, \preceq)$, denote the maximal number of immediate successors on $\mathcal{L}$ and $\preceq$. That is,

$$width(\mathcal{L}, \preceq) = \max_{\theta} |\{\phi \mid \theta \prec \phi \text{ and for no } \psi, \theta \prec \psi \prec \phi\}|.$$

THEOREM 20. *Let k be the maximal rank over all interesting sentences in the problem $(\mathcal{L}, \mathbf{r}, q)$. The level-wise algorithm computes the set of interesting sentences correctly, and the number of queries it makes is bounded by*

$$dc(k)\,\text{width}(\mathcal{L}, \preceq)\,|MTh(\mathcal{L}, \mathbf{r}, q)|.$$

PROOF. The number of sentences below any maximal element is bounded by $dc(k)$, and thus the number of elements not rejected from C_i at all stages together is bounded by $dc(k)|MTh(\mathcal{L}, \mathbf{r}, q)|$. Each of these sentences might create at most $width(\mathcal{L}, \preceq)$ new sentences for consideration in C_{i+1} that may be rejected (i.e., they are in $Bd^-(Th(\mathcal{L}, \mathbf{r}, q))$). □

This result holds for any $(\mathcal{L}, \mathbf{r}, q)$. For problems representable as sets one can derive more explicit bounds. In particular, in the problem of frequent sets the rank corresponds to the size of the set, the width is the number of attributes, and $dc(k) = 2^k$. A standard assumption in practical applications is that the size of frequent sets is bounded. In these cases the level-wise algorithm is indeed efficient:

COROLLARY 21. *Let k be the size of the largest frequent set, and n the number of attributes. The level-wise algorithm computes the set of frequent sets correctly, and the number of queries it makes is bounded by $2^k n|MTh(\mathcal{L}, \mathbf{r}, q)|$.*

As a further corollary of the above, we get that if the size of frequent sets is not too large then the size of $Bd^-(Th(\mathcal{L}, \mathbf{r}, q))$ is not prohibitive and thus the problem is feasible.

COROLLARY 22. *Let k be the size of the largest frequent set, and n the number of attributes.*

(i) *The size of sets in $Bd^-(Th(\mathcal{L}, \mathbf{r}, q))$ is bounded by $k + 1$.*
(ii) *If $k = O(\log n)$, the size of $Bd^-(Th(\mathcal{L}, \mathbf{r}, q))$ is bounded by $O(n^{O(1)}|MTh(\mathcal{L}, \mathbf{r}, q)|)$.*

We thus get an application for hypergraph transversals:

THEOREM 23. *For $k = O(\log n)$, the problem of computing hypergraph transversals, where the edges of the input graph are all of size at least $n-k$, is solvable in input polynomial time by the level-wise algorithm.*

PROOF. If the edge size is at least $n - k$, then the maximal sets that are not transversals are of size at most k. Set non-transversals to be "interesting" and use the algorithm. We get that the negative border is the required transversal hypergraph. □

This improves on previous results by Eiter and Gottlob [1995] (Theorem 5.4) that show that this is possible for constant k (and uses a brute force enumeration algorithm using property (i) above). Notice that the level-wise algorithm does not use the structure of original hypergraph directly. All it does is to test whether certain subsets are transversals of it or not.

6. THE DUALIZE AND ADVANCE ALGORITHM

The results of the previous section show that the *a priori* algorithm is optimal if we want to compute all frequent elements. It also performs very well when we want to compute the maximal elements, and the size of the maximal elements is small. Given this analysis, the disadvantage of the level-wise approach is clear. If there is an element in MTh whose rank is large, then all the downward closure of this element will be tested and this would require too much time.

Example 24. For example, consider the situation where there are n attributes, and k maximal sets of size $\frac{n}{k}$. Then, the size of the border is $O(n^2)$ (the maximal frequent sets are k, and the minimal nonfrequent sets are all pairs of attributes that belong to different maximal frequent sets), while the total number of frequent sets is $O(n^{n/k})$. For constant k, an algorithm that finds the maximal frequent sets by way of computing all the frequent sets can take exponential amount of computational time even if the size of the output is polynomial.

Intuitively, one would want in such case to have an algorithm that goes directly to the maximal element instead of exploring all of its downward closure first. Our algorithm captures exactly this intuition with the subroutine AMSS (A Most Specific Sentence). Given an interesting sentence φ, AMSS finds a maximal interesting sentence θ, such that $\varphi \preceq \theta$. Once a set of most specific sentences is found the algorithm focuses its search by computing the negative border of the sentences found so far (using a transversal computation), and starting its upward search from this negative border. Clearly, if progress can be made, it can be made from the negative border and thus the approach is guaranteed to succeed. These intuitions are formalized in the algorithm and its analysis that follows.

While the algorithm can be phrased for any $(\mathcal{L}, \preceq)$, our analysis only holds for problems representable as sets, and we thus describe it in the restricted setting.

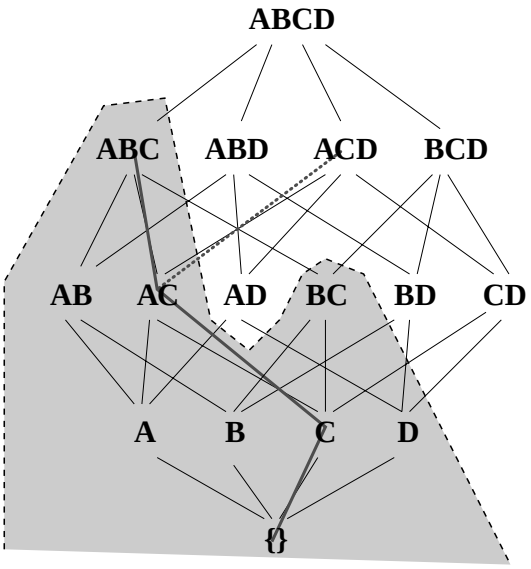

Fig. 4. The operation of the algorithm AMSS. The attributes are considered in the order C, A, D, B in the example. Sets C and AC are found frequent, the set ACD is not frequent, and finally ABC is found to be a maximal frequent set.

We first give the algorithm to compute one most specific sentence from Th. Denote $\psi \prec_1 \theta$, if $\psi \prec \theta$ and for no φ we have $\psi \prec \varphi \prec \theta$; in this case, we say that θ is an *immediate specialization* of ψ.

Algorithm AMSS. Given a relation $\mathbf{r}$ with attributes $\{R_1, \ldots R_m\}$, a language $\mathcal{L}$ with specialization relation $\preceq$, equivalent to the subset relation on R, and a quality predicate q, find a most specific sentence from $MTh(\mathcal{L}, \mathbf{r}, q)$.

(1) Find a permutation p of the numbers $1, 2, \ldots m$.
(2) $\psi := \{\}$.
(3) For $i = 1$ to m do:
    If $q(\mathbf{r}, \psi \cup \{R_{p(i)}\})$ holds, let $\psi := \psi \cup \{R_{p(i)}\}$.
(4) Output ψ.

The algorithm assumes that $\{\} \in \mathcal{L}$, and proceeds to specialize it successively until a most specific sentence is found (Figure 4). In Step 2, if ψ is initialized with an arbitrary sentence $\sigma \in Th$ instead of "true", then the algorithm will find a most specific sentence s' such that $s \preceq s'$.

Note that the correctness of the algorithm, and the subsequent analysis does not depend on the use of a specific permutation p. In order to discover a maximal frequent set, the algorithm has to consider all attributes sequentially, but the actual order does not matter. The use of random permutations is an interesting heuristic that can allow more efficient discovery of new maximal frequent sets.

Once a collection C of most specific sentences is found, any new most specific sentence F cannot be a subset or a superset of any of the ones found so far. It follows that F intersects all the complements of all sets in C and therefore is a transversal of the hypergraph whose edges are the complements of all sets in C.

To discover new most specific sentences, we start with a minimal transversal of the hypergraph whose edges are the complements of sets in C, and extend it to a most specific sentence. If every minimal transversal is considered for this extension, then every new most specific sentence will be discovered.

Denote by Algorithm AMSS(ψ_{init}) the parameterized version of the Algorithm AMSS, which starts by initializing ψ with the sentence ψ_{init}.

We now give the general algorithm for finding all most specific sentences.

Algorithm All_MSS. Given a relation $\mathbf{r}$ with attributes $\{R_1, \ldots R_m\}$, a language $\mathcal{L}$ with specialization relation $\preceq$, equivalent to the subset relation on R, x a quality predicate q, compute $MTh(\mathcal{L}, \mathbf{r}, q)$.

1. $i := 1$
2. $C_i = \{\}$
3. $\overline{D_i} :=$ {complements of sets in C_i}
4. Use a subroutine to enumerate the minimal transversals of $\overline{D_i}$
5. For each transversal X enumerated:
   (a) if $q(\mathbf{r}, X)$ holds, mark X as a counter example and quit loop
6. If, for every transversal, $q(\mathbf{r}, X)$ does not hold output C_i and exit.
7. For the counterexample X:
   Run Algorithm AMSS(X) to find a maximal superset Y of X such that $q(\mathbf{r}, Y)$ holds
8. $C_{i+1} = C_i \cup \{Y\}$
9. $i = i + 1$
10. Go to 3.

It is useful to notice that the transversal computation does not look at the data, only at elements of $\mathcal{L}$; if the input data is large, a complicated computation on $\mathcal{L}$ can still be much cheaper than just reading the data once.

Example 25. Consider the problem of computing frequent sets described in Figure 5.

The algorithm All_MSS starts with $C_1 = \emptyset$, and $\overline{D_1} = \{ABCD\}$. The transversals are $\mathrm{Tr}(\overline{D_1}) = \{A, B, C, D\}$. Assume that, in Step 5, the transversal A is tested first. Then, A is found interesting and the algorithm continues in Step 7 to find a maximal element Y. This can be done by adding one attribute at a time, and testing whether q holds, and yields $Y = ABC$. In the next iteration $C_2 = \{ABC\}$, $\overline{D_2} = \{D\}$, and $\mathrm{Tr}(\overline{D_2}) = \{D\}$. In Step 5 D, is found to be interesting, and, in Step 7, the algorithm finds that $Y = BD$ is maximal interesting. We therefore have $C_3 = \{ABC, BD\}$, $\overline{D_3} = \{D, AC\}$, and $\mathrm{Tr}(\overline{D_3}) = \{AD, AC\}$. All the elements of the transversal are not interesting and therefore the algorithm stops. The set C_3 is exactly MTh and $\mathrm{Tr}(\overline{D_3})$

6.1 The Complexity of the Algorithm

We now prove an upper bound for the complexity of the algorithm. Similar bounds were previously obtained by [Bshouty et al. 1996; Bioch and Ibaraki 1995] in the context of identifying Boolean functions with membership queries (see Section 2.3), for somewhat different algorithms. In order to establish correctness, we start with a simple lemma:

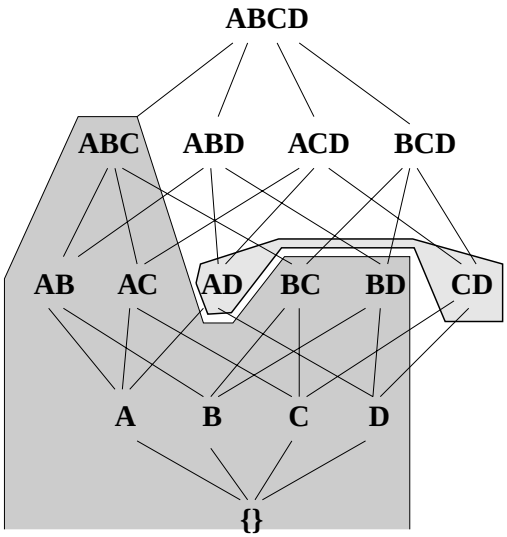

Fig. 5. The operation of the Dualize and Advance algorithm: Assume that *ABC* and *BD* are the maximal frequent sets found so far. Then any other frequent set must be a superset of either *AC* or *CD*. These are the transversals of the hypergraph with edges $D = ABCD \backslash ABC$ and $AC = ABCD \backslash BD$.

LEMMA 26. *For any iteration i of the algorithm, if $C_i \neq MTh(\mathcal{L}, \mathbf{r}, q)$, then at least one of the elements of $Tr(\overline{D_i})$ is interesting.*

PROOF. First note that the elements of C_i are verified by the algorithm to be maximal interesting. We therefore have $C_i \subseteq MTh(\mathcal{L}, \mathbf{r}, q)$. Now if there is a set $c \in MTh(\mathcal{L}, \mathbf{r}, q) \backslash C_i$, then (since $\preceq$ is monotone) there is a minimal interesting set not in C_i, that is there is an interesting set in $Bd^-(C_i)$. (Just walk down in the lattice to find such a set).

As we saw earlier, sets of attributes are already represented as sets and the identity mapping $f(X) = X$ is used. We thus get from Theorem 15 that $\mathrm{Tr}(\overline{D_i}) = Bd^-(S)$ and one of these elements is interesting. □

The question is how many sets X should be enumerated before finding such a counterexample on the negative border. The following example shows that there are cases where the size of $MTh(\mathcal{L}, \mathbf{r}, q)$ and its negative border are small, but in an intermediate step the size of the negative border of C_i may be large.

Example 27 [*Mannila and Räihä* 1986]. Consider the case where $MTh = MTh(\mathcal{L}, \mathbf{r}, q)$ includes all sets of size $n-2$, and $Bd^-(MTh)$ thus includes all sets of size $n-1$. Further consider the case where C_i is such that $\overline{D_i} = \{\{x_{2i-1}, x_{2i}\} \mid 1 \leq i \leq n/2\}$. Then the size of $\mathrm{Tr}(\overline{D_i})$ is $2^{n/2}$ while $Bd^-(MTh)$ is small.

LEMMA 28. *For any iteration i of the algorithm, if $C_i \neq MTh(\mathcal{L}, \mathbf{r}, q)$ then the number of sets enumerated before a counterexample set X is found is bounded by $|Bd^-(MTh(\mathcal{L}, \mathbf{r}, q))|$.*

PROOF. Denote $Bd^- = Bd^-(MTh(\mathcal{L}, \mathbf{r}, q))$. We show that each set X enumerated either matches an element of Bd^- exactly, or is interesting. In other words, the set X cannot be both not interesting, and a strict superset of an element in Bd^-. It follows that at most $|Bd^-|$ elements need to be enumerated.

To prove the claim, notice that every set that is deemed interesting by the C_i is indeed interesting. If X is both not interesting, and a strict superset of an element Z in Bd^-, then Z, which is not interesting, is claimed interesting by C_i (since $X \in Bd^-(C_i)$ is minimal, and $Z \subset X$); a contradiction. □

THEOREM 29. *If there is an incremental $T(I, i)$ time algorithm for computing hypergraph transversals then $MTh = MTh(\mathcal{L}, \mathbf{r}, q)$ can be computed in time polynomial in $|MTh|$ and $T(|MTh|, |Bd^-(MTh)|)$, while using at most $(|Bd^-(MTh)| + width(\mathcal{L}, \preceq)|MTh|)$ queries.*

PROOF. The bound follows by using algorithm All_MSS. By Lemma 26, each iteration finds a new maximal set, and therefore the algorithm is correct, and the number of iterations is $|MTh|$. By Lemma 28, in each iteration the algorithm runs a transversals subroutine that enumerates sets on the negative boundary. Each set is either a counterexample, or is a set of $Bd^-(MTh)$. If we keep the sets in $Bd^-(MTh)$ that we have already discovered, then, for each set X that the transversal subroutine enumerates, first we determine whether it is one of the sets we know already are in $Bd^-(MTh)$ (in which case we can ignore it), and otherwise we check if it is frequent. If not, then X is part of $Bd^-(MTh)$, but if it is frequent, we try to extend it to a maximal set. The extension of the counterexample X into a maximal set Y requires at most $rank(MTh)$ stages each with one query. The total number of queries is then $|Bd^-(MTh)|$ to discover the negative boundary, and at most $width(\mathcal{L}, \preceq)$ to discover each maximal set (since we can use the order induced by R and avoid trying to add an attribute more than once when going up in the lattice). □

Thus, we see that the connection to hypergraph transversals holds not only for the verification problem but also for the generation problem. It is also important to notice that the time complexity and query complexity of the algorithm are separated. Theorem 29 shows that, while the running time depends on the transversal enumeration and may not be polynomial, only a polynomial number of queries is required.

Recently, Fredman and Khachiyan [1996] presented an incremental algorithm for the HTR problem with time complexity $T(I, i) = (I + i)^{O(\log(I+i))}$. We can therefore conclude the following:

THEOREM 30. *For any problem representable as sets, $MTh(\mathcal{L}, \mathbf{r}, q)$ can be computed in time $t(|MTh| + |Bd^-(MTh)|)$, where $t(n) = n^{O(\log n)}$, while using at most $(|Bd^-(MTh)| + width(\mathcal{L}, \preceq)|MTh|$ queries.*

The theorem shows that the Dualize and Advance algorithm always takes subexponential time to the size of the output (number of maximal elements), and can in fact be exponentially faster than the *a priori* algorithm.

7. APPLYING THE DUALIZE AND ADVANCE ALGORITHM FOR COMPUTING MAXIMAL FREQUENT SETS, FOR FINDING MINIMAL KEYS, AND FOR LEARNING MONOTONE FUNCTIONS

In this section, we discuss how to adapt the Dualize and Advance algorithm to find maximal frequent sets of a $\{0, 1\}$ matrix and threshold value σ, for finding minimal keys in a database, and for learning monotone functions. The case of frequent sets is straightforward, and we briefly outline the approach here.

As in the general case, the following lemma guarantees the success of our method.

LEMMA 31. *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 $T \subseteq F$.*

To apply the general algorithm (Algorithm AMSS), we use the lattice structure efficiently: the process can be seen as a random walk in the lattice. Given X, in order to select a set Y such that $X \preceq_1 Y$ the only thing we have to do is to get an item $A \in R \backslash X$ and let $Y = X \cup \{A\}$. We give the algorithm AMFS (A Maximal Frequent Set), which finds a single maximal σ-frequent set containing a given set S of attributes. This algorithm corresponds to the parameterized version of the algorithm AMSS.

Algorithm AMFS(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 $\{A_{S_1}, \ldots, A_{S_l}\}$; 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 $i \leq |S|$, $p(i) = S_i$.
(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 that the AMFS algorithm does not need to make $width(\mathcal{L}, \preceq)$ passes over the data, but instead can be performed efficiently by maintaining an index per item.

THEOREM 32. *Let S be a σ-frequent set of a relation r. Then the algorithm AMFS(S) finds the lexicographically first (in according with the ordering given by p) maximal σ-frequent set containing attributes in S. Further, its time complexity is $O(|r|)$.*

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 $\alpha(X, r)$ 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 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 $O(m|R|) = O(|r|)$, linear to the size of the relation 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 (with respect to 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, they will all be included in X is 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 following is a corollary of Theorems 30 and 32:

COROLLARY 33. *The algorithm All_MFS finds all maximal σ-frequent sets of the input matrix M in time subexponential to the number of the maximal σ-frequent sets.*

7.1 Finding Minimal Keys

We now discuss our algorithm for discovering all minimal keys of a database. We first note that for the case of functional dependencies with fixed right-hand side, and for keys, even simpler algorithms can be used [Mannila and Räihä 1992; Khardon 1995]. In this case, one can access the database and directly compute $Bd^-(MTh)$ (according to the appropriate representation as sets, this corresponds to the so called agree sets of the relation). Then a single run of an HTR subroutine suffices. The current approach can be applied even if the access to the database is restricted to "Is-interesting" queries. Furthermore, as we show, it can be implemented efficiently, and depending on the database, it can be more efficient since it avoids the computation of the agree sets that is quadratic in the number of tuples in the database.

To keep an analogy with the problem of discovering maximal frequent sets, we use the notion of an *anti-key*. An *anti-key* in a database is a set of fields which is complement of some key of the 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 database is equivalent to the problem of finding all maximal anti-keys of the database. To keep presentation analogous to maximal frequent sets, we will henceforth in this section talk only of the problem of finding all maximal anti-keys of a database.

We first present algorithm AMAK (A Maximal Anti-Key) for finding one maximal anti-key containing a given set of fields.

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

1. Find a permutation p of $(1, 2, \ldots, m)$ such that for $j \leq s$, $p(j) = i_j$.
2. Set $A = \emptyset$.

3. For $j = 1$ to m:
   (a) If $A \cup \{f_{p(j)}\}$ is an anti-key, add $f_{p(j)}$ to A.
4. If $S \subset A$, return A.

As in the problem of frequent sets, we show that this procedure can be implemented efficiently. In principle, in order to check that a set is an anti-key all pairs of rows must be compared. We show that this can be done efficiently by maintaining suitable data structures. The details however, are more involved than in the case of frequent sets, and we first present the expanded version of algorithm AMAK.

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

1. Find a permutation p of $(1, 2, \ldots, m)$ such that for $j \leq s$, $p(j) = i_j$.
2. Set m pointers to the columns (i.e., fields) of the matrix according to the permutation so that we can assume without loss of generality that the columns of the matrix are in the order defined by p.
3. Compute right_to_left profile matrix $RL_{n\times m}$ as follows:
   (a) Consider the mth column of M. Relabel the values in this column with consecutive positive integers starting from 1 so that identical values are labeled with the same integer and different values are labeled with distinct integers. For all i, define $RL_{i,m}$ to be the integer labeling the value in $M_{i,m}$.
   (b) For $j = m - 1$ to 1:
   Consider the n pairs defined by the values in jth column of M and $(j + 1)$th column of RL. Relabel the pairs with consecutive positive integers starting from 1 so that identical value pairs are labeled with the same integer and different value pairs are labeled with distinct integers. For all i, define $RL_{i,j}$ to be the integer labeling the pair in $(M_{i,j}, RL_{i,(j+1)})$.
4. Initialize the left_to_right profile array $LR_{n\times 1}$ to be all 0's. Initialize A to be empty set.
5. For $j = 2$ to m:
   (a) Consider the n pairs defined by values in LR and the jth column of RL. Label all the pairs with consecutive positive integers starting from 1 so that identical pairs are labeled with the same integer and different pairs are labeled with distinct integers.
   (b) If the labeling uses all integers from $\{1, 2, \ldots, n\}$ then $A = A \cup \{j - 1\}$ else
      i. Consider the n pairs defined by the entries in LR and the $(j - 1)$th column of M. Label the pairs with consecutive positive integers so that identical pairs are labeled with the same integer and different pairs are labeled with distinct integers. For all i, update the value of $LR_{i,1}$ to be the integer labeling the ith pair.
      ii. If the labeling uses all integers from $\{1, \ldots, n\}$, then $A = A \cup \{j, j + 1, \ldots, m\}$ and go to Step 6.
6. For $k = 1$ to $|S|$: If $f_k \notin A$ then Stop.
7. Output A.

Observation. Let j be an integer from $\{1, 2, \ldots, m\}$. Consider the n tuples formed by taking the projection of the database with respect to the columns $\{j, j + 1, \ldots, m\}$. Then the jth column of RL represents the distinctness of these n tuples that is, $RL_{i_1,j}$ and $RL_{i_2,j}$ are different iff the i_1^{th} and i_2^{th} tuples are distinct. This follows by a simple induction on j.

Observation. At the end of the ith iteration of the for loop (just before Step 6, the array LR represents the distinctness of the n tuples formed by columns in the set $\{f_1, \ldots, f_i\} \setminus A$. This follows by a simple induction on the loop variable j. A field f_j is inserted in A only if the fields before j that are not in A together with the fields $f_{j+1}, \ldots, f_m$ form a key. Therefore, A is always the complement of a key.

THEOREM 34. *For a given set S of fields, suppose there is an anti-key that contains all the fields in S. Then, the algorithm AMAK outputs the lexicographically smallest maximal anti-key with respect to the permutation and that contains all the fields in S. Further, assuming that the time to access any field of any record is constant, the running time of the algorithm is $O(nm)$.*

PROOF. The algorithm greedily inserts fields in A, under the invariant that A remains an anti-key, and so outputs the lexicographically first maximal anti-key, with respect to permutation p. Since p has all fields in S before any other key, A includes all fields in S if S is an anti-key.

Note that the Step 3 makes one pass of the whole database and hence need $O(nm)$ time. Here we are assuming the domain is prefixed and finite so that the assignment of names can be done in linear time using a bucket sort like method. Similarly Step 5 makes one pass of the database. Other steps take $O(m)$or $O(n)$ time. Therefore the total time complexity of the algorithm is $O(nm)$. □

We now give the complete algorithm for finding all maximal antikeys, which is analogous to the algorithm for finding all maximal frequent sets. First we point out that an analogue of the Lemma 26 holds also for the case of maximal anti-keys.

LEMMA 35. *Given a collection C of maximal anti-keys of a database, 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 description of the algorithm All_MAK, 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, find all maximal anti-keys.

(1) $C = \{\}$
(2) Run algorithm AMAK(ϕ) and add to C the maximal anti-key discovered.
(3) While new antikeys 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 AMAK(X) and add any new maximal anti-key found to C.
(4) Output C.

We can now claim the following corollary.

COROLLARY 36. *The algorithm All_MAK finds all maximal anti-keys (and hence minimal keys) of the input database.*

7.2 Learning Monotone Functions

In this section, we apply the results of the previous section to the problem of learning monotone Boolean functions using membership queries. Theorem 5 shows that this problem is equivalent to the problem of finding maximal frequent sets.

Example 37. The problem of computing the maximal frequent sets described in Figure 4 is mapped to the problem of learning the function f whose DNF representation is $f = AD \vee CD$ and whose CNF representation is $f = (A \vee C)(D)$. The terms of the DNF correspond to the elements of Bd^-, and the clauses of the CNF are the complements of the sets in MTh.

As an immediate corollary of the results in Section 5, we get:

COROLLARY 38. *The level-wise algorithm can be used to learn the class of monotone CNF expressions where each clause has at least $n - k$ attributes and $k = O(\log n)$, in polynomial time, and with a polynomial number of membership queries.*

As a corollary of Theorem 10, we get a lower bound:

COROLLARY 39. *Any algorithm that learns monotone functions with membership queries must use at least $|DNF(f)| + |CNF(f)|$ queries.*

While the bound is not surprising, it hinges on the lower bound given by Angluin [1988]. It is shown there that an algorithm may need to take time exponential in the DNF size when not allowed CNF size as a parameter. Indeed the CNF size of the function used to show the lower bound is exponential. (The lower bound in Angluin [1988] is, however, more complex since the learner has access to several additional oracles.) On the other hand, by using Theorem 29, we see that with membership queries alone one can come close to this lower bound.

COROLLARY 40. *If there is an incremental T(I,i) time algorithm for computing hypergraph transversals, then there is a learning algorithm for monotone functions with membership queries, that produces both a DNF and a CNF representation for the function. The number of MQ queries is bounded by $(|DNF(f)| + n|CNF(f)|)$. The running time of the algorithm is polynomial in n and $T(|CNF(f)|, |DNF(f)|)$.*

As noted earlier, Bioch and Ibaraki [1995] have studied the same problem and have previously derived a similar result, as well as other related results. The result can also be derived from a more general construction in Bshouty et al. [1996] (Theorem 18), that studies the complexity of learning in the context of NP-Oracles.

Here again, using the result of Fredman and Khachiyan [1996], we can derive as a corollary a subexponential learning algorithm for this problem.

8. COMPUTING THE TRANSVERSALS OF THE HYPERGRAPH INCREMENTALLY

The results of Section 6 show that the efficient operation of the Dualize and Advance algorithm depends on an incremental algorithm for computing the transversals of a hypergraph. The general problem of finding all minimal transversals of a hypergraph in output polynomial time is still an open problem. The algorithm by Fredman and Khachiyan [1996] generates all transversals in provably output subexponential time, however it is difficult to implement.

In this section, we present a heuristic technique for finding the transversals of a hypergraph. Our technique has the advantage of being easy to describe and implement, as well as being able to continue the transversal computation at each step of the Dualize and Advance algorithm from the previous one. As a result, the transversal computation can be integrated to the algorithm instead of starting from the beginning at each step.

Consider two consecutive steps of the algorithm All_MSS, the ith and the $(i+1)$th. If some new maximal frequent sets were found during step i, then $\overline{D_i} \subset \overline{D_{i+1}}$. Let X be a transversal of $\overline{D_i}$. Then either X is a transversal of $\overline{D_{i+1}}$ as well, or it can become a transversal of $\overline{D_{i+1}}$ if it is expanded by a set of items that cover $\overline{D_{i+1}} \setminus \overline{D_i}$. In fact, each such transversal can be expended to a set of transversals of the set $\overline{D_{i+1}}$.

Example 41. Let $\overline{D_i} = \{\{A, B\}, \{B, C\}\}$, and $\overline{D_{i+1}} = \{\{A, B\}, \{B, C\}, \{D, E\}\}$. Then, the set $\{B\}$ is a transversal of $\overline{D_i}$, but $\overline{D_{i+1}}$ contains an additional edge that is not covered by $\{B\}$. We can expand $\{B\}$ by adding D or E (these cover the new edge). In both cases we get a transversal for $\overline{D_{i+1}}$ ($\{B, D\}$ and $\{B, E\}$, respectively.

It follows that, if we have the transversals of $\overline{D_i}$, we can incrementally create the transversals of $\overline{D_{i+1}}$ by expanding each of the original transversals so that they intersect each of the new complements of maximal elements.

In addition, if a transversal X is found to be not frequent at step i, then we do not expand this transversal in the next step. All the transversals we can generate from it have to be nonfrequent as well. This is an important improvement because it allows us to reduce the number of transversals we have to generate at each step.

The algorithm is given below:

Algorithm for computing transversals at Step $i+1$. Let $\mathcal{T}_i$ be the set of transversals at step i.

(1) Set $\mathcal{T}_{i+1} = \{\}$
(2) For each $X \in \mathcal{T}_i$
  (a) Expand X and obtain a set $\mathcal{X}'$ of transversals of $\overline{D_{i+1}}$.
  (b) For each $X' \in \mathcal{X}'$:
    i. It X' is not interesting, remove X' from $\mathcal{X}'$
  (c) $\mathcal{T}_{i+1} = \mathcal{T}_{i+1} \cup \mathcal{X}'$

To complete the algorithm we need a technique to efficiently extend each transversal in $\mathcal{T}_i$ so that it intersects all the complements of the new maximal

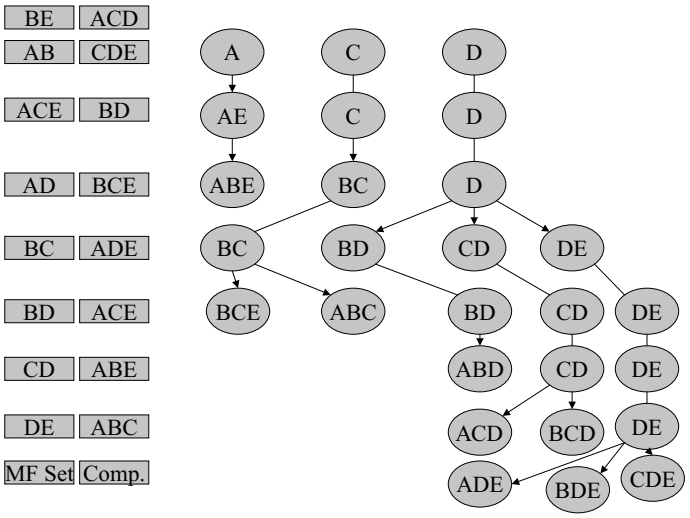

Fig. 6. To compute the transversals incrementally we use a tree structure: Assume attributes A, B, C, D, E. Let BE be the first maximal frequent set found. It's complement is ACD, and the transversals of the complement are A, C, and D. When a new maximal frequent set is found (AB in the second row), the existing transversals are extended, if necessary, to intersect the complement of the new maximal frequent set (CDE). In this case, the new transversals are AE, C, D. When a transversal is found to be non-frequent, (as is ABE in the example), the branch of the tree is pruned.

frequent sets. We use the tree-structure scheme for traversing the set of transversals of hypergraph that was presented in Kavvadias and Stavropoulos [1999]. The operation of the algorithm is shown in Figure 6.

9. RELATED WORK

Recently, a number of algorithms have been proposed to solve the MaxTh problem for frequent sets [Bayardo 1998; Lin and Kedem 1998; Burdick et al. 2001; Agrawal et al. 2000; Han et al. 2000; Gouda and Zaki 2001]. Below, we briefly describe the most recent and important of these algorithms and compare their design approach with Dualize and Advance. We note that none of these algorithms has provably worst case complexity that is subexponential to the size of the output, as Dualize and Advance does (Theorem 30). For some of these algorithms [Bayardo 1998; Lin and Kedem 1998; Han et al. 2000], we give examples that show that their worst-case complexity is exponential to the size of the output (i.e., to the number of maximal frequent sets). Therefore, in the worst case, these algorithms are provably asymptotically slower than Dualize and Advance. The other algorithms use heuristics and their worst case complexity is not yet known.

Pincer-Search [Lin and Kedem 1998] is similar to *a priori* and to Dualize-and-Advance in that it also uses the lattice method to enumerate the item sets. Pincer-Search combines a bottom-up and a top-down technique to find the maximal frequent sets. The bottom-up technique moves through the lattice level by level. The bottom-up process finds frequent sets, and nonfrequent sets. The

nonfrequent sets are used by a top down process to refine a set of potential maximal frequent sets. For example, assume that at the beginning of the algorithm, a set *ABCD* is a potential maximal frequent set. If a set *BD* is found to be non-frequent, then *ABCD* cannot be frequent any more. It is refined to the sets *ABD* and *ABC* that are potential frequent sets and their support is checked in the next iteration of the algorithm.

The combination of a bottom-up and a top-down search can result to significant improvements compared to *a priori*. One such case is if there exists only one maximal frequent set of cardinality $n - 1$ (where n is the number of attributes). In this case, Pincer-search finds this set in two database passes, and after finding the support of its subsets of cardinality 1 and 2.

The operation of Pincer-Search has similarities to the operation of the Dualize and Advance algorithm. The main difference is that Pincer-Search tries to guess maximal frequent sets, by considering large sets that may be frequent, while Dualize and Advance starts from sets that have to be frequent and expands them to maximal frequent sets. As a result, Dualize and Advance guarantees that new maximal sets are found in each iteration (if a maximal set is not found, then a set in the negative boundary is found) and thus is always making progress, while Pincer-Search does not always do that.

The following example shows a case where this happens. Assume that there are n attributes, all the sets of size at most k are frequent, and no sets of $k + 1$ are frequent. For such a dataset, the performance of Pincer-Search is identical to the performance of *a priori*. This is because the bottom-up search does not find any non-frequent sets until the kth step, and therefore the top down search cannot refine the candidate maximal frequent sets until that point. Therefore the number of sets for which the algorithm finds their support can be exponential to the number of frequent sets. If for example $k = n - c$, for some constant c, the number of frequent sets is $O(2^n)$, and the number of the maximal frequent sets is $O(n^c)$ (see also Example 24).

This example shows that Pincer-Search can take time that is exponential to the size of the output because it has to find the support of a number of sets that is exponential to the number of maximal frequent sets. In this case, the performance of Pincer-Search is exponentially worse that the performance of the Dualize and Advance algorithm, in terms of the number of Is-interesting queries executed.

FP-Growth [Han et al. 2000] uses a new data structure, the FP-tree, to represent all the transactions of the database. It then uses this structure with a recursive technique to find large frequent sets. This is a very efficient technique for finding all frequent sets. However, although the technique does not need to compute sets of candidate patterns, it still finds the support of all frequent sets and therefore can be less efficient than other techniques when used to find the maximal frequent sets only.

Max-Miner [Bayardo 1998] was one of the first successful practical algorithms for finding all maximal frequent sets. The algorithm uses Rymon's set enumeration [Rymon 1992] to enumerate all the itemsets (Figure 7). Max-Miner reduces the search space by pruning the tree to eliminate both supersets of infrequent sets and subsets of frequent sets. Essentially, when the algorithm

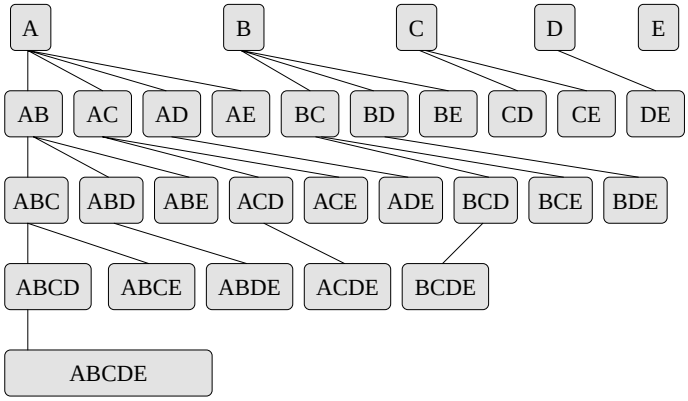

Fig. 7. Rymon's set enumeration tree.

computes the support of a set A, it also computes the support of the largest set that appears in the subtree rooted at this set (by the design of Rymon's tree, this is a superset of A). If this superset is frequent, then all other sets in this subtree must be frequent too. Thus, the algorithm avoids having to compute the support of all the frequent sets, and therefore performs better than *a priori* when the size of the maximal frequent sets is large.

Max-Miner uses a breadth-first-search technique to explore the set enumeration tree. It computes the support of candidate sets that are on the same level of the tree in one database pass. An important factor in the efficiency of the algorithm is the use of heuristics that reorder the attributes, and therefore change the order that the various sets appear in the tree [Burdick et al. 2001]. Experimentally, it has been shown that ordering the attributes in increasing support gives the best results [Bayardo 1998]. In the best case, Max-Miner can find all maximal frequent sets in a few database passes, and using a small number of Is-interesting queries (linear to the number of maximal frequent sets).

In its operation, Max-Miner uses a similar approach to Dualize and Advance in looking for large frequent sets as early as possible, and in trying to prune parts of the space that are known to be frequent. The difference is that Dualize and Advance uses a slow procedure (the transversal computation) that guarantees efficient pruning, while Max-Miner uses heuristics that are much more efficient to apply but do not offer such a guarantee, as the following example demonstrates.

An example of a dataset that results to running time that is exponential to the size of the output in the execution of Max-Miner is given in Figure 8. Assume that all itemsets are frequent, except the supersets of a given set with two attributes. Also assume that the two items in this set always have individually the highest support. This can happen if we take a dataset where all sets are frequent for a given small threshold, add two new items (D and E in Figure 8), and create a new dataset as follows: For each tuple in the original dataset, we create two new tuples and add these in the new dataset. The original attributes remain the same, but in the first new tuple attribute E is 1 and D is 0, and in the second new tuple E is 0 and D is 1. So the set DE is not frequent, but the

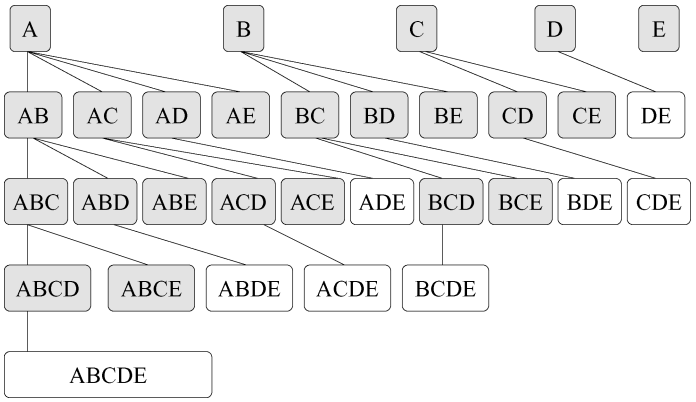

Fig. 8. A worst case for Max-Miner. Set DE and its supersets are the only nonfrequent sets. We assume that the ordering of the attributes, A, B, C, D, E, is in ascending support.

attributes D and E have always the highest individual support. In this case, Max-Miner has to compute the support of all frequent sets because no subtree can be eliminated. The number of Is-interesting queries evaluated is the same with *a priori*, and exponential to the number of the maximal frequent sets. Assuming n attributes, the algorithm has to find the support of $O(2^n)$ sets, while the number of maximal frequent sets is constant. In the example of Figure 8, the attributes are A, B, C, D, E, ordered in ascending support. If the set DE and its supersets are infrequent, the only maximal frequent sets are $ABCD$ and $ABCE$, yet Max-Miner has to find the support of all frequent itemsets.

Similar to Max-Miner, MAFIA [Burdick et al. 2001], DepthProject [Agrawal et al. 2000], and MaxGen [Gouda and Zaki 2001] use Rymon's set enumeration technique. The design of DepthProject has been influenced by the continuous increase in available main memory size. It assumes that the dataset fits in main memory and does not attempt to minimize the number of dataset passes. Both MAFIA and DepthProject use a depth-first-search mechanism to enumerate the itemsets. All three algorithms also use efficient pruning heuristics to avoid examining some itemsets that can be determined to be frequent or nonfrequent. Compared to Max-Miner, MAFIA, DepthProject and MaxGen use the same technique to enumerate the search space (although MAFIA and DepthProject enumerate the space in different order since they use depth-first-search enumeration), and use a similar set of heuristics to prune the set enumeration tree, including: (i) attribute reordering, and (ii) checking the support of the largest set in the subtree rooted at the current node. They also introduce new optimizations; one such optimization, used by MAFIA and MaxGen checks if candidate sets are subsets of known maximal frequent sets before computing their support. However, they use different techniques for computing the support, resulting to different performance characteristics. For example, MAFIA uses a very efficient vertical bitmap representation with compression, which significantly improves the running time. This set of algorithms (including Max-Miner) has proven to be very efficient in practice, however their performance

crucially depends on how effective the pruning heuristics that the algorithms employ are in reducing the search space. The worst-case complexity of these algorithms is not known, and so a theoretical comparison with the Dualize and Advance algorithm cannot be made.

10. DISCUSSION

In this article, we studied the data mining problem of searching for a set of maximally interesting sentences. This problem comes up in many data mining applications, including the well known problem of computing association rules.

We present a thorough analysis of two algorithms for the problem. These algorithms have complementary properties. The *a priori* (level-wise) algorithm is efficient as long as the rank of interesting sentences in the generalization lattice of the language is small. The algorithm is in fact optimal if we want to find all the interesting sentences. This result is also supported by recent experimental comparisons between various techniques [Zheng 2001]. The level-wise algorithm is too slow, however, if there are interesting sentences of large rank.

We introduce the Dualize and Advance algorithm that is useful in the general case. The algorithm alternates between phases of finding maximal elements, and computing the negative boundary of these elements via a transversal computation.

The analysis of the algorithm shows that the complexity of finding the most specific interesting sequences is lower than the complexity of finding all interesting sequences. The number of queries the algorithm makes to the database is essentially optimal, while the time complexity depends on the complexity of enumerating hypergraph transversals, for which the best known algorithm is mildly super-polynomial.

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 database. To the best of our knowledge, this is the only known subexponential algorithm for finding all the maximal σ-frequent sets of a relation.

A number of issues can be explored in future work. These include experiments on the trade-off between performing the transversal computation for focusing search and employing randomized search to find more most specific sentences from each seed. A particularly important issue concerns the number of passes over the database. In the description of the algorithm, we assume that the data fit in memory, but in many cases this is not so in practice. In this situation, an important consideration is to minimize the number of times we have to read the entire data from the file system into the main memory. This issue can be tackled in several ways. We can use a randomized version of algorithm AMSS (taking a random permutation p in Step 1 of algorithm AMSS) and run this algorithm a number of times at each starting point. All the randomized instantiations starting from a given point can be run at the same time during one pass over the database. Another aspect concerns the basic subroutine for going up in the lattice. In principle this procedure requires one round for each

attribute in the database (though it can be done on a large batch in one round). Recently, Bshouty [1996] introduced an algorithm for learning monotone functions in parallel that may be useful in this respect. Intuitively, the idea is that we have a large batch of points with which to go up; instead of going up one attribute at a time, we will test in each round the unions of the sets we already have, and of these will choose a subset that will give us the best leap in going up in the lattice. In this way the number of rounds is reduced considerably.

Another interesting possibility we plan to explore is to use the randomized heuristic 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 *a priori* algorithm of Agrawal and Srikant [1994] and Agrawal et al. [1996].

Finally, an extension of our theoretical framework will be useful. We assumed throughout the paper that the algorithms can only access the data via "is-interesting" queries. Clearly, more can be done with real databases. The question is whether real computational advantages can be gained by using such services. Another limitation of our results is that they only apply to problems representable as sets. One can easily generalize the Dualize and Advance algorithm for arbitrary lattices, and prove its correctness. The question is when can we take advantage of the structure so as to derive efficient algorithms using this method.

ACKNOWLEDGMENTS

We wish to thank Eyal Kushilevitz for pointing out the work in Bshouty et al. [1996], Roberto Bayardo, Rakesh Agrawal, and Christino Tamon for useful discussions, and Jordan Gergov for comments on an earlier version of the article.

REFERENCES

AGRAWAL, R. C., AGGARWAL, C. C., AND PRASAD, V. V. V. 2000. Depth first generation of long patterns. In *Knowledge Discovery and Data Mining*, pp. 108–118.

AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)* (May). ACM, New York, pp. 207–216.

AGRAWAL, R., MANNILA, H., SRIKANT, R., TOIVONEN, H., AND VERKAMO, A. I. 1996. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. AAAI Press, Menlo Park, Calif., pp. 307–328.

AGRAWAL, R. AND SRIKANT, R. 1994. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)* (Sept.). pp. 487–499.

AGRAWAL, R. AND SRIKANT, R. 1995. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)* (Taipei, Taiwan, Mar.). pp. 3–14.

ANGLUIN, D. 1988. Queries and concept learning. *Mach. Learn. 2*, 4, (Apr.), 319–342.

BAYARDO, R. J. 1998. Efficiently mining long patterns from databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York.

BELL, S. 2003. Deciding distinctness of query results by discovered constraints. Manuscript.

BELL, S. AND BROCKHAUSEN, P. 1995. Discovery of data dependencies in relational databases. Tech. Rep. LS-8 14. Universität Dortmund, Fachbereich Informatik, Lehrstuhl VIII, Künstliche Intelligenz.

BERGE, C. 1973. *Hypergraphs. Combinatorics of Finite Sets*, 3rd ed. North-Holland, Amsterdam, The Netherlands.

BIOCH, J. AND IBARAKI, T. 1995. Complexity of identification and dualization of positive Boolean functions. *Inf. Comput. 123*, 1, 50–63.

BSHOUTY, N. H. 1996. The monotone theory for the pac-model. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. ACM, New York.

BSHOUTY, N. H., CLEVE, R., GAVALDA, R., KANNAN, S., AND TAMON, C. 1996. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci. 52*, 421–433.

BURDICK, D., CALIMLIM, M., AND GEHRKE, J. 2001. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the International Conference on Data Engineering*.

EITER, T. AND GOTTLOB, G. 1995. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput. 24*, 6 (Dec.), 1278–1304.

FAYYAD, U. M., PIATETSKY-SHAPIRO, G., SMYTH, P., AND UTHURUSAMY, R., EDS. 1996. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, Calif.

FREDMAN, M. L. AND KHACHIYAN, L. 1996. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms 21*, 3 (Nov.), 618–628.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

GOUDA, K. AND ZAKI, M. J. 2001. Efficiently mining maximal frequent itemsets. In *ICDM*, pp. 163–170.

GUNOPULOS, D., KHARDON, R., MANNILA, H., AND TOIVONEN, H. 1997a. Data mining, hypergraph transversals, and machine learning. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*. ACM, New York.

GUNOPULOS, D., MANNILA, H., AND SALUJA, S. 1997b. Discovering all most specific sentences by randomized algorithms. In *Proceedings of the International Conference on Database Theory (ICDT'97)*.

HAN, J., PEI, J., AND YIN, Y. 2000. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. ACM, New York, pp. 1–12.

KAVVADIAS, D. AND STAVROPOULOS, E. C. 1999. Evaluation of an algorithm for the transversal hypergraph problem. In *Algorithm Engineering*, pp. 72–84.

KHARDON, R. 1995. Translating between Horn representations and their characteristic models. *J. Artif. Intel. Res. 3*, 349–372.

KNOBBE, A. J. AND ADRIAANS, P. W. 1995. Discovering foreign key relations in relational databases. In *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases* (Heraklion, Crete, Greece, Apr.). pp. 94–99.

LANGLEY, P. 1995. *Elements of Machine Learning*. Morgan-Kaufmann, San Mateo, Calif.

LIN, D.-I. AND KEDEM, Z. M. 1998. Pincer search: A new algorithm for discovering the maximum frequent set. In *Extending Database Technology*, pp. 105–119.

MANNILA, H. 1995. Aspects of data mining. In *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases* (Heraklion, Crete, Greece, Apr.). pp. 1–6.

MANNILA, H. 1996. Data mining: Machine learning, statistics, and databases. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management* (Stockholm, Sweden). pp, 2–9.

MANNILA, H. AND RÄIHÄ, K.-J. 1986. Design by example: An application of Armstrong relations. *J. Comput. Syst. Sci. 33*, 2, 126–141.

MANNILA, H. AND RÄIHÄ, K.-J. 1992. *Design of Relational Databases.* Addison-Wesley, Wokingham, U.K.

MANNILA, H. AND RÄIHÄ, K.-J. 1994. Algorithms for inferring functional dependencies. *Data Knowl. Eng. 12*, 1 (Feb.), 83–99.

MANNILA, H. AND TOIVONEN, H. 1997. Levelwise search and borders of theories in knowledge discovery. *Data Mining Knowl. Disc. 1*, 3, 241–258.

MANNILA, H., TOIVONEN, H., AND VERKAMO, A. I. 1994. Efficient algorithms for discovering association rules. In *Knowledge Discovery in Databases, Papers from the 1994 AAAI Workshop (KDD'94)* (Seattle, Wash., July), pp. 181–192.

MANNILA, H., TOIVONEN, H., AND VERKAMO, A. I. 1995. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95)* (Montreal, Ont., Canada, Aug.). AAAI Press, pp. 210–215.

MISHRA, N. AND PITT, L. 1997. Generating all maximal independent sets of bounded-degree hypergraphs. In *Proceedings of the Conference on Computational Learning Theory (COLT)*. pp. 211–217.

MITCHELL, T. M. 1982. Generalization as search. *Artif. Intel. 18*, 203–226.

RYMON, R. 1992. Search through systematic set enumeration. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

SCHLIMMER, J. 1993. Using learned dependencies to automatically construct sufficient and sensible editing views. In *Knowledge Discovery in Databases, Papers from the 1993 AAAI Workshop (KDD'93)* (Washington, D.C.). AAAI Press, pp. 186–196.

ULLMAN, J. D. 1988. *Principles of Database and Knowledge-Base Systems*, vol. I. Computer Science Press, Rockville, Md.

VALIANT, L. G. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput. 8*, 3, 410–421.

ZHENG, Z., KOHAVI, R., AND MASON, L. 2001. Real world performance of association rule algorithms. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York.

Received March 2001; revised April 2002, November 2002, February 2003; accepted February 2003