University of Helsinki
Department of Computer Science
Series of Publications C, No. C-1993-65

Improved Methods for Finding Association Rules

Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo

Helsinki, December 1993 (Revised February 1994)

Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland

Improved Methods for Finding Association Rules

Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo

University of Helsinki, Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, FINLAND
E-mail: {mannila, htoivone, verkamo}@cs.Helsinki.FI
Report C-1993-65, December 1993 (Revised February 1994), 20 pages

Abstract

Association rules are statements of the form "for 90 % of the rows of the relation, if the row has value 1 in the columns in set W, then it has 1 also in column B". Agrawal, Imielinski, and Swami introduced the problem of mining association rules from large collections of data, and gave a method based on successive passes over the database. We give an improved algorithm for the problem. The method is based on careful combinatorial analysis of the information obtained in previous passes; this makes it possible to eliminate unnecessary candidate rules. Experiments on a university course enrollment database indicate that the method outperforms the previous one by a factor of 5. We also give simple information-theoretic lower bounds for the problem of finding association rules, and show that sampling is in general a very efficient way of finding such rules.

Computing Reviews Categories and Subject Descriptors:
H.3.3 [Information Systems]: Information Storage and Retrieval – Information Search and Retrieval
I.2.6 [Computing Methodologies]: Artificial Intelligence – Learning
I.2.8 [Computing Methodologies]: Artificial Intelligence – Problem Solving, Control Methods, and Search

General Terms:
Databases, machine learning, artificial intelligence.

Additional Key Words and Phrases:
Database mining, knowledge discovery in databases, association rules, covering sets.

1 Introduction

Data mining (database mining, knowledge discovery in databases) has recently been recognized as a promising new field in the intersection of databases, artificial intelligence, and machine learning (see, e.g., [PSF91]). The area can be loosely defined as finding interesting rules or exceptions from large collections of data.

Recently, Agrawal, Imielinski, and Swami introduced a class of regularities, *association rules*, and gave an algorithm for finding such rules from a database with binary data [AIS93]. An association rule is an expression $W \Rightarrow B$, where W is a set of attributes and B a single attribute. The intuitive meaning of such a rule is that in the rows of the database where the attributes in W have value true, also the attribute B tends to have value true. For instance, a rule might state that students taking courses CS101 and CS120, often also take the course CS130. This sort of information can be used, e.g., in assigning classrooms for the courses. Applications of association rules include customer behavior analysis for example in a supermarket or banking environment, and telecommunications alarm diagnosis and prediction.

In this paper we study the properties of association rule discovery in relations. We give a new algorithm for the problem that outperforms the method in [AIS93] by a factor of 5. The algorithm is based on the same basic idea of repeated passes over the database as the method in [AIS93]. The difference is that our algorithm makes careful use of the combinatorial information obtained from previous passes and in this way avoids considering many unnecessary sets in the process of finding the association rules. Our experimental data consists of two databases, namely university course enrollment data and the fault management database of a switching network. The empirical results show a good, solid performance for our method.

We also study the theoretical properties of the problem of finding the association rules that hold in a relation. We give a probabilistic analysis of two aspects of the problem, showing that sampling is an efficient way of finding association rules, and that in random relations almost all association rules are small. We also give a simple information-theoretic lower bound for finding one rule, and show that an algorithm suggested by Loveland in [Lov87] in a different framework actually meets this lower bound.

The rest of this paper is organized as follows. Section 2 introduces the problem and the notations. Section 3 describes our algorithm for finding association rules. The analysis of sampling is given in Section 4. Empirical results and a comparison to the results of [AIS93] are given in Section 5. Section 6 shortly discusses association rules in sequential data. Related work is presented in Section 7. Section 8 is a short conclusion. Appendix A contains the probabilistic analyses of random relations and the lower bound result. Appendix B gives an overview of the implementation.

2 Problem

First we introduce some basic concepts, using the formalism presented in [AIS93]. Let $R = \{I_1, I_2, \ldots, I_m\}$ be a set of attributes, also called items, over the binary domain $\{0,1\}$. The input $r = \{t_1, \ldots, t_n\}$ for the data mining method is a relation over the relation schema $\{I_1, I_2, \ldots, I_m\}$, i.e., a set of binary vectors of size m. Each row can be considered as a set of properties or items (that is, $t[i] = 1 \Leftrightarrow I_i \in t$).

Let $W \subseteq R$ be a set of attributes and $t \in r$ a row of the relation. If $t[A] = 1$ for all $A \in W$, we write $t[W] = \bar{1}$. An *association rule* over r is an expression $W \Rightarrow B$, where $W \subseteq R$ and $B \in R \setminus W$. Given real numbers γ (confidence threshold) and σ (support threshold), we say that r satisfies $W \Rightarrow B$ with respect to γ and σ, if

$$|\{i \mid t_i[WB] = \bar{1}\}| \geq \sigma n$$

and

$$\frac{|\{i \mid t_i[WB] = \bar{1}\}|}{|\{i \mid t_i[W] = \bar{1}\}|} \geq \gamma.$$

That is, at least a fraction σ of the rows of r have 1's in all the attributes of WB, and at least a fraction γ of the rows having a 1 in all attributes of W also have a 1 in B. Given a set of attributes X, we say that X is *covering*[1] (with respect to the database and the given support threshold σ), if

$$|\{i \mid t_i[X] = \bar{1}\}| \geq \sigma n.$$

That is, at least a fraction σ of the rows in the relation have 1's in all the attributes of X.

[1] Agrawal et al. use the term *large* [AIS93].

As an example, suppose support threshold $\sigma = 0.3$ and confidence threshold $\gamma = 0.9$, and consider the example database

$$ABCD, ABEFG, ABHIJ, BK.$$

Now, three of four rows contain the set $\{AB\}$, so the support is $|\{i \mid t_i[AB] = \bar{1}\}| = 0.75$; supports of A, B, and C are 0.75, 1, and 0.25, correspondingly. Thus, $\{A\}$, $\{B\}$, and $\{AB\}$ have supports larger than the threshold $\sigma = 0.3$ and are covering, but $\{C\}$ is not. Further on, the database satisfies $\{A\} \Rightarrow B$, as $\{AB\}$ is covering, and the confidence $\frac{0.75}{0.75}$ is larger than $\gamma = 0.9$. The database does not satisfy $\{B\} \Rightarrow A$ because the confidence $\frac{0.75}{1}$ is less than the threshold γ.

The coverage is monotone with respect to contraction of the set: if X is covering and $B \subseteq X$, then $t_i[B] = \bar{1}$ for any $i \in \{i \mid t_i[X] = \bar{1}\}$, and therefore B is also covering. On the other hand, association rules do not have monotonicity properties with respect to expansion or contraction of the left-hand side: if $W \Rightarrow B$ holds, then $WA \Rightarrow B$ does not necessarily hold, and if $WA \Rightarrow B$ holds, then $W \Rightarrow B$ does not necessarily hold. In the first case the rule $WA \Rightarrow B$ does not necessarily have sufficient support, and in the second case the rule $W \Rightarrow B$ does not necessarily hold with sufficient confidence.

3 Finding association rules

Basic algorithm

The approach in [AIS93] to finding association rules is to first find all covering attribute sets X, and then separately test whether the rule $X \setminus \{B\} \Rightarrow B$ holds with sufficient confidence.[2] We follow this approach and concentrate on the algorithms that search for covering subsets.

To know if a subset $X \subseteq R$ is not covering, one has to read at least a fraction $1 - \sigma$ of the rows of the relation, that is, for small values of support threshold σ almost all of the relation has to be considered. During one pass through the database we can, of course, check for several subsets whether

[2]It is easy to see that this approach is in a sense optimal: the problem of finding all covering subsets of R can be reduced to the problem of finding all association rules that hold with a given confidence. Namely, if we are given a relation r, we can find the covering sets by adding an extra column B with all 1's to r and then finding the association rules that have B on the right-hand side and hold with certainty 1.

they are covering or not. If the database is large, it is important to make as few passes over the data as possible. The extreme method would be to do just one pass and check for each of the 2^m subsets of R whether they are covering or not. This is infeasible for all but the smallest values of m.

The method of [AIS93] makes multiple passes over the database. During a database pass, new *candidates* for covering sets are generated, and support information is collected to evaluate which of the candidates actually are covering. The candidates are derived from the database tuples by extending previously found covering sets in the *frontier*. For each database pass, the frontier consists of those covering sets that have not yet been extended. Each set in the frontier is extended up to the point where the extension is no longer *expected* to be covering. If such a candidate unexpectingly turns out to be covering, it is included in the frontier for the next database pass. The expected support required for this decision is derived from the frequency information of the items of the set. Originally the frontier only contains the empty set.

An essential property of the method of [AIS93] is that both candidate generation and evaluation is performed during the database pass. The method of [AIS93] further uses two techniques to prune the candidate space during the database pass. These are briefly described in Appendix B.

We take a slightly different approach. Our method tries to use all available information from previous passes to prune candidate sets between the database passes; the passes are kept as simple as possible. The method is as follows. We produce a set L_s as the collection of all covering sets of size s. The collection C_{s+1} will contain the candidates for L_{s+1}: those sets of size $s+1$ that can possibly be in L_{s+1}, given the covering sets of L_s.

1. $C_1 := \{\{A\} \mid A \in R\}$;
2. $s := 1$;
3. **while** $C_s \neq \emptyset$ **do**
4.   database pass: let L_s be the elements of C_s that are covering;
5.   candidate generation: compute C_{s+1} from L_s;
6. $s := s + 1$;
7. **od**;

The implementation of the database pass on line 4 is simple: one just uses a counter for each element of C_s.

In candidate generation we have to compute a collection C_{s+1} that is certain to include all possible elements of L_{s+1}, but which does not contain any unnecessary elements. The crucial observation is the following. Recall that L_s denotes the collection of all covering subsets X of R with $|X| = s$. If $Y \in L_{s+e}$ and $e \geq 0$, then Y includes $\binom{s+e}{s}$ sets from L_s. This claim follows immediately from the fact that all subsets of a covering set are covering.

Despite its triviality, this observation is powerful. For example, if we know that $L_2 = \{AB, BC, AC, AE, BE, AF, CG\}$, we can conclude that ABC and ABE are the only possible members of L_3, since they are the only sets of size 3 whose all subsets of size 2 are included in L_2. This further means that L_4 must be empty.

In particular, if $X \in L_{s+1}$, then X must contain $s+1$ sets from L_s. Thus a specification for the computation of C_{s+1} is to take all sets with this property:

$$C_{s+1} = \{X \subseteq R \mid |X| = s+1 \text{ and } X \text{ includes } s+1 \text{ members of } L_s\} \quad (1)$$

This is, in fact, the smallest possible candidate collection C_{s+1} in general. For any L_s, there are relations such that the collection of covering sets of size s is L_s, and the collection of coverings sets of size $s+1$ is C_{s+1}, as specified above.[3]

The computation of the collection C_{s+1} so that (1) holds is an interesting combinatorial problem. A trivial solution would be to inspect all subsets of size $s+1$, but this is obviously wasteful. One possibility is the following. First compute a collection C'_{s+1} by forming pairwise unions of such covering sets of size s that have all but one attribute in common:

$$C'_{s+1} = \{X \cup X' \mid X, X' \in L_s, |X \cap X'| = s-1\}.$$

Then $C_{s+1} \subseteq C'_{s+1}$, and C_{s+1} can be computed by checking for each set in C'_{s+1} whether the defining condition of C_{s+1} holds. The time complexity is

$$O(s|L_s|^2 + |C'_{s+1}|s|L_s|);$$

further, $|C'_{s+1}| = O(|L_s|^2)$, but this bound is very rough.

An alternative method is to form unions of sets from L_s and L_1:

$$C''_{s+1} = \{X \cup X' \mid X \in L_s, X' \in L_1, X' \not\subseteq X\},$$

and then compute C_{s+1} by checking the inclusion condition. (Note that the work done in generating C_{s+1} does not depend on the size of the database, but only on the size of the collection L_s.)

[3] Results on the possible relative sizes of L_s and C_{s+1} can be found in [Bol86].

Refinements

Instead of computing C_{s+1} from L_s, one can compute several families $C_{s+1}, \ldots, C_{s+e}$ for some $e > 1$ directly from L_s. That is, we first compute C_{s+1} as above. Then we compute C'_{s+2} and C_{s+2} assuming that all elements of C_{s+1} will in fact be covering, etc., until C_{s+e} has been computed. Then we check in a single database pass which subsets of $C_{s+1} \cup \cdots \cup C_{s+e}$ are covering. An alternative is to generate sets $C'_{s+1}, \ldots, C'_{s+e}$ simultaneously by taking unions of two sets from C_s and checking that the size of the union is at most $s + e$. Then C_{s+j} is computed from C'_{s+j} by checking for each $X \in C'_{s+j}$ whether X includes exactly $\binom{s+j}{s}$ sets of L_s. We call this latter method the e-extension method.

This system is especially advantageous if the number of sets is so small that the coverage of all the sets in these families can be checked in a single pass, so that the idea of pass minimization from [AIS93] is used. For example, if

$$L_2 = \{AB, BC, AC, AD, BD, CD, AE, FG, BG\},$$

we know immediately that $L_5 = \emptyset$, since

$$\binom{5}{2} = 10 > |L_2|.$$

The sets that can possibly be in L_3 are

$$ABC, BCD, ACD, ABD,$$

and the only possible member of L_4 is $ABCD$. In this example it would be practical to evaluate the coverage of all 5 potential sets in $L_3 \cup L_4$ in a single pass, instead of two passes.

A suitable number e of levels to check in one pass could be computed by first computing C_{s+1} as in the basic algorithm, then using this to compute C_{s+2}, etc., until the combined size of the sets C_i is close to the number of sets whose coverage can be checked in a single pass. (Again, note that these computations are relatively cheap, as they do not involve the database.) An essential improvement is that candidates are only generated once and between the database passes, unlike in the algorithm of [AIS93].

The computational complexity of the algorithms can be analyzed in terms of the quantities $|L_s|$, $|C_s|$, $|C'_s|$, and the size n of the database. The running time is linear in n and exponential in the size of the largest covering set.

For reasons of space we omit here the more detailed analysis. The database passes dominate the running time of the methods, and for very large values of n the algorithms can be quite slow. However, in the next section we shall see that by analyzing only small samples of the database we obtain a good approximation of the covering sets.

4 Analysis of sampling

We now consider the use of sampling in finding covering sets. We show that small samples are usually quite good for finding covering sets.

Let τ be the support of a given set X of attributes. Consider a random sample with replacement of size h from the relation. Then the number of rows in the sample that contain X is a random variable x distributed according to $B(h, \tau)$, i.e., binomial distribution of h trials, each having success probability τ.

The Chernoff bounds [AS92, HR90] state that for all a we have

$$Pr[x > h\tau + a] < e^{-2a^2/h}.$$

That is, the probability that the estimated support is off by at least α is

$$Pr[x > h(\tau + \alpha)] < e^{-2\alpha^2 h^2/h} = e^{-2\alpha^2 h},$$

i.e., bounded by a quantity exponential in h. For example, if $\alpha = 0.02$, then for $h = 3000$ the probability is $e^{-2.4} \approx 0.09$. (Similar results could be even more easily obtained by using the standard normal approximation.)

This means that sampling is a powerful way of finding association rules. Even for fairly low values of support threshold σ, a sample consisting of 3000 rows gives an extremely good approximation of the coverage of an attribute set. Therefore algorithms working in main memory are quite useful for the problem of finding association rules.

Appendix A contains an analysis of covering sets in random relations and a lower bound result for the problem of finding association rules.

5 Experiments

To evaluate the efficiency of our methods, we compare the original algorithm in [AIS93] to our algorithm. Candidate generation is performed by extending sets in L_s with other sets in L_s to achieve (at most) e-extensions. We compare a less aggressive extending strategy with $e = 1$ and a more aggressive strategy with $e = s$, where the size of the candidate sets is doubled during each iteration step. (We refer to our algorithm as *off-line candidate determination*; the variants are noted in the following as OCD_1 and OCD_s.) In addition to the original algorithm of [AIS93] (noted in the following by AIS_{orig}), we also implemented a minor modification of it that refrains from extending any set with an item that is not a covering set by itself (noted in the following by AIS_{mod}). Details about the implementations can be found in Appendix B.

Data

We have used data from two different sources to evaluate the algorithms. The first database is a course enrollment database, including registration information of 4734 students (one tuple per student). Each row contains the courses that a student has registered for, with a total of 127 possible courses. On the average, each row contains 4 courses. A simplified version of the database includes only the students with at least 2 courses (to be interesting for generating rules); this database consists of 2836 tuples, with an average of 6.5 items per tuple. The figures and tables in this paper represent this latter course enrollment data.

The second database is a telephone company fault management database, containing some 30,000 records of switching network notifications. The total number of attributes is 210. The database is basically a string of events, and we map it to relational form by considering it in terms of overlapping windows. This method is discussed in Section 6. The experiments on this data support the conclusions achieved with the course database.

The database sizes we have used are representative for sampling which would result in very good approximations, as was concluded in Section 4.

Results

Each algorithm finds of course the same covering sets and the same rules. The number of covering sets found with different support thresholds is presented in Table 1. Correspondingly, the number of association rules is presented in Table 2; we used a confidence threshold γ of 0.7 during all the experiments. The tables show that the number of covering sets (and rules) increases very fast with a decreasing support threshold.

Figure 1 presents the total time (in seconds) as a function of the inverse of the support threshold σ; we prefer to use the inverse of σ since it corresponds

Size	Support σ							
	0.40	0.20	0.18	0.16	0.14	0.12	0.10	0.08
1	2	11	13	14	14	14	16	18
2		10	17	26	35	53	68	79
3		4	5	12	22	52	102	192
4		1	1	1	5	19	69	171
5						1	19	76
6							1	29
7								3
$\sum$	2	26	36	53	76	139	275	568

Table 1: Number of covering sets.

	Support σ							
	0.40	0.20	0.18	0.16	0.14	0.12	0.10	0.08
Count	0	26	30	48	81	196	544	1426
Max size	0	4	4	4	4	5	6	7

Table 2: Number and maximal size of rules ($\gamma = 0.7$).

to the common sense idea of the "rarity" of the items. Figure 1 shows clearly that *OCD* is much more time efficient than *AIS*. The time requirement of *OCD* is typically 10–20 % of the time requirement of *AIS*, and the advantage of *OCD* increases as we lower σ. However, the difference between the algorithms is notable even with a large σ. The difference between the two variants of *OCD* is not large, and the modification we implemented in *AIS* did not affect its performance significantly. When we look at the total time as a function of $|L|$, the number of covering sets (presented in Figure 2), we observe that both algorithms behave more or less linearly with respect to $|L|$, but the time requirements of *OCD* increase much more slowly.

As an abstract measure of the amount of database processing we examine the *effective volume of the candidates*, denoted by V_{eff}. It is the total volume of candidates that are evaluated, weighted by the number of database tuples that must be examined to evaluate them. This measure is representative of the amount of processing needed during the database passes, independent of

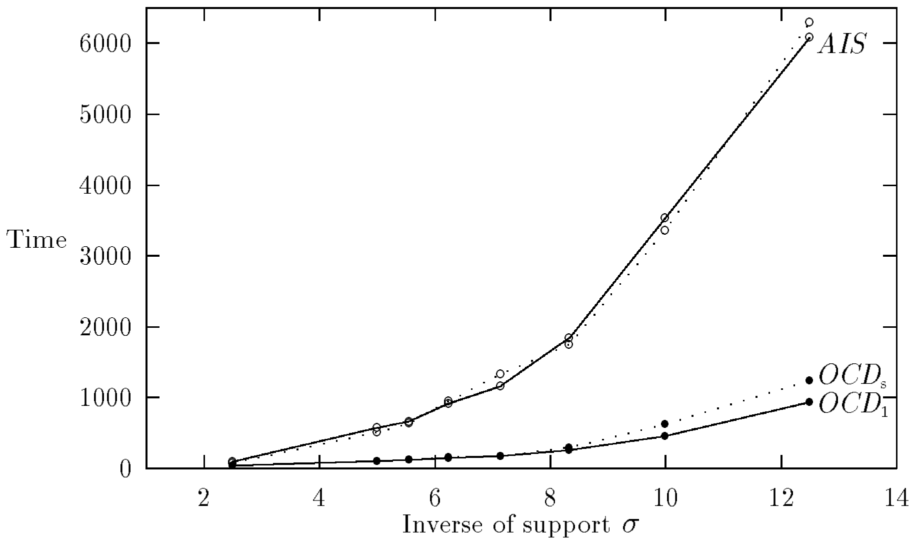

Figure 1: Total time (in seconds) as a function of the inverse of support.

implementation details.

The principal reason for the performance difference of *AIS* and *OCD* can be seen in Figures 3 and 4. They present the behavior of V_{eff} as the support threshold σ decreases and $|L|$, the number of covering sets, increases. The number of candidate sets (and their volume) considered by *OCD* is always smaller than that of *AIS* and the difference increases as more sets are found. These figures also explain why the more aggressively extending variant OCD_s does not perform better than the basic OCD_1: even though a more aggressive strategy often reduces the number of passes, it also results in so many more candidates that the reduction in time is offset. The number of passes for each algorithm is presented in Table 3: within this data OCD_s sometimes requires less passes than the others, but the number of passes obviously depends strongly on the data that is studied.

Table 4 presents the number of candidates considered by each method. The numbers for *AIS* are much higher, as it may generate the same candidates over and over again during the database pass. On the other hand, *OCD* only generates any candidate once (during the generation time) and checks that its subsets are covering before evaluating it against the database. While the number of potential candidates generated by *OCD* is much smaller than that

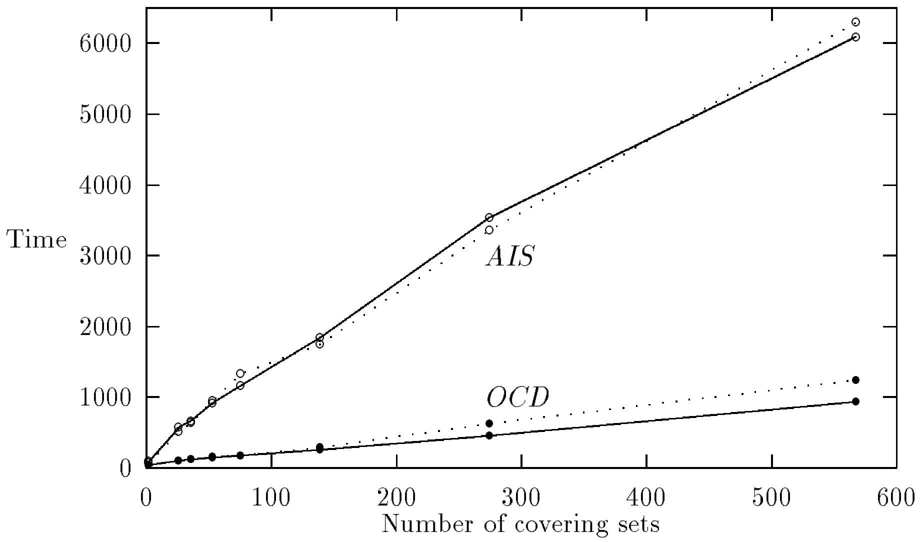

Figure 2: Total time (in seconds) as a function of the number of covering sets.

	Support σ							
	0.40	0.20	0.18	0.16	0.14	0.12	0.10	0.08
OCD_1	2	4	4	4	5	5	6	7
OCD_s	2	3	3	3	4	4	4	4
AIS_{orig}	2	4	4	4	4	5	6	7
AIS_{mod}	2	4	4	4	4	5	6	7

Table 3: Iteration passes.

for *AIS*, still fewer candidates need to be evaluated during the database pass.

If sampling is not used or the samples are large, the data does not remain in the main memory between passes, but it has to be reloaded for each pass. Thus it would be important to minimize the number of data passes. Also, if we want to overlap the database reads and the processing, the amount of processing performed during each database pass should be small. Figure 5 presents a typical profile of V_{eff} during the passes of one run (with $\sigma = 0.1$).

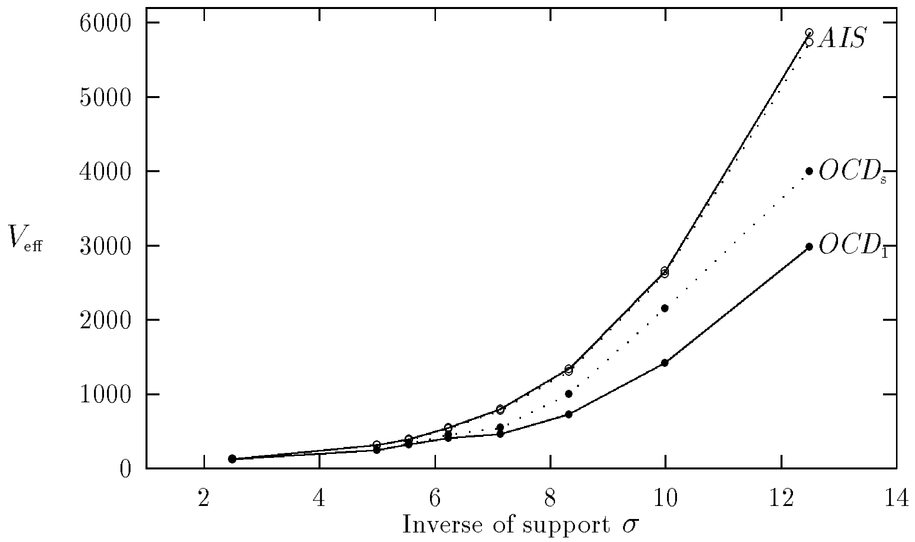

Figure 3: Effective volume of candidates as a function of the inverse of support.

	Support σ							
	0.40	0.20	0.18	0.16	0.14	0.12	0.10	0.08
OCD_l	128	196	242	289	362	552	950	1756
OCD_s	128	217	300	434	625	1084	1799	4137
AIS_{orig}	8517	37320	42175	48304	52415	65199	95537	118369
AIS_{mod}	9106	38068	42983	48708	53359	66704	96992	120749

Table 4: Generated candidates.

While the area beneath the curve corresponds to the total volumes, the height of the curve at each point describes how much processing is needed during that pass. High peaks correspond to passes where the overlapping of I/O and processing may be endangered if the database is large.

Since the confidence threshold γ only affects the number of rules generated from the covering sets we have not varied it in our experiments. On the other hand, suitable values for the support threshold σ depend very much on the

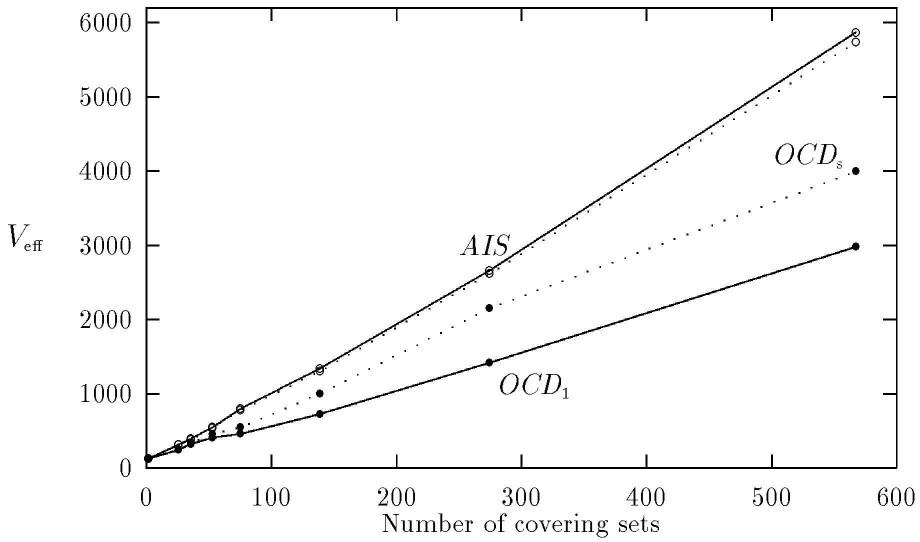

Figure 4: Effective volume of candidates as a function of the number of covering sets.

database.

6 Association rules in sequential data

We now consider discovering association rules in sequential data.

A sequence of events can be mapped to relational form by considering it in terms of overlapping windows. For instance, with a window size of 20 events, each window, or set, has 20 attributes equal to 1. Within our example application area, telecommunications fault management, a more useful approach is to use windows of a fixed real time size rather than a fixed number of events. With our database, a time window of about two hours, corresponding to approximately 60 events, was found suitable: with larger windows the number or size of rules hardly grew. This is also consistent with the background knowledge of the application area.

The rules found from sequential data reveal sets of events that tend to occur close to each other. The described windowing method is just one possible way to find association rules in sequential data; note however that

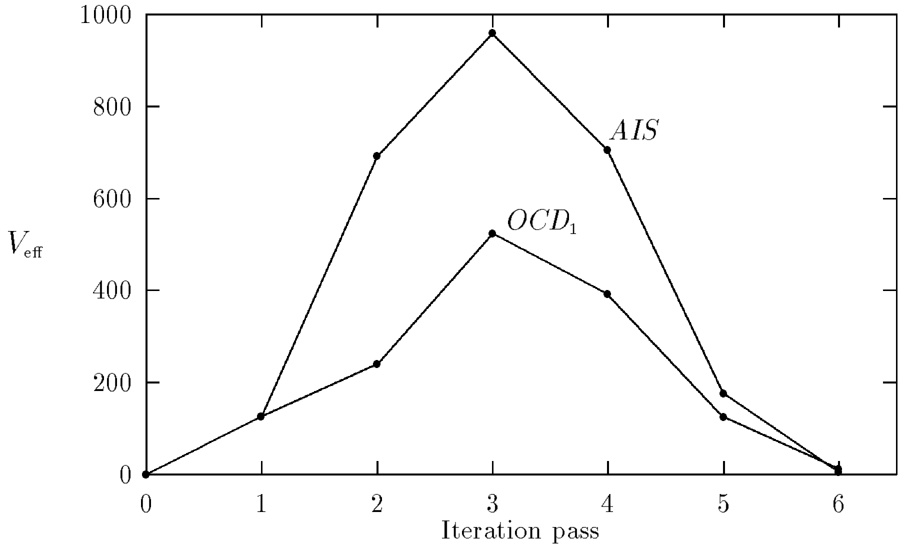

Figure 5: Effective volume of candidates during each iteration pass.

rules found with this method have two interesting and important properties. (1) There is no order implied on the events. This is valuable in a distributed environment, where time stamps may be inaccurate and the order of arrival to the central management application may be different from the order of the actual events. (2) The events in a rule need not be adjacent, as long as they fit in the same window. Thus, unrelated threads of events do not significantly disturb each other. — Rules can be utilized, e.g., in prediction or explanation of events.

The algorithms presented in Section 3 apply directly to the windowed sequence data. Windows can be generated by preprocessing the data into relational form. However, the database pass can be implemented much more efficiently in an incremental fashion, taking advantage of the fact that successive windows are largely overlapping. In the incremental algorithm, the window slides over the string of events, and only minimal updates need to be carried out. For each candidate, a counter tells how many of the attributes of the candidate are in the window. As soon as the whole candidate is in the window, the window index is saved; when the candidate is no longer in the window, the support counter is increased by the number of successive windows that contained the candidate. The candidates are indexed by their attributes, so that every time the window moves, exactly those candidates are accessed where something changes. In comparison to the unincremental method, the gain of this method grows linearly with the window size. In practical experiments, the efficiency has been 25 to 100 fold.

7 Related work

The basic inspiration for this paper is naturally [AIS93]. In this section we discuss briefly how finding association rules differs from traditional machine learning.

The discovery of rules from empirical data is, of course, a major topic in artificial intelligence and machine learning [Mug90, KM90]. The problem of finding association rules is, however, quite different from most machine learning tasks. First, we are not interested in obtaining a complete description of the data. Rather, we are content with some information that describes some (interesting) parts of the data. This can be compared to, e.g., the task of inferring decision trees or other classifiers [Qui86], where even in the presence of noise the task is still to obtain a reasonably small description of the whole data. Second, the examples we have are all positive; no negative information is given.

In the area of data mining [FPSM91, IJI92, PS93] the algorithms [PS91] have some similarities to the methods above. The PAC-model of Valiant [Val84, Ang92] is a robust setting for the study of theoretical properties of machine learning tasks. It is an interesting open question whether the task of finding association rules can be formulated within this framework.

We refer to [AIS93] for additional references.

8 Concluding remarks

Association rules are a simple and natural class of database regularities, useful e.g. in various analysis or prediction tasks. We have considered the problem of finding the association rules that hold in a given relation. Following the work of [AIS93], we have given an algorithm that uses all existing information between database passes to avoid checking the coverage of redundant sets. The algorithm gives clear empirical improvement when compared against the previous results, and it is simple to implement. We discussed discovery of association rules in sequential data, windowing with some interesting properties.

We have also analyzed the theoretical properties of the problem of finding association rules. We showed that sampling is an efficient technique for finding rules of this type, and that algorithms working in main memory can give extremely good approximations. In Appendix A we give some additional theoretical results. We also give a simple lower bound for a special case of the problem, and note that an algorithm from the different framework of [Lov87] actually matches this bound.

Several problems remain open. Some of the pruning ideas in [AIS93] are probably quite useful in certain situations; recognizing when to use such methods would help in practice. An algorithmic problem is how to find out as efficiently as possible what candidate sets occur in a given database row. Currently we simply check each candidate separately, i.e., if AB and AC are two candidates, the A entry of each row is checked twice. On certain stages of the search the candidates are heavily overlapping, and it could be useful to utilize this information.

A general problem in data mining is how to choose the interesting rules among the large collection of all rules. The use of support and confidence thresholds is one way of pruning uninteresting rules, but some other methods are still needed. In the course enrollment database many of the discovered rules correspond to normal process in the studies. This could be eliminated by considering a partial ordering among courses and by saying that a rule $W \Rightarrow B$ is not interesting if B is greater than all elements of W with respect to this ordering.

References

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 International Conference on Management of Data (SIGMOD 93)*, pages 207 – 216, May 1993.

[Ang92] D. Angluin. Computational learning theory: survey and selected bibliography. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 351–369. ACM Press, New York, NY, 1992.

[AS92] Noga Alon and Joel H. Spencer. *The Probabilistic Method.* John Wiley Inc., New York, 1992.

[Bol86] Béla Bollobás. *Combinatorics*. Cambridge University Press, Cambridge, 1986.

[ES90] Margaret Ellis and Bjarne Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley, 1990.

[FPSM91] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1 – 27. AAAI Press / The MIT Press, Menlo Park, CA, 1991.

[HR90] Torben Hagerup and Christine Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305–308, 1989/90.

[IJI92] *International Journal of Intelligent Systems*, 7(7):587–704, September 1992. Gregory Piatetsky-Shapiro, editor. Special Issue: Knowledge Discovery in Data- and Knowledge Bases.

[KM90] Y. Kodratoff and R. S. Michalski, editors. *Machine Learning: An Artificial Intelligence Approach*, volume III. Morgan Kaufmann, Los Altos, California, 1990.

[Lov87] Donald W. Loveland. Finding critical sets. *Journal of Algorithms*, 8:362 – 371, 1987.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms, Volumes 1–3*. Springer-Verlag, Berlin, 1984.

[Mug90] Stephen Muggleton. *Inductive Acquisition of Expert Knowledge*. Addison Wesley, Reading, MA, 1990.

[Näh92] Stefan Näher. LEDA user manual, version 3.0. Technical report, Max-Planck-Institut für Informatik, Im Stadtwald, D–6600 Saarbrücken, 1992.

[PS91] Gregory Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, 1991.

[PS93] Gregory Piatetsky-Shapiro, editor. *Knowledge Discovery in Databases,* AAAI-93 workshop working notes distributed to attendees, Washington, D.C., July 11–15 1993.

[PSF91] Gregory Piatetsky-Shapiro and William J. Frawley, editors. *Knowledge Discovery in Databases.* AAAI Press / The MIT Press, Menlo Park, CA, 1991.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

A Probabilistic analysis and a lower bound

In this appendix, we present some results describing the theoretical properties of the problem of finding association rules.

We first first show that in one model of random relations all covering sets are small. Consider a random relation $r = \{t_1, \ldots, t_n\}$ over attributes $R = \{I_1, I_2, \ldots, I_m\}$; assume that each entry $t_i[A_j]$ of the relation is 1 with probability q and 0 with probability $1-q$, and assume that the entries are independent. Then the probability of $t_i[X] = \bar{1}$ is q^h, where $h = |X|$. The number x of such rows is distributed according to $B(n, q^h)$, and we can again use the Chernoff bounds to obtain

$$Pr[x > \sigma n] = Pr[x > nq^h + n(s - q^h)] < e^{-2n(s-q^h)^2}.$$

This is furthermore bounded by $e^{-\sigma n}$, if $\sigma \geq 2q^h$. (For $\sigma = 0.01$ and $q = 0.5$, this means $h \geq 8$, and for $\sigma = 0.01$ and $q = 0.1$ it means $h \geq 3$.) Now the expected number of covering sets of size h is bounded by $m^h e^{-\sigma n}$. This is less than 0.5, provided $\sigma n > h \ln m + \ln 2$. Thus a random relation typically has only very few covering sets. Of course, relations occurring in practice are not random.

Next we describe some lower bound observations. Note first that a relation with one row consisting of all 1's satisfies all association rules. Thus the output of an algorithm that produces all association rules holding in a relation can be of exponential size in the number of attributes in the relation.

We now give an information-theoretic lower bound for finding one association rule in a restricted model of computation. We consider a model of computation where the only way of getting information from relation r is by asking questions of the form "is the set X covering". This model is realistic in the case the relation r is large and stored using a database system, and the model is also quite close to the one underlying the design of the algorithm in [AIS93].

Assume the relation r has n attributes. We claim that in the worst case one needs at least

$$\log \binom{n}{k} \approx k \log(n/k)$$

questions of the form "is the set X covering" to locate one maximal covering set, where k is the size of the covering set.

The proof of this claim is simple. Consider relations with exactly 1 maximal covering set of size k. There are $\binom{n}{k}$ different possible answers to the problem of finding the maximal covering set. Each question of the form "is the set X covering" provides at most 1 bit of information.

Loveland [Lov87] has considered the problem of finding "critical sets". Given a function $f : \mathcal{P}(R) \to \{0,1\}$ that is *downwards monotone* (i.e., if $f(X) = 1$ and $Y \subseteq X$, then $f(Y) = 1$), a set X is *critical* if $f(X) = 1$, but $f(Z) = 0$ for all supersets Z of X. Thus maximal covering sets are critical sets of the function $f(X) = 1$, if X is covering, and $f(X) = 0$, otherwise. The lower bound above matches exactly the upper bound provided by one of Loveland's algorithms. The lower bound above can easily be extended to the case where the task is to find k maximal covering sets. An interesting open problem is whether Loveland's algorithm can be extended so that it meets the generalized lower bound.

B Pruning methods and implementations

The method of [AIS93] uses two techniques to prune the candidate space during the database pass. In the "remaining tuples optimization", the occurrences of each frontier set are counted. A candidate is pruned by the optimization method, when there are less occurrences of the frontier set left than are needed for the candidate to be covering. Remember that the total number of occurrences of a frontier set has been evaluated in earlier passes. In the "pruning function optimization", items are assigned weights based on their rarity, and tuples are assigned weights from their items with synthesized functions. This method prunes a candidate, if—based on its weight—it is so rare that it can not be covering. To know the weight threshold for each set in the frontier, for each candidate set that is not expected to be covering—

and thus could be in the frontier in the next database pass—σn highest total weights of database rows containing the set are maintained. The lowest of these values is then stored, and the weights of candidates are compared against this weight threshold of the corresponding frontier set. The success of this method depends on the distributional properties of items.

The implementations of the algorithms have been kept reasonably straightforward. Attention was paid to the realization of the ideas rather than to optimizations of time or space. We wrote the algorithms in C++ [ES90], and used data structures from LEDA library [Näh92]. Implementations of both algorithms use the same basic data structures and algorithms for representing and manipulating sets of attributes. This ensures that timing comparisons are fair. Attributes are represented by their names as strings. Attribute sets are implemented as sorted sequences of strings, and collections of attribute sets, i.e. C_i and L_j, as sorted sequences of attribute sets. The sorted sequences are implemented as (2,4)-trees [Meh84].

The above mentioned pruning methods of [AIS93] require some additional data structures. "Remaining tuples optimization" only needs a counter for each frontier set, plus checking of the pruning condition when the support count for a candidate is being increased. For "pruning function optimization", the weights of items are stored in a randomized search tree during the first database pass. The weight thresholds are stored in another randomized search tree each database pass, to be utilized in the next iteration. The candidates are pruned by this method as soon as they are created. We have not implemented any memory saving techniques referred to in [AIS93] that might decrease the precision of the pruning.