Discovering Roll-Up Dependencies

Jef Wijsen
University of Antwerp
Jef.Wijsen@uia.ua.ac.be

Raymond T. Ng
University of British Columbia
rng@cs.ubc.ca

Toon Calders
University of Antwerp
Toon.Calders@uia.ua.ac.be

Abstract

We introduce the problem of discovering functional determinacies that result from "rolling up" data to a higher abstraction level. Such a determinacy is called a *Roll-Up Dependency* (RUD). An example RUD is: The probability that two files in the same directory have the same file extension, is greater than a specific number. We show the applicability of RUDs for OLAP and data mining. We consider the problem of mining RUDs that satisfy specified support and confidence thresholds. This problem is **NP**-hard in the number of attributes. We give an algorithm for this problem. Experimental results show that the algorithm uses linear time in the number of tuples of the input database.

1 Introduction

The problem of discovering functional dependencies (FDs) from relational databases has been studied [KMRS92]. Although FDs are the most important dependencies in database design, FDs are not prevalent in data mining. This may be attributed to the fact that FDs other than those known at design time, are fairly rare. We propose *Roll-Up Dependencies* (RUDs) with clear and interesting applications in data mining and OLAP. RUDs generalize FDs for attribute domains (called *levels*) that are organized in a finer-than order. Such finer-than relations are very common in data mining and OLAP; they have been called *concept hierarchies* [HCC93], or *roll-up/drill-down* lattices. For example, time can be measured in days (level DAY) or weeks (level WEEK). DAY is finer than WEEK, and there is a many-to-one mapping from DAY values to WEEK values capturing the natural containment of days in weeks. A price can be expressed in cents (level CENT) or integral Euros (level EURO).

KDD-99 San Diego CA USA

RUDs allow attributes to be compared for equality at different levels. For example, consider the relation scheme $(D : \texttt{DAY})(Price : \texttt{CENT})$ that stores the daily price in cents of a particular product. We have one price per day, as expressed by the FD:

$$D \rightarrow Price. \tag{1}$$

In general, cent prices can change from day to day, and we may be interested in finding regularities in price fluctuations. We may find that within a week, all cent prices are equal if they are expressed to the nearest integral Euro, as stated by the RUD:

$$(D : \texttt{WEEK}) \rightarrow (Price : \texttt{EURO}). \tag{2}$$

The meaning is that if the D-values of two tuples belong to the same week then their *Price*-values round to the same integral Euro. It is worth noting that the dependency (1) above does not imply the dependency (2), nor vice versa. The dependency (2) does imply, for example, $(D : \texttt{DAY}) \rightarrow (Price : \texttt{EURO})$.

In the next section, we present a few key motivations and applications of RUDs. Section 3 studies the problem of mining RUDs. We first define RUDs, and then we characterize the strength of a RUD by adapting the common notions of support and confidence. RUDMINE is the problem of finding the RUDs that satisfy specified support and confidence thresholds. We study the computational complexity of RUDMINE. Section 4 gives an algorithm for mining RUDs, and Section 5 reports on our experience with a real-life dataset. Section 6 discusses related work. Section 7 presents a conclusion.

2 Applications of RUDs

2.1 Scheme Decomposition

Consider the scheme:

$$(I : \texttt{ITEM})(D : \texttt{DAY})(S : \texttt{STORE})(Price : \texttt{CENT}). \tag{3}$$

A tuple $(I : u)(D : v)(S : w)(Price : x)$ expresses that the price of item u on day v in store w amounted

to x cents. Intuitively, one may think of this scheme as a *data cube* scheme, where I, D, and S are the dimensions, and *Price* is the measurement. A typical OLAP query then is *"Give for each item the highest sales price per area and year."*

Suppose the price of an item does not change within a month and a store chain, as captured by the RUD:

$$(I : \texttt{ITEM})(D : \texttt{MONTH})(S : \texttt{CHAIN}) \rightarrow (\mathit{Price} : \texttt{CENT}).$$

For example, the price of the item "Lego System 6115" in all stores of the Toysrus chain invariably was 912 cents during January, 1997. Then a relation over scheme (3) will store a lot of redundant prices: whenever two tuples that agree on I roll up to the same month and chain, they must necessarily agree on *Price*. The data redundancy can be avoided by decomposing scheme (3) into the schemes:

$$(I : \texttt{ITEM})(D : \texttt{MONTH})(S : \texttt{CHAIN})(\mathit{Price} : \texttt{CENT}), \quad (4)$$

$$(I : \texttt{ITEM})(D : \texttt{DAY})(S : \texttt{STORE}). \quad (5)$$

Scheme (4) is to store the price of an item for a particular month and chain. Scheme (5) is to store whether a particular item was sold in a particular store at a particular day, and can be dropped if such information is not needed.

2.2 Generalizing Data

We may be faced with a large number of pricing information, giving such a profusion of detailed information that direct comparison is impossible. For OLAP and data mining, it is often desirable to reduce the mass of data to a few manageable "representative" figures from which meaningful comparisons can be made. In this section we show that RUDs are intimately related to data generalization and summarization.

2.2.1 Rolling Up Data

On examination of a relation over scheme (4) we may observe that, although cent prices change within a year, the price fluctuations are small. In particular, we may find that within a year and a chain, an item is always sold at the same integral Euro price, as described by the RUD:

$$(I : \texttt{ITEM})(D : \texttt{YEAR})(S : \texttt{CHAIN}) \rightarrow (\mathit{Price} : \texttt{EURO}). \quad (6)$$

For example, the round Euro price of the item "Lego System 6115" in each store of the Toysrus chain invariably was 9 Euros during 1997. For applications in which Euro prices are sufficiently accurate, we can replace scheme (4) by the following one:

$$(I : \texttt{ITEM})(D : \texttt{YEAR})(S : \texttt{CHAIN})(\mathit{Price} : \texttt{EURO}). \quad (7)$$

The advantage is that relations over scheme (7) are, on average, twelve times smaller than relations over scheme (4) (as there are twelve months in a year).

2.2.2 Summarizing Data

In the preceding example, we rolled up days to years, and stores to chains, and rounded *Price*-values to the nearest integral Euro. Another way of summarizing *Price*-values is by taking the arithmetic mean, or any other measure of *central tendency* (median, mode). However, if important decisions in real life are based on the summarized *Price*-values, then we have to be careful that the *dispersion* of the prices summarized is not too high, or else the arithmetic mean is of limited significance. We argue that RUDs in combination with *confidence* can be used as a measure of dispersion. Assume that RUD (6) is highly but not fully satisfied. The confidence of this RUD is the conditional probability that the *Price*-values of two tuples round to the same integral Euro, given the two tuples concern the prices of the same item within a single year and a single chain. A high confidence then means that the dispersion of the prices of a particular item within a year and chain is small, and consequently, taking average prices is meaningful. In this way, RUDs give us an indication about the representativeness of central tendency measures.

The foregoing does not only hold for numeric data (like *Price*), but also for categorical data, as demonstrated by our real-life experiments (see Section 4). Consider a file system where files roll up to a certain type (FTYPE), such as "document," "program-source-code," or "program-object-code." The question is whether one can reasonably classify directories according to the types of the files they contain. If files in the same directory are of the same type with a high confidence, then such classification is reasonable; otherwise it is not.

3 The RUD Mining Problem

3.1 Definition of RUDs

Domain names are called *levels*. Every level l has associated with it a disjoint *domain* of *values*, denoted $dom(l)$. In all examples throughout this paper, levels appear in uppercase typewriter style. Examples used earlier include DAY, WEEK, CENT, and EURO. The family of all levels is a partially ordered set, as shown in Figure 1. The order is denoted $\preceq$, and corresponds to a finer-than relationship. Whenever $l \preceq l'$, there is a total function mapping each element $v \in dom(l)$ to an element $v' \in dom(l')$, and we say that v *rolls up* to v' in l'. Roll-up is transitive in the sense that whenever v rolls up to v' in l', and v' rolls up to v'' in l'' then v must roll up to v'' in l''. Roll-up is reflexive in the sense that every element $v \in dom(l)$ rolls up to itself in l. Details can be found in [WN98].

A *relation scheme* (or *scheme*) is a set $S = \{(A_1 : l_1), \ldots, (A_n : l_n)\}$ where $n \geq 0$, and $A_1, \ldots, A_n$ are distinct attributes, and $l_1, \ldots, l_n$ are levels. For

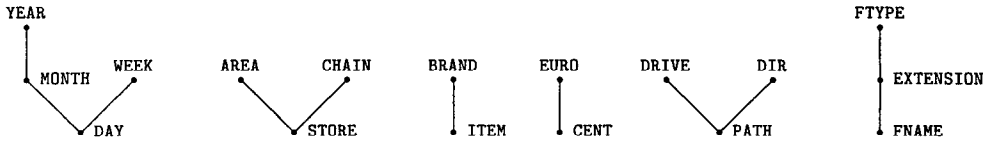

Figure 1: The Partially Ordered Set of Levels

simplicity, curly braces and commas will be omitted in the denotation of a set. Let S denote the scheme $(A_1 : l_1)(A_2 : l_2)\ldots(A_n : l_n)$ hereinafter. A *tuple* over S is a set $(A_1 : v_1)(A_2 : v_2)\ldots(A_n : v_n)$ where each $v_i \in dom(l_i)$. A *relation (instance)* over S is a set of tuples over S. Figure 3 *top* shows a relation over $(Path : \texttt{PATH})(Fname : \texttt{FNAME})$, with its intuitive meaning: a tuple $(Path : x)(Fname : y)$ means that x is the location of file y. Paths are a concatenation of drive (C or D) and directory.

A *generalization scheme* P of S is a set $(A_{i_1} : l_{i_1})\ldots(A_{i_m} : l_{i_m})$ satisfying the following conditions:

1. each A_{i_j} is an attribute among $A_1, \ldots, A_n$, and each l_{i_j} is a level such that if $A_{i_j} = A_k$ then $l_k \preceq l_{i_j}$ $(k \in [1..n],\ j \in [1..m])$; and
2. whenever $A_{i_j} = A_{i_k}$ with $j \neq k$ then $l_{i_j} \not\preceq l_{i_k}$ and $l_{i_k} \not\preceq l_{i_j}$ $(j, k \in [1..m])$.

If P is a generalization scheme of S, we also say that P is a generalization of S. Given the set of levels in Figure 1, there are more than three hundred generalizations of the scheme $(I : \texttt{ITEM})(D : \texttt{DAY})(S : \texttt{STORE})(Price : \texttt{CENT})$. Two such generalizations are: $(I : \texttt{ITEM})(D : \texttt{YEAR})(S : \texttt{AREA})$ and $(I : \texttt{BRAND})(S : \texttt{CHAIN})(S : \texttt{AREA})$. Because AREA and CHAIN are not comparable by $\preceq$, the last generalization can contain both $(S : \texttt{AREA})$ and $(S : \texttt{CHAIN})$.

The only assumption we make about $\preceq$ is that it is a partial order. We do not assume that the order $\preceq$ is a lattice, nor do we partition the set of all levels into different hierarchies, as is done in other proposals. The order $\preceq$ on levels gives rise to a relation $\trianglelefteq$ on generalization schemes as follows. For generalizations P and Q of scheme S, we write $P \trianglelefteq Q$ iff for every $(A : l) \in Q$, there exists some $(A : l') \in P$ such that $l' \preceq l$. While $\preceq$ is only a partial order, the set of all generalization schemes of S, ordered by $\trianglelefteq$, is a complete lattice (Theorem 1). We call this lattice the *roll-up lattice* over S. Figure 2 shows the roll-up lattice over the scheme $(D : \texttt{MONTH})(S : \texttt{STORE})$. Moreover, it is possible to assign a number to the nodes of the roll-up lattice such that (i) the top is numbered 0, and (ii) if P is a child of Q in the roll-up lattice, then the number of P equals the number of Q plus one. (P is called a child of Q, denoted $P \in children(Q)$, if P is an immediate descendant of Q in the roll-up lattice. In terms of lattice theory, $children(Q)$ are the nodes covered by Q.) The unique number that is in this way assigned to each node, is called the *stratum* of the node. This stratification of the roll-up lattice will be exploited by our RUD mining algorithm.

Theorem 1 *The set of all generalization schemes of S, ordered by $\trianglelefteq$, is a complete lattice. Furthermore, there exists a function f from the set of generalization schemes of S to $\mathbb{N}$ such that*

1. $f(\top) = 0$, *and*
2. *if* $P \in children(Q)$ *then* $f(P) = f(Q) + 1$.

The proof of Theorem 1 can be found in [WN98, Cal99]. Our notion of roll-up lattice extends and generalizes several earlier proposals found in the literature. Our notion is more general than the one in [HRU96], because the same attribute can appear more than once in a lattice element. This extension is natural and useful in OLAP applications. Dimensionality reduction [GCB+97] is embedded naturally in our roll-up lattice.

Finally, a *roll-up dependency* (RUD) over S is a statement of the form $P \rightarrow Q$ where P and Q are generalizations of S. A RUD gives rise to a convenient *histogram* representation of a relation instance, as is shown in Figure 3. Figure 3 *bottom* gives the histogram of the relation *MYFILES* and the RUD $(Path : \texttt{DRIVE}) \rightarrow (Fname : \texttt{EXTENSION})$. The first row in the histogram indicates that there are 2 tuples t in *MYFILES* such that $t(Path)$ rolls up to drive C and $t(Fname)$ rolls up to extension doc.

3.2 Notion of Satisfaction: Support and Confidence

Let S be a scheme. Let t and t' be tuples over S and let P be a generalization of S. We say that the tuples t and t' are *equivalent under* P (or *P-equivalent*) iff for every member $(A : l)$ of P, $t(A)$ and $t'(A)$ roll up to the same value in l. To continue with the example relation *MYFILES* given in Figure 3 *top*, the first two tuples are equivalent under $(Path : \texttt{DRIVE})(Fname : \texttt{EXTENSION})$, because their *Path*-values both roll up to drive C, and their *Fname*-values both roll up to extension doc. We say that a relation R *satisfies* a RUD $P \rightarrow Q$ iff for all tuples t, t' of R, whenever t and t' are P-equivalent

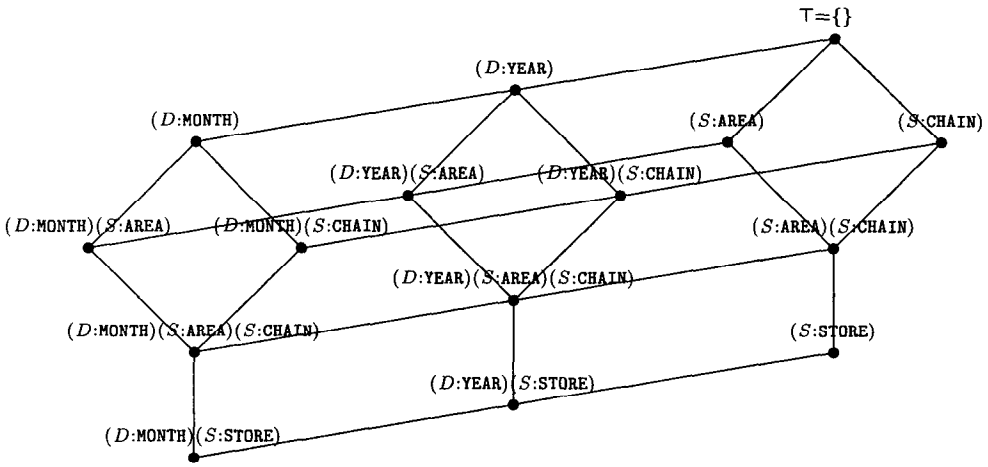

Figure 2: The Family of Generalizations of $(D : \texttt{MONTH})(S : \texttt{STORE})$ Ordered by $\trianglelefteq$

MYFILES

Path : PATH	*Fname* : FNAME
C:/Research/	Paper.doc
C:/Teaching/	Course.doc
C:/Research/	Paper.ps
C:/Teaching/	Course.ps
D:/Private/	Resume.doc
D:/Private/	Letter.doc
D:/Teaching/	Program.exe
D:/Teaching/	Handouts.ps
D:/Teaching/	Slides.ps
D:/Teaching/	Spreadsheet.ps

Path : DRIVE	*Fname* : EXTENSION	− #
C	doc	− 2
	ps	− 2
D	doc	− 2
	exe	− 1
	ps	− 3

Figure 3: A Relation Instance *MYFILES* over Scheme $(\mathit{Path} : \texttt{PATH})(\mathit{Fname} : \texttt{FNAME})$ and the Histogram of $(\mathit{Path} : \texttt{DRIVE}) \rightarrow (\mathit{Fname} : \texttt{EXTENSION})$

then they are Q-equivalent as well. It follows that the relation *MYFILES* falsifies the RUD $(\mathit{Path} : \texttt{DRIVE}) \rightarrow (\mathit{Fname} : \texttt{EXTENSION})$, as the first and the third tuple are equivalent under $(\mathit{Path} : \texttt{DRIVE})$ but not under $(\mathit{Fname} : \texttt{EXTENSION})$. That is, it is not true that all files on the same drive have the same file extension.

The notion of satisfaction introduced in the above paragraph is classical in the sense that a RUD is either satisfied or falsified by a relation; there is no third possibility. In data mining, one is typically not only interested in rules that are fully satisfied, but also in rules that are highly satisfied. We now introduce a relaxed notion of satisfaction by adapting the classical notions of support and confidence [AIS93].

Let R be a relation over the scheme S. Let $P \rightarrow Q$ be a RUD over S. The *confidence* of $P \rightarrow Q$ is the conditional probability that two tuples that are randomly selected from R without replacement, are Q-equivalent given they are already P-equivalent. The *support* of $P \rightarrow Q$ is the probability that two tuples that are randomly selected from R without replacement, are P-equivalent. Given a relation R, the support and the confidence of $P \rightarrow Q$ will be denoted $\mathit{sup}(P \rightarrow Q, R)$ and $\mathit{conf}(P \rightarrow Q, R)$ respectively. We give some differences between our notions of support and confidence and the notions with the same name used in association rule mining [AIS93]:

- Our support and confidence refer to pairs of tuples, rather than individual tuples. This is because only pairs of tuples can give evidence for or against a RUD. Contrast this with association rules where individual customer transactions can give evidence for or against a rule.
- In the definition of support, we consider *only left-*

hand sides of RUDs. In this way, support and confidence are independent—the support can be smaller or greater than the confidence. This is a divergence from the association rule literature, where the support is defined as the fraction of transactions satisfying both the left-hand *and the right-hand* side of a rule, and the support cannot exceed the confidence. In our framework, multiplying support and confidence gives the fraction of tuple pairs that are equivalent under both the left-hand and the right-hand side of the RUD under consideration. We prefer our notion of support because it has a natural characteristic: the confidence is a conditional probability; the probability that the conditioning event occurs, is the support.

- The confidence of a RUD lies between 0 and 1. If a RUD is fully satisfied, then its confidence is 1. The closer the confidence of a RUD $P \rightarrow Q$ is to 0, the greater the spread of the Q-values within each P-equivalence class. In this way, confidence is an indicator of dispersion. There are other more sophisticated measures of dispersion, like entropy.

Given a relation instance and a RUD, support and confidence can be computed from the histogram of the RUD. Precise mathematical formulas are easy to obtain and are given in [WN98]. We only include an example here. Recall that the support of $(Path : \texttt{DRIVE}) \rightarrow (Fname : \texttt{EXTENSION})$ is the probability that two files reside on the same drive. For the relation *MYFILES* shown in Figure 3, the support of $(Path : \texttt{DRIVE}) \rightarrow (Fname : \texttt{EXTENSION})$ is given by $\frac{C_2^4+C_2^6}{C_2^{4+6}} = \frac{7}{15}$. ($C_k^n$ denotes the total number of combinations of choosing k objects from n objects.) The C_2^4 gives the total number of *pairs* of files on drive C. Similarly, C_2^6 corresponds to the pairs of files on drive D. Finally, C_2^{4+6} gives the total number of pairs of files. The confidence of $(Path : \texttt{DRIVE}) \rightarrow (Fname : \texttt{EXTENSION})$ is given by: $\frac{(C_2^2+C_2^2)+(C_2^2+C_2^1+C_2^3)}{C_2^4+C_2^6} = \frac{2}{7}$. The denominator, as discussed above, gives the total number of pairs of tuples that are equivalent under $(Path : \texttt{DRIVE})$. Among those, there are a total of $(C_2^2+C_2^2)$ pairs of files that roll up to drive C and that roll up to the same file extension (doc and ps). Similarly, the next $(C_2^2 + C_2^1 + C_2^3)$ in the numerator gives the total number of pairs of files that roll up to drive D and that roll up to the same file extension. Here, C_2^1 is considered to be zero.

Apart from the notion of confidence discussed so far, we have also considered an alternative confidence measure for RUDs in [WN98]. The confidence measure discussed so far gives equal weight to all tuple pairs. Thus, larger groups may dominate smaller groups in the overall confidence. For the example given in Figure 3, there may be ten times more D-drive tuples than C-drive tuples. Under the current confidence measure, the RUD may become more a reflection of the situation on drive D than of a global one. For certain applications, it is desirable that each group be assigned equal weight, independent of its size. Thus, we also developed a measure that is based on the average confidence per group. See [WN98] for more details.

3.3 The RUDMINE Problem

Suppose we are given a very large relation over scheme (3). This relation gives such a profusion of detailed information that direct comparison of prices is impossible. The information has first to be simplified to a higher generalization level. In a first attempt, we may decide that prices in integral Euros are sufficiently accurate, so we roll up *Price*-values from CENT to EURO. We are still left with the problem of finding meaningful generalization levels for the attributes I, D, and S. By meaningful, we mean that the dispersion of the Euro prices within each group of aggregated data must be reasonably small. That is, we are actually looking for a generalization P of $(I : \texttt{ITEM})(D : \texttt{DAY})(S : \texttt{STORE})$ such that the confidence of $P \rightarrow (Price : \texttt{EURO})$ is high. This problem is generalized and captured by RUDMINE.

RUDMINE is the problem of finding all RUDs with a fixed right-hand generalization scheme that satisfy certain support and confidence thresholds. More formally, a RUDMINE problem is a quintet (S, Q, ts, tc, R) where S is a scheme, Q is a generalization scheme of S, support threshold ts and confidence threshold tc are rational numbers between 0 and 1, and R is a relation over S. The answer to the RUDMINE problem (S, Q, ts, tc, R) is the set containing every generalization scheme P of S that satisfies the following conditions: (i) $sup(P \rightarrow Q, R) \geq ts$ and $conf(P \rightarrow Q, R) \geq tc$; and (ii) P has no attributes in common with Q. The proof of the following result can be found in [WN98].

Theorem 2 *RUDMINE is* **NP***-hard.*

An upper bound for the complexity of RUDMINE depends on the complexity of roll-up functions. For practical situations, RUDMINE is in **NP**, and hence is **NP**-complete.

In the following section, we use the following notational conventions. The support of a RUD $P \rightarrow Q$ is independent of Q. For convenience, we can write $sup(P, R)$ instead of $sup(P \rightarrow Q, R)$ for any Q. Thus, we can talk about the support of a generalization scheme. Finally, we will use percentages to denote probabilities.

4 A RUD Mining Algorithm

Consider the RUDMINE problem (S, Q, ts, tc, R). Let S_l be the greatest subset of S that has no attributes in common with Q. Our algorithm is outlined in Figure 4

0. INPUT: A RUDMINE problem (S, Q, ts, tc, R).
1. OUTPUT: The solution of the RUDMINE problem.
2. % C^k computes the candidate generalization schemes at stratum k.
3. % *TooLow* computes the negative border.
4. $C^0 := \{\top\}$; *TooLow* := {}; $k := 0$
5. **while** $C^k \neq \{\}$ **loop**
6. % EVALUATE
7. compute $sup(P \rightarrow Q, R)$ and $conf(P \rightarrow Q, R)$ for
8. each $P \in C^k$ using "histogram inversion," and
9. • output each $P \in C^k$ with $sup(P \rightarrow Q, R) \geq ts$ and $conf(P \rightarrow Q, R) \geq tc$
10. • $C^{k+1} := \bigcup\{children(P) \mid P \in C^k$ and $sup(P \rightarrow Q, R) \geq ts\}$
11. • add to *TooLow* each $P \in C^k$ with $sup(P \rightarrow Q, R) < ts$
12. % PRUNE
13. **for** each $P \in$ *TooLow* **loop**
14. **for** each $P' \in C^{k+1}$ **loop**
15. **if** $P' \trianglelefteq P$ **then** remove P' from C^{k+1} **end-if**
16. **end-loop**
17. **end-loop**
18. $k := k + 1$
19. **end-loop**

Figure 4: Stratum-Wise Algorithm for Solving RUDMINE

and follows the idea of levelwise search [MT97]: start from the top of the roll-up lattice over S_l (i.e., the most general node), and then generate and evaluate more and more nodes down the lattice, without ever evaluating those nodes that cannot be interesting given the information obtained in earlier iterations. More precisely, the roll-up lattice over S_l is traversed stratum-wise (cf. Theorem 1) from top to bottom to find the generalizations of S_l with sufficient support and confidence. The set C^k contains the candidate generalization schemes at stratum k. If an element P of C^k has sufficient support, then all children of P are inserted in C^{k+1}. An "apriori trick" is used for pruning: if the support of $P \rightarrow Q$ is below the threshold, and $P' \trianglelefteq P$, then we know *a priori* that $P' \rightarrow Q$ must fail the support threshold. This is because the support decreases monotonically if we traverse the roll-up lattice from top to bottom. The set *TooLow* contains the nodes of the roll-up lattice whose support was effectively computed but failed the minimum threshold support; this corresponds to what is called the *negative border* in association rule mining. Nodes below the negative border must necessarily fail the threshold support. Note incidentally that no such monotonicity property holds for confidence, and hence confidence is not used for pruning.

We developed and implemented a technique called "histogram inversion" for computing the support and confidence of candidate generalization schemes. The inverted histogram of $P \rightarrow Q$ is the histogram of $Q \rightarrow P$. Figure 5 shows the inverted histogram of the histogram in Figure 3. Both histograms contain the same figures but in a different order. For the RUDMINE problem (S, Q, ts, tc, R), the input database R is first partitioned according to the generalization scheme Q. This is shown in Figure 6 *left* for the relation *MYFILES* and the right-hand side (*Fname* : `EXTENSION`). Significantly, as Q is fixed for a given RUDMINE problem, this partitioning has to be done *only once.* From this partitioned input database, we can construct fragments of the inverted histogram of $P \rightarrow Q$ for any P. These fragments can then be merged to obtain all figures needed to compute the support and confidence of $P \rightarrow Q$. Figure 6 shows the computation of the support and confidence of (*Path* : `DRIVE`) $\rightarrow$ (*Fname* : `EXTENSION`). The figure shows one merge operation; in practice, merge can be done two histograms at a time. In this example, the fixed right-hand side is a singleton, but in general, our algorithm allows multiple right-hand side attributes.

Theoretically, histogram inversion is an interesting application of the well-known Bayes' theorem, which reads as follows in our framework:

$$conf(P \rightarrow Q, R) = \frac{sup(Q, R) \times conf(Q \rightarrow P, R)}{sup(P, R)} \quad (8)$$

for any relation R. Equation (8) expresses the relation between $P \rightarrow Q$ and the "inverted rule" $Q \rightarrow P$; note

Fname : EXTENSION	*Path* : DRIVE	#
doc	C	2
	D	2
ps	C	2
	D	3
exe	D	1

Figure 5: Inverted Histogram of (*Path* : DRIVE) → (*Fname* : EXTENSION)

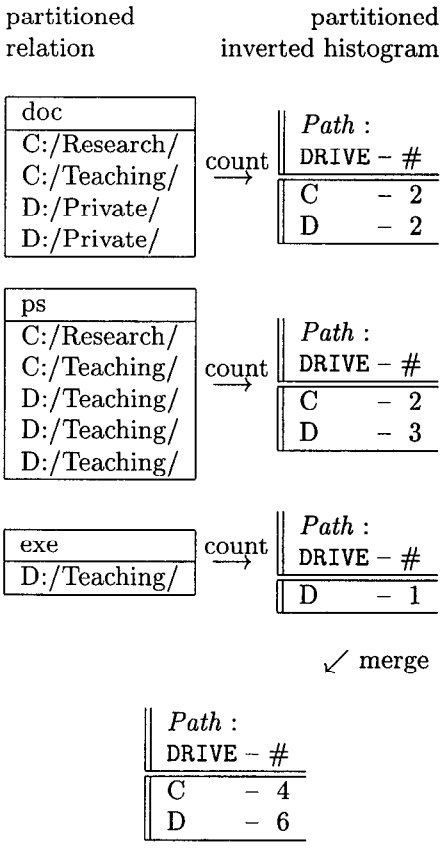

Figure 6: Technique Used for Computing Support and Confidence

that Q and hence $sup(Q, R)$ are constant in RUDMINE.

The advantages of the histogram inversion technique are as follows. The original non-inverted histograms can become very large as one goes down the roll-up lattice, and may not fit into main memory. In the histogram inversion approach, we only need to store smaller fragments, and we can compute fragments in main memory until it is filled. For generalization schemes near the top of the roll-up lattice, on the other hand, the size of the (inverted) histograms is small. In that case, we can compute several generalization schemes together in main memory, and the cost of reading the (partitioned) input relation is amortized over multiple generalization schemes.

Significantly, our algorithm computes complete and exact results, in the sense that if a certain RUD is an answer of the RUDMINE problem under consideration, then that RUD will be found by the algorithm.

5 Experimental Results

Our experiments were performed on a 200 MHz Pentium PC with 64 MB of main memory. The database was built by gathering properties of files on a number of hard disks. Properties of files included *Fname*, *Path*, *Time* (of creation), and *Size*. The roll-up lattice contained up to 200 nodes. These data are fairly easy to obtain, roll-ups are natural, and our familiarity with the dataset facilitated interpreting the output of the algorithm.

Scale-Up Figure 7 shows the execution time of the algorithm as we increase the number of tuples in the input database, for three different levels of threshold support (1%, 5%, and 10%). In the experiments, the fixed right-hand side was (*Fname* : EXTENSION). The graph shows that the algorithm scales quite linearly. Note that the execution time is independent of the threshold confidence, as the confidence is not used for pruning.

Figure 8 shows the execution time as we increase the number of attributes that can appear at the left-hand side. More precisely, consider the RUDMINE problem (S, Q, ts, tc, R). Let S_l be the greatest subset of S that has no attributes in common with Q, and let $S_r = S \backslash S_l$. S_l contains all possible left-hand side attributes. In our experiment S_l contained seven attributes. For *every* non-empty subset S'_l of S_l we measured the time needed to solve the RUDMINE problem $(S'_l \cup S_r, Q, ts, tc, R)$. The graph shows that the execution time can vary considerably depending on which set of attributes is chosen. For example, attributes with a small number of distinct values are easier to handle.

Pruning As stated by Theorem 2, RUDMINE is **NP**-hard. The source of exponentiality is that the size

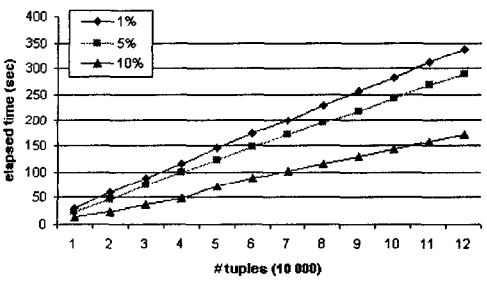

Figure 7: Execution Time in Function of the Number of Tuples

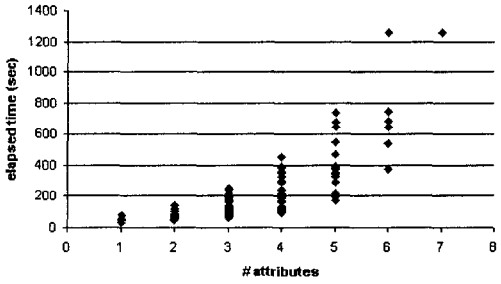

Figure 8: Execution Time in Function of the Number of Attributes (50 000 Tuples, 5% Threshold Support)

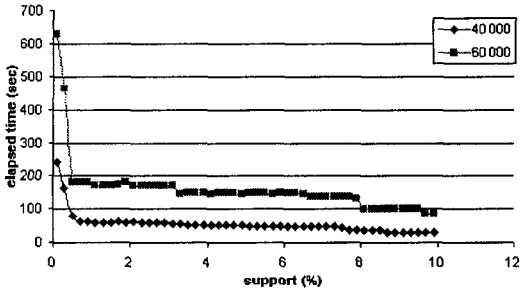

Figure 9: Execution Time in Function of the Threshold Support

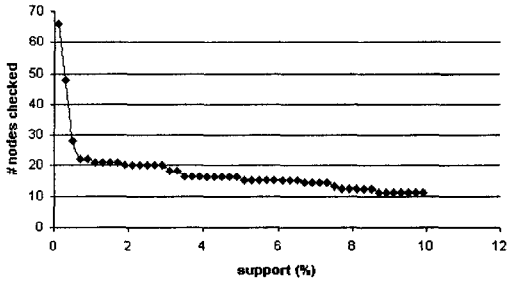

Figure 10: Number of Nodes of the Roll-Up Lattice that are Effectively Evaluated in Function of the Threshold Support

of the roll-up lattice is exponential in the number of attributes. In the worst case, our algorithm has to evaluate every element of the roll-up lattice. Therefore, it is important to see how many nodes of the roll-up lattice have to be evaluated in practice. Figure 9 shows the execution time as we increase the threshold support, for two different database sizes. Increasing the threshold support from 0.1% to 0.7% results in a drastic decrease of the execution time. The reason is explained by Figure 10, which shows the number of nodes of the roll-up lattice that are effectively evaluated as we increase the threshold support. These are the nodes that are above or on the negative border. The diagram shows that the pruning strategy used is quite effective for this dataset. Significantly, the nodes that are pruned away are near the bottom of the roll-up lattice, below the negative border. These nodes correspond to RUDs with large histograms that would otherwise be expensive to evaluate.

Interpretation of Output RUDs As expected, many RUDs had low support. Remind that our support only considers left-hand sides of RUDs. If the left-hand side is, for example, (*Path* : DIR), then the support of the RUD is the probability that two files belong to the same directory. If the number of directories is large, and files are evenly distributed over directories, then the support of the RUD is low.

More interesting than the support is the confidence of a RUD. Here caution is in order when interpreting high confidences. For example, RUDs with (*Path* : DRIVE) at the right-hand side happened to have a high confidence independent of the left-hand side, just because the confidence of $\{\} \rightarrow$ (*Path* : DRIVE) is high. The confidence of $\{\} \rightarrow$ (*Path* : DRIVE) is the probability that two files reside on the same drive; if n files are distributed over two drives (C and D), then one can prove that this probability will be at least $\frac{n-2}{2n-2} \approx 50\%$. The confidence of a RUD $P \rightarrow Q$ has therefore to be compared with the confidence of $\{\} \rightarrow Q$ to verify its statistical significance. An example of a statistically significant RUD was (*Path* : DIR) $\rightarrow$ (*Fname* : EXTENSION), saying that files in the same directory tend to have the same file extension. This RUD had a confidence of more than 50%, while the RUD $\{\} \rightarrow$ (*Fname* : EXTENSION) had a confidence of less than 2%.

6 Related Work

RUDs generalize FDs for relational databases that support roll-up/drill-down. The discovery of FDs from relational databases has already been studied before the explosive growth of data mining research [KMRS92]. Roll-up plays an important role in data mining and related areas. A lattice framework for OLAP has been

provided by Harinarayan et al. [HRU96]. As we pointed out in Section 3, our notion of *roll-up lattice* is more general than what has so far been proposed in the literature.

We believe that the applicability of RUD mining is far beyond that of FD mining. An OLAP user can rely on discovered RUDs to decide which roll-up operations are beneficial—rather than performing roll-up in an *ad hoc* manner. The integration of OLAP and data mining is a promising research area, which has been called *OLAP mining* [Han97].

This work is also strongly related to OLAP in the following way. Many computational issues involved in the algorithm to solve the RUDMINE problem are similar to those in computing data cubes. As a concrete example, Harinarayan et al. [HRU96] study the selection of views to materialize for a data cube. The views typically group data by one or more dimensions, and then apply a distributive set function on each group so obtained. Rather than distributive, our notion of confidence is an example of a holistic set function [GCB+97]. Little work has addressed data cubes that compute a holistic set function.

An interesting research topic is the use of RUDs in (multidimensional) database design. Our work was inspired by the work on temporal FDs (TFDs) and temporal normalization of Wang et al. [WBBJ97], and the TFDs proposed by ourselves [Wij99]. Loosely speaking, a TFD corresponds to a RUD where roll-up is only provided for one single dedicated timestamping attribute.

Theorem 2 shows that RUDMINE is **NP**-hard in the number of attributes. Data mining problems that have exponential complexity in terms of the number of attributes are not rare. Another example is the mining of quantitative association rules [SA96, WM98].

7 Summary

Roll-up dependencies (RUDs) generalize FDs for domains (called *levels*) that are related by a partial order that captures roll-up semantics. From this partially ordered set of levels we derive a complete *roll-up lattice*. Our construct of roll-up lattice is a generalization of several earlier proposals. RUDs have a high application potential in database design, data mining, and OLAP. We addressed the problem RUDMINE: discover RUDs whose support and confidence exceed certain specified threshold values. We can show that the problem is **NP**-hard. An upper bound for the complexity depends on the complexity of roll-up functions. We implemented an algorithm for mining RUDs, based on a technique called "histogram inversion." Experimental results show that the algorithm uses linear time in the number of tuples.

An interesting and important research goal is to further generalize our roll-up framework, and to study the impact of such generalizations on RUDs. For example, instead of saying that two prices in cents roll up to the same integral Euro, we may wish to express that the distance between two cent prices is less than one Euro.

References

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., 1993.

[Cal99] T. Calders. Het ontdekken van roll-up afhankelijkheden in databases. Master's thesis, University of Antwerp, 1999. In Dutch.

[GCB+97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–53, 1997.

[Han97] J. Han. OLAP mining: An integration of OLAP with data mining. In *Proceedings of the 7th IFIP 2.6 Working Conference on Database Semantics (DS-7)*, pages 1–9, 1997.

[HCC93] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(1):29–40, 1993.

[HRU96] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 205–216, Montreal, Canada, 1996.

[KMRS92] M. Kantola, H. Mannila, K.-J. Räihä, and H. Siirtola. Discovering functional and inclusion dependencies in relational databases. *Internat. Journal of Intelligent Systems*, 7:591–607, 1992.

[MT97] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[SA96] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 1–12, Montreal, Canada, 1996.

[WBBJ97] X.S. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical design for temporal databases with multiple granularities. *ACM Trans. on Database Systems*, 22(2):115–170, 1997.

[Wij99] J. Wijsen. Temporal FDs on complex objects. To appear in the March, 1999 issue of *ACM Trans. on Database Systems*, 1999.

[WM98] J. Wijsen and R. Meersman. On the complexity of mining quantitative association rules. *Data Mining and Knowledge Discovery*, 2(3):263–281, 1998.

[WN98] J. Wijsen and R.T. Ng. Discovering roll-up dependencies. Technical report, The University of British Columbia, Dept. of Computer Science, 1998. Also available at `http://www.uia.ua.ac.be/u/jwijsen/`.