Methods and problems in data mining*

Heikki Mannila**

Department of Computer Science
University of Helsinki,
FIN-00014 Helsinki, Finland
E-mail: Heikki.Mannila@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/~mannila/

Abstract. *Knowledge discovery in databases* and *data mining* aim at semiautomatic tools for the analysis of large data sets. We consider some methods used in data mining, concentrating on levelwise search for all frequently occurring patterns. We show how this technique can be used in various applications. We also discuss possibilities for compiling data mining queries into algorithms, and look at the use of sampling in data mining. We conclude by listing several open research problems in data mining and knowledge discovery.

1 Introduction

Knowledge discovery in databases (KDD), often called data mining, aims at the discovery of useful information from large collections of data. The discovered knowledge can be rules describing properties of the data, frequently occurring patterns, clusterings of the objects in the database, etc. Current technology makes it fairly easy to collect data, but data analysis tends to be slow and expensive. There is a suspicion that there might be nuggets of useful information hiding in the masses of unanalyzed or underanalyzed data, and therefore semiautomatic methods for locating interesting information from data would be useful. Data mining has in the 1990's emerged as visible research and development area; see [11] for a recent overview of the area.

This tutorial describes some methods of data mining and also lists a variety of open problems, both in the theory of data mining and in the systems side of it. We start in Section 2 by briefly discussing the KDD process, basic data mining techniques, and listing some prominent applications.

Section 3 moves to a generic data mining algorithm, and discusses some of the architectural issues in data mining systems. Section 4 considers the specific problems of finding association rules, episodes, or keys from large data sets.

* To appear in the proceedings of International Conference on Database Theory, Delphi, Greece, January 1997, F. Afrati and P. Kolaitis (ed.), Springer-Verlag

** Part of this work was done while the author was visiting the Max Planck Institut für Informatik in Saarbrücken, Germany. Work supported by the Academy of Finland and by the Alexander von Humboldt Stiftung.

In Section 5 we consider the possibilities of specifying KDD tasks in a high level language and compiling these specifications to efficient discovery algorithms. Section 6 studies the use of sampling in data mining, and Section 7 considers the possibilities of representing large data sets by smaller condensed representations, Section 8 gives a list of open problems in data mining.

Before starting on the KDD process, I digress briefly to other some topics not treated in this paper. An important issue in data mining is its relationship with machine learning and statistics. I refer to [11, 31] for some discussions on this. *Visualization* of data is an important technique for obtaining useful information from large masses of data. The area is large; see [25] for an overview. Visualization can also be useful for making the discovered patterns easier to understand.

Clustering is obviously a central technique in analyzing large data collections. The literature on the area is huge, and too wide to even scratch here.

Similarity searches are often needed in data mining applications: how does one find objects that are roughly similar to the a query point. Again, the literature is vast, and we provide only two recent pointers: [4, 49].

2 The KDD process

The goal of knowledge discovery is to obtain useful knowledge from large collections of data. Such a task is inherently interactive and iterative: one cannot expect to obtain useful knowledge simply by pushing a lot of data to a black box. The user of a KDD system has to have a solid understanding of the domain in order to select the right subsets of data, suitable classes of patterns, and good criteria for interestingness of the patterns. Thus KDD systems should be seen as interactive tools, not as automatic analysis systems.

Discovering knowledge from data should therefore be seen as a process containing several steps:

1. understanding the domain,
2. preparing the data set,
3. discovering patterns (data mining),
4. postprocessing of discovered patterns, and
5. putting the results into use.

See [10] for a slightly different process model and excellent discussion.

The KDD process is necessarily iterative: the results of a data mining step can show that some changes should be made to the data set formation step, postprocessing of patterns can cause the user to look for some slightly modified types of patterns, etc. Efficient support for such iteration is one important development topic in KDD.

Prominent applications of KDD include health care data, financial applications, and scientific data [39, 30]. One of the more spectacular applications is the SKICAT system [9], which operates on 3 terabytes of image data, producing a classification of approximately 2 billion sky objects into a few classes. The task is obviously impossible to do manually. Using example classifications

provided by the users, the system learns classification rules that are able to do the categorization accurately and fast.

In industry, the success of KDD is partly related to the rise of the concepts of data warehousing and on-line analytical processing (OLAP). These strategies for the storage and processing of the accumulated data in an organization have become popular in recent years. KDD and data mining can be viewed as ways of realizing some of the goals of data warehousing and OLAP.

3 A generic data mining algorithm and architecture

A fairly large class of data mining tasks can be described as the search for interesting and frequently occurring patterns from the data. That is, we are given a class $\mathcal{P}$ of patterns or sentences that describe properties of the data, and we can specify whether a pattern $p \in \mathcal{P}$ occurs frequently enough and is otherwise interesting. That is, the generic data mining task is to find the set

$$PI(d, \mathcal{P}) = \{p \in \mathcal{P} \mid p \text{ occurs sufficiently often in } d \text{ and } p \text{ is interesting}\}.$$

An alternative formalism would be to consider a language $\mathcal{L}$ of sentences and view data mining as the problem of finding the sentences in $\mathcal{L}$ that are "sufficiently true" in the data and furthermore fulfill the user's other criteria for interestingness. This point of view has either implicitly or explicitly been used in discovering integrity constraints from databases, in inductive logic programming, and in machine learning [6, 7, 26, 30, 32]; some theoretical results can be found in [37], and a suggested logical formalism in [23].

While the frequency of occurrence of a pattern or the truth of a sentence can defined rigorously, the interestingness of patterns or sentences seems much harder to specify and measure.

A general algorithm for finding $PI(d, \mathcal{P})$ is to first compute all frequent patterns by the following algorithm *Find-frequent-patterns*, and then select the interesting ones from the output.

Algorithm 1 FFP: Finding all frequent patterns. Assume that there is an ordering $<$ defined between the patterns of $\mathcal{P}$.

1. $\mathcal{C} := \{p \in \mathcal{P} \mid$ for no $q \in \mathcal{P}$ we have $q < p\}$;
   – $\mathcal{C}$ contains the initial patterns from $\mathcal{P}$;
2. **while** $\mathcal{C} \neq \emptyset$ **do**
3. **for** each $p \in \mathcal{C}$
4. find the number of occurrences of p in d;
5. $\mathcal{F} := \mathcal{F} \cup \{p \in \mathcal{C} \mid p$ is sufficiently frequent in $d\}$;
6. $\mathcal{C} := \{p \in \mathcal{P} \mid$ all $q \in \mathcal{P}$ with $q < p$ have been considered already and it is possible that p is frequent$\}$;
7. **od**;
8. output $\mathcal{F}$.

The algorithm proceeds by first investigating the initial patterns with no predecessors in the ordering $<$. Then, the information about frequent patterns is used to generate new candidates, i.e., patterns that could be frequent on the basis of the current knowledge.

In the next section we show how this algorithm can be used to solve several data mining problems. If line 6 is instantiated differently, hill-climbing searches for best descriptions [19, 30] can also be fitted into this framework. In hill-climbing, the set $\mathcal{C}$ will contain only the neighbors of the current "most interesting" pattern.

The generic algorithm suggests a data mining system architecture consisting of a discovery module and a database management system. The discovery module sends queries to the database, and the database answers. The queries are typically of the form "How many objects in the database match p", where p is a possibly interesting pattern; the database answers by giving the count.

If implemented naively, this architecture leads to slow operations. To achieve anything resembling the efficiency of tailored solutions, the database management system should be able to utilize the strong similarities between the queries generated by the discovery module.

The view of data mining as locating frequently occurring and interesting patterns from data suggests that data mining can benefit from the extensive research done in the area of *combinatorial pattern matching* (CPM); see, e.g., [14]. One can even state the following *CPM principle of data mining*:

> *It is better to use complicated primitive patterns and simple logical combinations than simple primitive patterns and complex logical form.*

4 Examples

In this section we discuss three data mining problems where instantiations of the above algorithm can be used.

4.1 Association rules

Given a schema $R = \{A_1, \ldots, A_p\}$ of attributes with domain $\{0, 1\}$, and a relation r over R, an *association rule* [1] about r is an expression of the form $X \Rightarrow B$, where $X \subseteq R$ and $B \in R \setminus X$. The intuitive meaning of the rule is that if a row of the matrix r has a 1 in each column of X, then the row tends to have a 1 also in column B.

Examples of data where association rules might be applicable include the following.

- A student database at a university: rows correspond to students, columns to courses, and a 1 in entry (s, c) indicates that the student s has taken course c.
- Data collected from bar-code readers in supermarkets: columns correspond to products, and each row corresponds to the set of items purchased at one time.

- A database of publications: the rows and columns both correspond to publications, and $(p, p') = 1$ means that publication p refers to publication p'.
- A set of measurements about the behavior a system, say exchanges in a telephone network. The columns correspond to the presence or absence of certain conditions, and each row correspond to a measurement: if entry (m, c) is 1, then at measurement m condition c was present.

Given $W \subseteq R$, we denote by $s(W, r)$ the *frequency* of W in r: the fraction of rows of r that have a 1 in each column of W. The *frequency* of the rule $X \Rightarrow B$ in r is defined to be $s(X \cup \{B\}, r)$, and the *confidence* of the rule is $s(X \cup \{B\}, r)/s(X, r)$.

In the discovery of association rules, the task is to find all rules $X \Rightarrow B$ such that the frequency of the rule is at least a given threshold σ and the confidence of the rule is at least another threshold θ. In large retailing applications the number of rows might be 10^6 or even 10^8, and the number of columns around 5000. The frequency threshold σ typically is around 10^{-2} — 10^{-4}. The confidence threshold θ can be anything from 0 to 1. From such a database one might obtain thousands or hundreds of thousands of association rules. (Of course, one has to be careful in assigning any statistical significance to findings obtained from such methods.)

Note that there is no predefined limit on the number of attributes of the left-hand side X of an association rule $X \Rightarrow B$, and B is not fixed, either; this is important so that unexpected associations are not ruled out before the processing starts. It also means that the search space of the rules has exponential size in the number of attributes of the input relation. Handling this requires some care for the algorithms, but there is a simple way of pruning the search space.

We call a subset $X \subseteq R$ *frequent* in r, if $s(X, r) \geq \sigma$. Once all frequent sets of r are known, finding the association rules is easy. Namely, for each frequent set X and each $B \in X$ verify whether the rule $X \setminus \{B\} \Rightarrow B$ has sufficiently high confidence.

How can one find all frequent sets X? This can be done in a multitude of ways [1, 2, 16, 18, 43, 48]. A typical approach [2] is to use that fact that all subsets of a frequent set are also frequent. A way of applying the framework of Algorithm *Find-frequent-patterns* is as follows.

First find all frequent sets of size 1 by reading the data once and recording the number of times each attribute A occurs. Then form *candidate* sets of size 2 by taking all pairs $\{B, C\}$ of attributes such that $\{B\}$ and $\{C\}$ both are frequent. The frequency of the candidate sets is again evaluated against the database. Once frequent sets of size 2 are known, candidate sets of size 3 can be formed; these are sets $\{B, C, D\}$ such that $\{B, C\}$, $\{B, D\}$, and $\{C, D\}$ are all frequent. This process is continued until no more candidate sets can be formed.

As an algorithm, the process is as follows.

Algorithm 2 Finding frequent sets.

1. $\mathcal{C} := \{\{A\} \mid A \in R\}$;
2. $\mathcal{F} := \emptyset$;
3. $i := 1$;

4. **while** $\mathcal{C} \neq \emptyset$ **do**
5. $\mathcal{F}' :=$ the sets $X \in \mathcal{C}$ that are frequent;
6. add $\mathcal{F}'$ to $\mathcal{F}$;
7. $\mathcal{C} :=$ sets Y of size $i+1$ such that
8. each subset W of Y of size i is frequent;
9. $i := i+1$;
10. **od**;

The algorithm has to read the database at most $K+1$ times, where K is the size of the largest frequent set. In the applications, K is small, typically at most 10, so the number of passes through the data is reasonable.

A modification of the above method is obtained by computing for each frequent set X the subrelation $r_X \subseteq r$ consisting of those rows $t \in r$ such that $t[A] = 1$ for all $A \in X$. Then it is easy to see that for example $r_{\{A,B,C\}} = r_{\{A,B\}} \cap r_{\{B,C\}}$. Thus the relation r_X for a set X of size k can be obtained from the relations $r_{X'}$ and $r_{X''}$, where $X' = X \setminus \{A\}$ and $X'' = X \setminus \{B\}$ for some $A, B \in X$ with $A \neq B$. This method has the advantage that rows that do not contribute to any frequent set will not be inspected more than once. For comparisons of the two approaches, see [2, 18, 43].

The algorithms described above work quite nicely on large input relations. Their running time is approximately $O(NF)$, where $N = np$ is the size of the input and F is the sum of the sizes of the sets in the candidate collection $\mathcal{C}$ during the operation of the algorithm [37]. This is nearly linear, and the algorithms seem to scale nicely to tens of millions of examples. Typically the only case when they fail is when the output is too large, i.e., there are too many frequent sets.

The methods for finding frequent sets are simple: they are based on one nice but simple observation (subsets of frequent sets must be frequent), and use straightforward implementation techniques.

A naive implementation of the algorithms on top of a relational database system would be easy: we need to pose to the database management system queries of the form "What is $s(\{A_1, \ldots, A_k\}, r)$?", or in SQL

```
select count(*) from r t
where t[A_1] = 1 and ··· and t[A_k] = 1
```

The number of such queries can be large: if there are thousands of frequent sets, there will be thousands of queries. The overhead in performing the queries on an ordinary DBMS would probably be prohibitive.

The customized algorithms described above are able to evaluate masses of such queries reasonably efficiently, for several reasons. First, all the queries are very simple, and have the same general form; thus there is no need to compile each query individually. Second, the algorithms that make repeated passes through the data evaluate a large collection of queries during a single pass. Third, the algorithm that build the relations r_X for frequent sets X use the results of previous queries to avoid looking at the whole data for each query.

Association rules are a simple formalism and they produce nice results for binary data. The basic restriction is that the relation should be sparse in the

sense that there are no frequent sets that contain more than about 15 attributes. Namely, the framework of finding all association rules generates typically at least as many rules as there are frequent sets, and if there is a frequent set of size K, there will be at least 2^K frequent sets.

The information about the frequent sets can actually be used to approximate fairly accurately the confidences and supports of a far wider set of rules, including negation and disjunction [36].

4.2 Finding episodes from sequences

The basic ideas of the algorithm for finding association rules are fairly widely applicable. In this section we describe an application of the same ideas to the problem of finding repeated *episodes* in sequences of events.

Consider a sequence of events (e, t), where e is the type of the event and t is the time when the event occurred. Such data is routinely collected in, e.g., telecommunication networks, process monitoring, quality control, user interface studies, and epidemiology. There is an extensive statistical literature on how such data can be analyzed, but most methods are suited only for small numbers of event types.

For example, a telecommunications network alarm database is used to collect all the notifications about abnormal situations in the network. The number of event types is around 200, and there are 1000-10000 alarms per day [17].

As a first step in analyzing such data, one can try to find which event types occur frequently close together. Denoting by E the set of all event types, an *episode* φ is a partially ordered set of elements from E. An episode might, e.g., state that events of type A and B occur (in either order) before an event of type C.

Given an alarm sequence $r = (e_1, t_1), \ldots, (e_n, t_n)$, a *slice* r_t of r of width W consists of those events (e_i, t_i) of r such that $t \leq t_i \leq t + W$. An episode φ *occurs* in r_t, if there are events in r_t corresponding to the event types of φ and they occur in an order respecting the partial order of φ. An episode is *frequent*, if it occurs in sufficiently many slices of the original sequence.

How to find all episodes from a long sequence of events? Using the same idea as before, we first locate frequent episodes of size 1, then use these to generate candidate episodes of size 2, verify these against the data, generate candidate episodes of size 3, etc. The algorithm can be further improved by using incremental recognition of episodes; see [38] for details, and [35] for extensions with logical variables etc. The results are good: the algorithms are efficient, and using them one can find easily comprehensible results about the combinations of event types that occur together. See also [42] for a temporal logic approach to this area.

4.3 Finding keys or functional dependencies

The key finding problem is: given a relation r, find all minimal keys of r. It is a special case of the problem of finding the functional dependencies that hold

in a given relation. Applications of the key finding problem include database design, semantic query optimization [24, 44, 46]; one can also argue that finding functional dependencies is a necessary step in some types of structure learning.

The size of an instance of the key finding problem is given by two parameters: the number of rows, and the number of columns. In the typical database applications the n, the number of rows, can be thousands or millions. The number of attributes p is typically somewhere from 5 to 50. However, for some data mining applications, p could easily be 1000.

While the problem of finding the keys of a relation is simple to state, its algorithmic properties turn out to be surprising complex. See [33, 34] for a variety of results, and Section 8 for theoretically intriguing open problems.

The algorithm for finding $PI(d, p)$ in Section 3 can straightforwardly be applied to finding the keys of a relation. The patterns are sets of attributes. A pattern $X \subseteq R$ is frequent, if X is a superkey, and the relation $<$ between patterns is the converse of set inclusion.

In the first two examples in this section algorithm *Find-frequent-patterns* produced good results. However, for finding keys this method is not particularly suitable, since the part of the pattern space that has to be inspected can be huge, as a key can be very far from the starting point in the ordering $<$. Several suggested algorithms try to jump in the subset lattice to avoid looking at all superkeys.

5 KDD queries and their compilation

Viewing data mining as computing $PI(d, p)$ is based on the methodological idea of first generating lots of rules or patterns, and then letting the user select the truly interesting ones from these. The advantage of this approach is that one does not have to go back to the data every time the user finds a new topic of interest. The disadvantage is that if the user is interested in only a very tightly specified set of rules or patterns, finding a far wider set of rules can be quite wasteful.

To make the idea of generating rules and selecting interesting ones from them work, one has to provide the user methods and tools for selecting, ordering, and grouping of rules. See [29, 30] for some work along these lines. Many data mining systems try to do the pruning of uninteresting rules while the rules are located; it seems to me that the user's needs are so hard to predict that an automatic selection of interesting patterns is not easy. Still, if the user has a fixed set of interesting attributes in mind, or otherwise has some specific knowledge of the patterns that are needed, it would be wasteful to avoid using this information.

Currently, data mining research and development consists mainly of isolated applications. One can even argue that data mining is today at the same state as data management was in the 1960's [20, 21]: then all data management applications were *ad hoc*; only the advent of the relational model and powerful query languages made it possible to develop applications fast. Consequently, data mining would need a similar theoretical framework.

The approach of computing $PI(d, \mathcal{P})$ presented in the previous section provides one possible framework. A data mining task could be given by specifying the class $\mathcal{P}$ and the interestingness predicate. That is, the user of a KDD system makes a *pattern query*. Mimicking the behavior of an ordinary DBMS, the KDD system compiles and optimizes the pattern query, and executes it by searching for the patterns that satisfy the user's specifications and have sufficient frequency in the data.

As an example, consider the simple case of mining for association rules in a course enrollment database. The user might say that he/she is interested only in rules that have the "Data Management" course on the left-hand side. This restriction can be utilized in the algorithm for finding frequent sets: only candidate sets that contain "Data Management" need to be considered.

Developing methods for such pattern queries and their optimization is currently one of the most interesting research topics in data mining. So far, even the simple techniques such as the above have not been sufficiently studied. It is not clear how far one can get by using such methods, but the possible payoff is large.

Such queries have some similarities with the strategies adopted in OLAP. The difference is mainly that OLAP is verification-driven, in the sense that the questions are fairly well fixed in advance. Data mining, on the other hand, is discovery-driven: one does not want to specify in advance what exactly is searched for.

In addition to developing query processing strategies for data mining applications, changes in the underlying storage model can also have a strong effect on the performance. A very interesting experiment in this direction is the work on the Monet database server developed at CWI in the Netherlands by Martin Kersten and others [5, 19]. The Monet system is based on the vertical partitioning of the relations: a relation with k attributes is decomposed into k relations, each with two attributes: the OID and one of the original attributes. The system is built on the extensive use of main memory, has an extensible set of basic operations, and supports shared-memory parallelism. Experiments with Monet on data mining applications have produced quite good results [18, 19].

6 Sampling

Data mining is often difficult for at least two reasons: first, there are lots of data, and second, the data is multidimensional. The hypothesis or pattern space is in most cases exponential in the number of attributes, so the multidimensionality can actually be the harder problem.

A simple way of alleviating the problems caused by the volume of data (i.e., the number of rows) is to use sampling. Even small samples can give quite good approximation to the association rules [2, 48] or functional dependencies [28] that hold in a relation. See [27] for a general analysis on the relationship between the logical form of the discovered knowledge and the sample sizes needed for discovering it.

The problem with using sampling is that the results can be wrong, with a small probability. A possibility is to first use a sample and then verify (and, if

necessary, correct) the results against the whole data set. For instances of this scheme, see [28, 48]; also the generic algorithm can be modified to correspond to this approach. We give the sample-and-correct algorithm for finding functional dependencies.

Algorithm 3 Finding the keys of a relation by sampling and correcting.
Input. A relation r over schema R.
Output. The set of keys of r.
Method.

1. $s :=$ a sample of r;
2. $\mathcal{K} := keys(s)$;
3. **while** there is a set $X \in \mathcal{K}$ such that X is not a key of r **do**
4. add some rows $u, v \in r$ with $u[X] = v[X]$ to s;
5. $\mathcal{K} := keys(s)$;
6. **od**;
7. output $\mathcal{K}$.

7 Condensed representations

We remarked in Section 2 that KDD is an iterative process. Once a data mining algorithm has been used to discover potentially interesting patterns, the user often wants to view these patterns in different ways, have a look at the actual data, visualize the patterns, etc. A typical phenomenon is also that some pattern p looks interesting, and the user wants to evaluate other patterns that closely resemble p. In implementing such queries, caching of previous results is obviously useful. Still, having to go back to the original data each time the user wants some more information seems somewhat wasteful. Similarly, in the generic data mining algorithm presented in Section 3 the frequency and interestingness of each pattern are verified against the database. It would be faster to look at some sort of short representation of the data.

Given a data collection $d \in \mathcal{D}$, and a class of patterns $\mathcal{P}$, a *condensed representation* for d and $\mathcal{P}$ is a data structure that makes it possible to answer queries of the form "How many times does $p \in \mathcal{P}$ occur in d" approximately correctly and more efficiently than by looking at d itself.

A simple example of a condensed representation is obtained by taking a sample from the data: by counting the occurrences of the pattern in the sample, one gets an approximation of the number of occurrences in the original data. Another, less obvious example is given by the collection of frequent sets of a 0-1 valued relation [36]: the collection of frequent sets can be used to give approximate answers to arbitrary boolean queries about the data, even though the frequent sets represent only conjunctive concepts. The data cube [15] can also be viewed as a condensed representation for a class of queries. Similarly, in computational geometry the notion of an ε-approximation [41] is closely related.

Developing condensed representations for various classes of patterns seems a promising way of improving the effectiveness of data mining algorithms. Whether this approach is generally useful is still open.

8 Open problems

Data mining is an applied field, and research in the area seems to succeed best when done in cooperation with the appliers. Hence I am a little hesitant to offer a list of research questions or open problems; on the other hand, it seems to me that such a list gives a reasonable sample of the research issues in the area.

Thus, this section contains a list of research topics in data mining that I consider interesting or important. The problems are very varying, from architectural issues to specific algorithmic questions. For brevity, the descriptions are quite succinct, and I also provide only a couple of references.

Framework and general theory

1. Develop a general theory of data mining. Possible starting points are [6, 7, 23, 26, 30, 37]. (One might call this *the theory of inductive databases.*)
2. What is the relationship between the logical form of sentences to be discovered and the computational complexity of the discovery task? (The issue of logical form vs. sample size is considered in [27].)
3. Prove or disprove the CPM principle (Section 3).
4. What can be said about the performance of Algorithm 3 and its analogues for other problems?
5. How useful is the concept of condensed representation? [36]

System and language issues

6. What is a good architecture of a data mining system? How should a database management system and the search modules be connected? [19, 3]
7. Develop a language for expressing KDD queries and techniques for optimizing such queries. Some suggestions are given in [22, 45, 40].
8. A subproblem of the previous one: how can caching strategies help in processing sequences of related queries? [19]
9. Extend the association rule framework to handle attributes with continuous values. (Some partial solutions to this problem are given in [12, 13, 47].)
10. Investigate the usefulness of temporal databases in the mining of event sequence data.
11. Develop tools for selecting, grouping, and visualizing discovered knowledge [29, 30]. How can background knowledge be used?

Algorithmic open problems

12. Design an algorithm for the key finding problem that works in polynomial time with respect to the size of the output and the number of attributes, and in subquadratic time in the number of rows. Solutions to the two following problems would imply considerable progress for this problem.

13. Finding the keys of a relation contains as a subproblem the problem of finding transversals of hypergraphs [33, 8, 36]. Given a hypergraph $\mathcal{H}$, can the set $Tr(\mathcal{H})$ of its transversals be computed in time polynomial in $|\mathcal{H}|$ and $|Tr(\mathcal{H})|$?
14. When one reduces the problem of finding keys to transversals of hypergraphs, one has to solve the following preliminary problem. Given a relation r over schema R and two rows $u, v \in r$, denote $ag(u,v) = \{A \in R \mid u[A] = v[A]\}$, and let $disag(u,v) = R \setminus ag(u,v)$. Denote $ag(r) = \{ag(u,v) \mid u, v \in r, u \neq v\}$, and $disag(r) = \{disag(u,v) \mid u, v \in r, u \neq v\}$. Further, let $maxag(r)$ be the collection of maximal elements of $ag(r)$ and $mindisag(r)$ the collection of minimal elements of $disag(r)$. Given a relation r, compute $mindisag(r)$ in time $O(q(|mindisag(r)|)f(|r|))$, where q is a polynomial and $f = o(|r|^2)$.
15. In the analysis of event sequences, patterns are typically subsequences. Given a sequence $s = a_1 \cdots a_n$, a pattern $p = b_1 \cdots b_k$ and a window width W, decide in $o(nk)$ time whether p occurs as a subsequence in s within a window of width W, i.e., whether there are indices $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that $i_k - i_1 \leq W$ and for all $j = 1, \ldots, k$ we have $a_{i_j} = b_j$. [38, 35] (The solution should work for very large alphabets.)

Acknowledgements

Comments from Dimitrios Gunopulos and Hannu Toivonen are gratefully acknowledged.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*, pages 207 – 216, May 1993.
2. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.
3. R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, pages 287–290, 1996.
4. S. Berchtold, D. A. Keim, and H. P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 28–29, Mumbay, India, 1996. Morgan Kaufmann.
5. P. A. Boncz, W. Quak, and M. L. Kersten. Monet and its geographical extensions: a novel approach to high-performance GIS processing. In P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Advances in Database Technology - EDBT'96*, pages 147–166, 1996.
6. L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI–93)*, pages 1058 – 1053, Chambéry, France, 1993. Morgan Kaufmann.

7. L. De Raedt and S. Džeroski. First-order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375 – 392, 1994.
8. T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278 – 1304, Dec. 1995.
9. U. M. Fayyad, S. G. Djorgovski, and N. Weir. Automating the analysis and cataloging of sky surveys. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 471 – 494. AAAI Press, Menlo Park, CA, 1996.
10. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1 –34. AAAI Press, Menlo Park, CA, 1996.
11. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA, 1996.
12. T. Fukuda et al. Data mining using two-dimensional optimized association rules: Scheme, algorithms, visualization. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'96)*, pages 13–23, 1996.
13. T. Fukuda et al. Mining optimized association rules for numeric attributes. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'96)*, 1996.
14. Z. Galil and E. Ukkonen, editors. *6th Annual Symposium on Combinatorial Pattern Matching (CPM 95)*, volume 937 of *Lecture Notes in Computer Science*, Berlin, 1995. Springer.
15. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *12th International Conference on Data Engineering (ICDE'96)*, pages 152 – 159, New Orleans, Louisiana, Feb. 1996.
16. J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 420 – 431, Zurich, Swizerland, 1995.
17. K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. Knowledge discovery from telecommunication network alarm databases. In *12th International Conference on Data Engineering (ICDE'96)*, pages 115 – 122, New Orleans, Louisiana, Feb. 1996.
18. M. Holsheimer, M. Kersten, H. Mannila, and H. Toivonen. A perspective on databases and data mining. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 150 – 155, Montreal, Canada, Aug. 1995.
19. M. Holsheimer, M. Kersten, and A. Siebes. Data surveyor: Searching the nuggets in parallel. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 447 – 467. AAAI Press, Menlo Park, CA, 1996.
20. T. Imielinski. A database view on data mining. Invited talk at the KDD'95 conference.
21. T. Imielinski and H. Mannila. Database mining: a new frontier. *Communications of the ACM*, 1996. To appear.
22. T. Imielinski and A. Virmani. M-sql: Query language for database mining. Technical report, Rutgers University, January 1996.

23. M. Jaeger, H. Mannila, and E. Weydert. Data mining as selective theory extraction in probabilistic logic. In R. Ng, editor, *SIGMOD'96 Data Mining Workshop, The University of British Columbia, Department of Computer Science, TR 96-08*, pages 41–46, 1996.
24. M. Kantola, H. Mannila, K.-J. Räihä, and H. Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7(7):591 – 607, Sept. 1992.
25. D. Keim and H. Kriegel. Visualization techniques for mining large databases: A comparison. *IEEE Transactions on Knowledge and Data Engineering*, 1996. to appear.
26. J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*, pages 335 – 359. Academic Press, London, 1992.
27. J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'94)*, pages 77 – 85, Minneapolis, MN, May 1994.
28. J. Kivinen and H. Mannila. Approximate dependency inference from relations. *Theoretical Computer Science*, 149(1):129 – 149, 1995.
29. M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 401 – 407, Gaithersburg, MD, Nov. 1994. ACM.
30. W. Kloesgen. Efficient discovery of interesting statements in databases. *Journal of Intelligent Information Systems*, 4(1):53 – 69, 1995.
31. H. Mannila. Data mining: machine learning, statistics, and databases. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management, Stockholm*, pages 1–6, 1996.
32. H. Mannila and K.-J. Räihä. Design by example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 33(2):126 – 141, 1986.
33. H. Mannila and K.-J. Räihä. *Design of Relational Databases.* Addison-Wesley Publishing Company, Wokingham, UK, 1992.
34. H. Mannila and K.-J. Räihä. On the complexity of dependency inference. *Discrete Applied Mathematics*, 40:237 – 243, 1992.
35. H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 146 – 151, Portland, Oregon, Aug. 1996. AAAI Press.
36. H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 189 – 194, Portland, Oregon, Aug. 1996. AAAI Press.
37. H. Mannila and H. Toivonen. On an algorithm for finding all interesting sentences. In *Cybernetics and Systems, Volume II, The Thirteenth European Meeting on Cybernetics and Systems Research*, pages 973 – 978, Vienna, Austria, Apr. 1996.
38. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210 – 215, Montreal, Canada, Aug. 1995.

39. C. J. Matheus, G. Piatetsky-Shapiro, and D. McNeill. Selecting and reporting what is interesting. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 495 – 515. AAAI Press, Menlo Park, CA, 1996.
40. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, 1996. To appear.
41. K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, New York, 1993.
42. B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 351–354, 1996.
43. A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 432 – 444, Zurich, Swizerland, 1995.
44. J. Schlimmer. Using learned dependencies to automatically construct sufficient and sensible editing views. In *Knowledge Discovery in Databases, Papers from the 1993 AAAI Workshop (KDD'93)*, pages 186 – 196, Washington, D.C., 1993.
45. W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI Press, Menlo Park, CA, 1996.
46. M. Siegel. Automatic rule derivation for semantic query optimization. Technical Report BUCS Tech Report # 86-013, Boston University, Computer Science Department, Dec. 1986.
47. R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'96)*, pages 1–12, Montreal, Canada, 1996.
48. H. Toivonen. Sampling large databases for association rules. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 134 – 145, Mumbay, India, Sept. 1996. Morgan Kaufmann.
49. D. A. White and R. Jain. Algorithms and strategies for similarity retrieval. Technical Report VCL-96-101, Visual Computing Laboratory, University of California, San Diego, 9500 Gilman Drive, Mail Code 0407, La Jolla, CA 92093-0407, July 1996.

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