Searching for Dependencies at Multiple Abstraction Levels

Toon Calders[*] Raymond T. Ng[†] Jef Wijsen[‡]

Abstract

The notion of roll-up dependency (RUD) extends functional dependencies with generalization hierarchies. RUDs can be applied in OLAP and database design. The problem of discovering RUDs in large databases is at the center of this paper. An algorithm is provided that relies on a number of theoretical results. The algorithm has been implemented; results on two real-life datasets are given. The extension of functional dependency (FD) with roll-ups turns out to capture meaningful rules that are outside the scope of classical FD mining. Performance figures show that RUDs can be discovered in linear time in the number of tuples of the input dataset.

1 Introduction

Generalization hierarchies are important in OLAP and data mining [CT97, GCB+97, Han97]. Examples of such hierarchies are numerous. Along the spatial dimension, for example, countries are divided into states, states into cities, cites into quarters, and so on. Along the temporal dimension, years are divided into months, months into days, days into hours, and so on. These hierarchies can be used to aggregate, for example, census data. Starting from the daily number of births by city, we can compute the annual number of births by country. The capacity of making such generalizations is crucial in today's decision support systems.

It is natural to ask which abstraction level is appropriate for a given dataset. For example, Belgium's birth-rate per 1000 inhabitants amounted to 11 in 1998. The number of births varies in function of social status and region, and such variations may be of interest to political decision makers, to whom annual birth-rates by social status and region are thus more meaningful than overall figures. A major problem addressed in this paper is that of finding those abstraction levels in databases that allow significant data aggregation without hiding important variations.

Suppose we have a dataset showing birth-rates by city for the year 1998. The observation that the birth-rate does not change within a region is expressed by:

$$(\mathit{City}, \mathtt{REGION}) \rightarrow (\mathit{Birth\_Rate}, \mathtt{PERMILLE}) \ ,$$

and called a *roll-up dependency* (RUD). The meaning is that whenever two cities belong to the same region, then their birth-rate expressed as an entire permille is the same. The following RUD expresses that birth-rates are homogeneous throughout the country:

$$(\mathit{City}, \mathtt{COUNTRY}) \rightarrow (\mathit{Birth\_Rate}, \mathtt{PERMILLE}) \ .$$

[*]calders@uia.ua.ac.be, University of Antwerp, Belgium.

[†]rng@cs.ubc.ca, University of British Columbia, Canada.

[‡]jef.wijsen@umh.ac.be, University of Mons-Hainaut, Belgium.

If there are significant differences in birth-rates among different regions of the country, the latter RUD will be falsified by the data, meaning that country-level birth-rates will hide existing regional differences, which can be undesirable. In this example, we only consider generalization along the spatial dimension. In general, abstraction may occur along several dimensions simultaneously.

The problem addressed in this paper is the following. Given a relational table (or a data cube) R with multiple attributes (or dimensions). Every attribute of R takes its values from a specified domain $dom(R)$. The domain values are ordered into generalization or *is_a* hierarchies. The problem is to find roll-up dependencies that are satisfied with high confidence.

This paper is organized as follows. The next section discusses some applications of the proposed construct. Roll-up dependencies and roll-up lattices are defined in Sections 3 and 4 respectively. The problem of mining RUDs is introduced in Section 5, and an algorithm for solving it in Section 6. Section 7 shows some experiments on real datasets. Related work is discussed in Section 8. Finally, Section 9 concludes the paper. Certain lengthy proofs are moved to the appendix.

2 Applications

We provide some applications of the proposed framework. The applications cover different areas such as OLAP, data mining, and database design.

2.1 Data Cube Reduction

Suppose we are given a dataset (or data cube) that contains detailed thermometer readings from numerous weather stations throughout the country. This results in a huge relation over the scheme $\{Station, Time, Temperature\}$. This information may be too detailed for effective data analysis. Of course, we can compute average daily temperatures by region, or average hourly temperatures for the entire country, thereby significantly decreasing the level of detail. It is natural to ask which level of detail is appropriate. If there are significant temperature differences between regions, it may be undesirable to hide these regional variations by summarizing temperatures at the country level. The goal becomes to find generalization levels that do not hide significant differences during data aggregation. Assume we discover that the data satisfies the RUD:

$$(Time, \mathtt{HOUR}), (Station, \mathtt{REGION}) \rightarrow (Temperature, \mathtt{INTEGER}) \ ,$$

stating that the temperatures, rounded to the nearest integer, are the same for all readings performed in the same region during the same one-hour-period. In that case we can significantly reduce the original data cube by rolling up *Time*-values to `HOUR`, *Station*-values to `REGION`, and by rounding *Temperature*-values.

2.2 Database Design

In temporal databases, generalization hierachies along the temporal dimension are denoted by the term *time granularity* and have been outlined in [BDE$^+$98]. The impact of time granularity on database design has been studied by a number of authors [WBBJ97, Wij99]. Suppose a relation over the scheme $\{Part, Warehouse, Price, Day\}$. A tuple $\{(Part, p), (Warehouse, w), (Cost, c), (Day, d)\}$ means that at day d, the part p was available at warehouse w at a cost of c. Suppose further that the cost of a part at a given warehouse cannot change within a month, which is expressed by the RUD:

$$\mathit{Part}, \mathit{Warehouse}, (\mathit{Day}, \texttt{MONTH}) \rightarrow \mathit{Cost} \enspace .$$

That is, whenever we have two tuples that agree on *Part* and *Warehouse*, and whose *Day*-values fall within the same month, then these tuples must necessarily agree on *Cost*. In that case, it is appropriate to decompose the scheme into two new schemes: one scheme $\{\mathit{Part}, \mathit{Warehouse}, \mathit{Day}\}$ to store the daily availability of parts from warehouses, and a second one $\{\mathit{Part}, \mathit{Warehouse}, \mathit{Cost}, \mathit{Month}\}$ to store the monthly cost price by warehouse for each part. Note that the time indication in the second scheme is in months, whereas the original scheme uses days. A difference with classical normalization is that the function mapping days to months is supposed to be a constant function outside the user-stored data. So RUDs are functional dependencies (FD) extended with pairs like, for example, $(\mathit{Day}, \texttt{MONTH})$ to indicate that *Day*-values should be converted into months before being compared for equality. The mapping of dates into values of the domain `MONTH` is part of the database scheme, not of the user-stored data.

The design principles used in "temporal normalization" also apply to non-temporal dimensions. Suppose, for example, that warehouses are grouped into commercial regions, and that the monthly cost is fixed for each region. This regularity would be expressed by the RUD:

$$\mathit{Part}, (\mathit{Warehouse}, \texttt{REGION}), (\mathit{Day}, \texttt{MONTH}) \rightarrow \mathit{Cost} \enspace .$$

Then it is advantageous to further generalize the scheme $\{\mathit{Part}, \mathit{Warehouse}, \mathit{Cost}, \mathit{Month}\}$ into $\{\mathit{Part}, \mathit{Region}, \mathit{Cost}, \mathit{Month}\}$, to store for each part the monthly cost per commercial region. Here again, we assume the existence of a fixed grouping of warehouses into regions. A discussion of RUDs in the spatio-temporal domain can be found in [WN99].

2.3 Prediction

Discovered RUDs can be used later on for prediction. Suppose we discovered from historical population records that the annual birth-rate does not change much between cities of the same region. If birth-rates for certain cities are lacking, we can reasonably guess that these rates should be similar to birth-rates of other cities in the same region. This way of predicting unknown values resembles k-nearest-neighbor; the difference is that the neighborhood is determined by similarity w.r.t. a generalization hierarchy, rather than by a distance metric and a number k. This idea has been elaborated in [BW01] for a slightly different framework.

3 Roll-Up Dependencies

3.1 Roll-ups

Definition 1 formalizes the notion of "roll-up." We assume a family of domain names, called *levels*, equipped with an order relation $\preceq$. This ordered set is called a *roll-up scheme* and captures the generalization hierarchies introduced in Section 1. A roll-up scheme is depicted in Figure 1. We have, for example, $\texttt{DAY} \preceq \texttt{MONTH}$ expressing that months are divided into days. The precise semantics is captured by the construct of *roll-up instance*, which assigns an extension to every level and "instantiates" the generalization hierarchy. For example, the day *6 Feb 2001* is an instance of `DAY` that "rolls up" to the month *February 2001*, which belongs

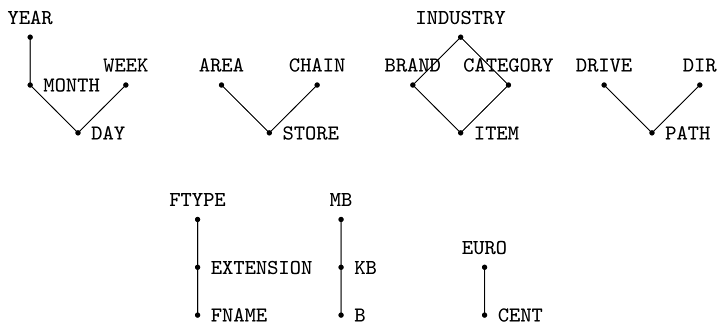


Figure 1: A roll-up scheme.

to MONTH. It is natural to require that roll-up be transitive: if *February 2001* rolls up further to the year *2001*, then *6 Feb 2001* should roll up to that same year.

Definition 1 We assume the existence of three countably infinite and pairwise disjoint sets: a set $\mathcal{A}$ of *attributes*, a set $\mathcal{L}$ of *levels*, and a set $\mathcal{D}$ of *constants*. A *roll-up scheme* is a pair $(L, \preceq)$ where L is a finite subset of $\mathcal{L}$ and $\preceq$ is a partial order on L. Let $l_1, l_2 \in L$. We write $l_1 \prec l_2$ iff $l_1 \preceq l_2$ and $l_1 \neq l_2$. The covering relation [DP90] of $\preceq$ is denoted $\prec\!\!\cdot$. That is, $l_1 \prec\!\!\cdot\, l_2$ iff $l_1 \prec l_2$ and $l_1 \preceq l_3 \prec l_2$ implies $l_3 = l_1$. If $l_1 \prec\!\!\cdot\, l_2$, l_1 is said to be *covered* by l_2.

A *roll-up instance* over the roll-up scheme $(L, \preceq)$ is a pair (ext, I) where:

- *ext* is a total function with domain L that maps every $l \in L$ to a disjoint set of constants, and
- I is a set of functions from $\mathcal{D}$ to $\mathcal{D}$ as follows. For every $l_1, l_2 \in L$ with $l_1 \preceq l_2$, I contains a total function from $ext(l_1)$ to $ext(l_2)$, denoted $I_{l_1}^{l_2}$, satisfying:
  1. for every $l \in L$, I_l^l is the identity function on $ext(l)$, and
  2. for every $l_1, l_2, l_3 \in L$ with $l_1 \preceq l_2 \preceq l_3$, for every $c \in ext(l_1)$, $I_{l_1}^{l_3}(c) = I_{l_2}^{l_3}(I_{l_1}^{l_2}(c))$.

  If $I_{l_1}^{l_2}(c) = d$, then we also say that c in l_1 *rolls up* to d in l_2.

□

3.2 Generalization Schemes (Genschemes)

A *relation scheme* is a set of attribute-level pairs, for example, $\{(\textit{Item}, \texttt{ITEM}), (\textit{Store}, \texttt{STORE}), (\textit{Day}, \texttt{DAY})\}$ to store the daily availability of items from stores. A *genscheme* (from "generalization" scheme) can be built from a relation scheme by replacing a level l by a level l' with $l \prec l'$, and/or by entirely omitting certain attributes. For example, $\{(\textit{Store}, \texttt{CHAIN}), (\textit{Day}, \texttt{MONTH})\}$ is a genscheme of the above relation scheme: the level STORE has been replaced by CHAIN where STORE $\prec$ CHAIN, and DAY by MONTH; the attribute *Item* has been omitted. The same attribute can appear twice in a genscheme provided that the associated levels are not comparable by $\preceq$. For example, $\{(\textit{Store}, \texttt{CHAIN}), (\textit{Store}, \texttt{AREA}), (\textit{Day}, \texttt{MONTH})\}$ is a valid genscheme as CHAIN and AREA are not related by $\preceq$, as shown in Figure 1.

A genscheme G of a relation scheme S naturally leads to a partitioning of each relation over S: two tuples belong to the same partition if they become equal after rolling up their attribute values to the levels specified by G. For example, the tuples $\{(\textit{Item}, \text{Lego44}), (\textit{Store}, \text{Toysrus}),$

$(\mathit{Day}, 9\text{ Feb }2001)\}$ and $\{(\mathit{Item}, \text{Lego77}), (\mathit{Store}, \text{Toysrus}), (\mathit{Day}, 17\text{ Feb }2001)\}$ belong to the same partition w.r.t. the genscheme $\{(\mathit{Store}, \texttt{CHAIN}), (\mathit{Day}, \texttt{MONTH})\}$ as *9 Feb 2001* and *17 Feb 2001* roll up to the same month.

Definition 2 Let a roll-up scheme $(L, \preceq)$ be given, as well as a roll-up instance (ext, I) over this roll-up scheme. A *relation scheme* S over $(L, \preceq)$ is a set $\{(A_1, l_1), \ldots, (A_n, l_n)\}$ where $n \geq 0$, $A_1, \ldots, A_n$ are pairwise distinct attributes, and $l_1, \ldots, l_n$ are (not necessarily distinct) levels of L. A *tuple* over the relation scheme $\{(A_1, l_1), \ldots, (A_n, l_n)\}$ is a set $\{(A_1, v_1), \ldots, (A_n, v_n)\}$ where $v_i \in \mathit{ext}(l_i)$ for each $i \in [1..n]$. A *relation* over the relation scheme S is a set of tuples over S.

A *genscheme* of the relation scheme $\{(A_1, l_1), \ldots, (A_n, l_n)\}$ is a set $\{(A_{i_1}, l_{i_1}), \ldots, (A_{i_m}, l_{i_m})\}$ satisfying:

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 $l_{i_k} \not\preceq l_{i_j}$ ($j, k \in [1..m]$).

Let G be a genscheme of the relation scheme S. Two tuples t_1, t_2 over S are said to be *G-equivalent*, denoted $t_1 \sim_G t_2$, iff for each pair (A, l_1) of S, for each pair (A, l_2) of G, $I_{l_1}^{l_2}(t_1(A)) = I_{l_1}^{l_2}(t_2(A))$. Let R be a relation over S. Obviously, $\sim_G$ is an equivalence relation on R; we write $R/\sim_G$ to denote the set of equivalence classes induced by $\sim_G$ on R. $\square$

For simplicity, if G is a genscheme of S such that both G and S contain the same pair (A, l), then we can replace in G the attribute-level pair (A, l) by the attribute A, where it is understood that the level l can be found in the relation scheme. For example, we can write simply *Day* instead of $(\mathit{Day}, \texttt{DAY})$ if the underlying relation scheme already specifies $(\mathit{Day}, \texttt{DAY})$. This simplification was implicitly assumed in the examples of Section 1.

3.3 Roll-up Dependencies (RUDs)

A *roll-up dependency* (RUD) is an implication between two genschemes of the same underlying relation scheme. Satisfaction is defined in the natural manner: $G \to H$ is satisfied if and only if whenever two tuples are G-equivalent then they are also H-equivalent.

Definition 3 Let a roll-up scheme $(L, \preceq)$ be given, as well as a roll-up instance (ext, I) over this roll-up scheme.

A *roll-up dependency* (RUD) over the relation scheme S is an expression $G \to H$ where G and H are genschemes of S. A roll-up dependency $G \to H$ over the relation scheme S is *satisfied* by a relation R over S iff for all tuples t_1, t_2 of R, if $t_1 \sim_G t_2$ then $t_1 \sim_H t_2$. $\square$

For a concrete example, consider the relation *FILES* of Figure 2. The first tuple of *FILES* means that the size of the file `Paper.doc` with path `C:\Research\` is 41 590 bytes. The roll-up scheme in use is that of Figure 1 with its intuitive semantics. The path `C:\Research\` rolls up to drive `C:` in `DRIVE`, and to directory `\Research\` in `DIR`. That is, a path is a drive followed by a directory. The first two tuples are equivalent under $\{(\mathit{Pa}, \texttt{DRIVE}), (\mathit{Fn}, \texttt{EXTENSION})\}$: their *Pa*-values both roll up to drive `C:`, and their *Fn*-values to extension `.doc`. The relation falsifies the RUD $(\mathit{Pa}, \texttt{DRIVE}) \to (\mathit{Fn}, \texttt{EXTENSION})$ because it is not true that all files on the same drive have the same file extension.

FILES	$(Pa, \texttt{PATH})$	$(Fn, \texttt{FNAME})$	$(Si, \texttt{B})$
	`C:\Research\`	`Paper.doc`	41 590
	`C:\Teaching\`	`Course.doc`	65 024
	`C:\Research\`	`Paper.ps`	1 327 081
	`C:\Teaching\`	`Course.ps`	134 302
	`D:\Private\`	`Resume.doc`	754 348
	`D:\Private\`	`Letter.doc`	251 738
	`D:\Teaching\`	`Program.exe`	1 040 960
	`D:\Teaching\`	`Handouts.ps`	370 079
	`D:\Teaching\`	`Slides.ps`	418 325
	`D:\Teaching\`	`Spreadsheet.ps`	93 428

Figure 2: The relation *FILES* over the relation scheme $\{(Pa, \texttt{PATH}), (Fn, \texttt{FNAME}), (Si, \texttt{B})\}$.

3.4 Ordering Genschemes

The order $\preceq$ on levels carries over naturally to an order $\trianglelefteq$ on genschemes. We have, for example, $\{(Store, \texttt{STORE}), (Day, \texttt{MONTH})\} \trianglelefteq \{(Store, \texttt{CHAIN}), (Day, \texttt{YEAR})\}$. Also $\{(Store, \texttt{CHAIN}), (Day, \texttt{YEAR})\} \trianglelefteq \{(Day, \texttt{YEAR})\}$, so omitting attributes also leads to a more general scheme.

Definition 4 Let S be a relation scheme over the roll-up scheme $(L, \preceq)$. We define a binary relation, denoted $\trianglelefteq$, on the set of all genschemes of S, as follows. For all genschemes G, H of S: $G \trianglelefteq H$ iff for all (A, l) in H, there is some (A, l') in G with $l' \preceq l$. □

Obviously, if $G \trianglelefteq H$ are two genschemes of the relation scheme S, then any relation over S will satisfy $G \to H$. The relation $\trianglelefteq$ will be explored in detail in the next section.

4 Roll-Up Lattice

In this section, we prove some additional properties of the relation $\trianglelefteq$ on the family of genschemes of a relation scheme S. First, we show that the relation $\trianglelefteq$ is a complete lattice, and second, we show that the elements of this lattice are layered in so-called strata. These properties are essential in the development of an algorithm for mining RUDs later on.

In the remainder we will implicitly assume a fixed roll-up instance for each roll-up scheme. Also, most definitions are relative to some roll-up scheme that is implicitly understood.

4.1 Complete Lattice

Given a roll-up scheme, the set of all genschemes of a relation scheme S, ordered by $\trianglelefteq$, is a complete lattice. The lattice will be denoted $S^{\diamond}$. Figure 3 shows the Hasse diagram of $S^{\diamond}$ for $S = (Pa, \texttt{PATH})(Si, \texttt{B})$; the underlying roll-up scheme is the one of Figure 1.

Not every set of attribute-level pairs is a valid genscheme; for example, $\{(Store, \texttt{STORE}), (Day, \texttt{MONTH}), (Day, \texttt{YEAR})\}$ is no genscheme because `MONTH` and `YEAR` are comparable by $\preceq$, i.e., $\texttt{MONTH} \preceq \texttt{YEAR}$. The operator $\lfloor \cdot \rfloor$ takes as its argument a set of attribute-level pairs and turns it into a genscheme by removing a pair (A, l') if the set already contains a pair (A, l) with $l \prec l'$. In the preceding example, applying $\lfloor \cdot \rfloor$ has the effect of removing $(Day, \texttt{YEAR})$.

Definition 5 Let G be a set of attribute-level pairs; i.e., $G = \{(A_1, l_1), \ldots, (A_n, l_n)\}$ with $n \geq 0$, $A_1, \ldots, A_n \in \mathcal{A}$, and $l_1, \ldots, l_n \in \mathcal{L}$. $\lfloor G \rfloor$ denotes the smallest subset of G containing (A_i, l_i) iff for every (A_j, l_j) of G, whenever $A_j = A_i$ and $l_j \preceq l_i$ then $l_i = l_j$.

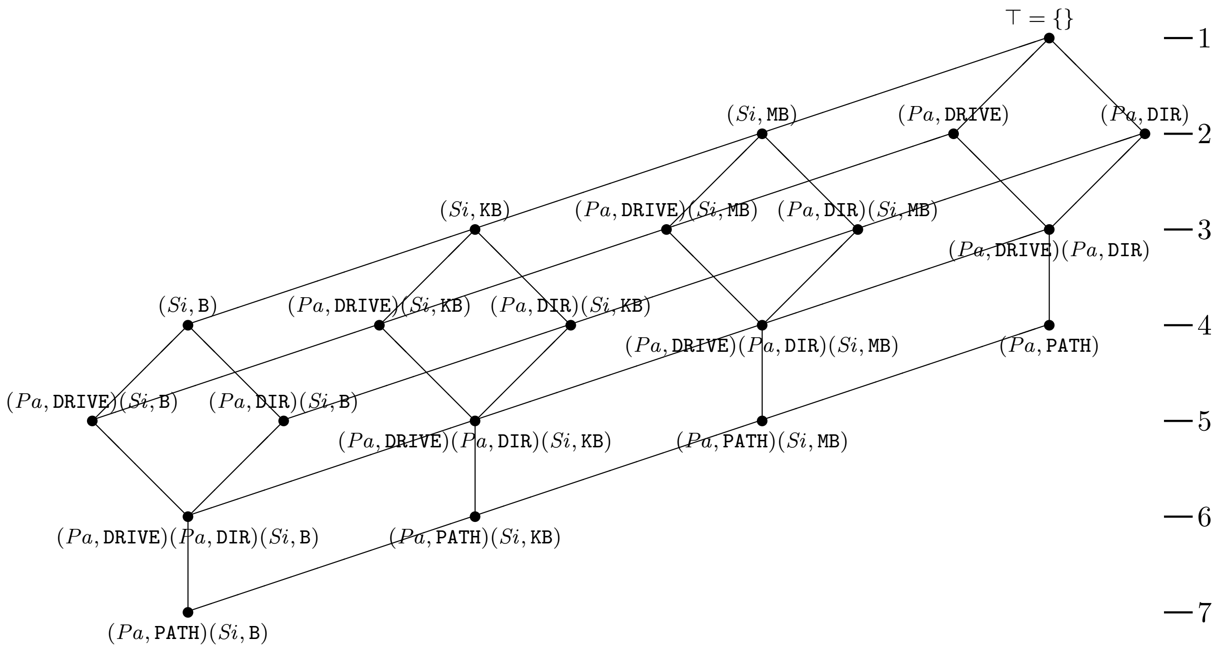

$\top = \{\}$
$(Si, \mathtt{MB})$
$(Pa, \mathtt{DRIVE})$
$(Pa, \mathtt{DIR})$
$(Si, \mathtt{KB})$
$(Pa, \mathtt{DRIVE})(Si, \mathtt{MB})$
$(Pa, \mathtt{DIR})(Si, \mathtt{MB})$
$(Pa, \mathtt{DRIVE})(Pa, \mathtt{DIR})$
$(Si, \mathtt{B})$
$(Pa, \mathtt{DRIVE})(Si, \mathtt{KB})$
$(Pa, \mathtt{DIR})(Si, \mathtt{KB})$
$(Pa, \mathtt{DRIVE})(Pa, \mathtt{DIR})(Si, \mathtt{MB})$
$(Pa, \mathtt{PATH})$
$(Pa, \mathtt{DRIVE})(Si, \mathtt{B})$
$(Pa, \mathtt{DIR})(Si, \mathtt{B})$
$(Pa, \mathtt{DRIVE})(Pa, \mathtt{DIR})(Si, \mathtt{KB})$
$(Pa, \mathtt{PATH})(Si, \mathtt{MB})$
$(Pa, \mathtt{DRIVE})(Pa, \mathtt{DIR})(Si, \mathtt{B})$
$(Pa, \mathtt{PATH})(Si, \mathtt{KB})$
$(Pa, \mathtt{PATH})(Si, \mathtt{B})$
—1
—2
—3
—4
—5
—6
—7

Figure 3: The lattice $(Pa, \mathtt{PATH})(Si, \mathtt{B})^{\diamond}$.

We can write $(A_1, l_1)(A_2, l_2) \ldots (A_n, l_n)$ instead of $\{(A_1, l_1), (A_2, l_2), \ldots, (A_n, l_n)\}$, omitting braces and commas. $\square$

Theorem 1 *Let S be a relation scheme over the roll-up scheme $(L, \preceq)$. The set of all genschemes of S, ordered by $\trianglelefteq$, is a complete lattice.*

PROOF. See Appendix A. $\square$

Definition 6 Let S be a relation scheme over the roll-up scheme $(L, \preceq)$. The set of all genschemes of S, ordered by $\trianglelefteq$, is denoted $S^{\diamond}$ (where the roll-up scheme is implicitly understood). $S^{\diamond}$ is called the *roll-up lattice* of S. The top and bottom elements of S are denoted $\top$ and $\bot$ respectively. The infimum of two genschemes G and H is denoted $G \wedge H$.

Let G, H be genschemes of S. We write $G \triangleleft H$ iff $G \trianglelefteq H$ and $G \neq H$. The covering relation of $\trianglelefteq$ is denoted $-\!\!\triangleleft$. That is, $G -\!\!\triangleleft H$ iff $G \triangleleft H$ and $G \trianglelefteq F \triangleleft H$ implies $F = G$. We define:

- $gen(G) := \{H \in S^{\diamond} \mid G -\!\!\triangleleft H\}$, the *direct generalizations* of G, and
- $spec(G) := \{H \in S^{\diamond} \mid H -\!\!\triangleleft G\}$, the *direct specializations* of G.

$\square$

Consider the roll-up lattice of Figure 3. The direct specializations of the genscheme $(Si, \mathtt{KB})$ are $(Si, \mathtt{B})$, $(Pa, \mathtt{DRIVE})(Si, \mathtt{KB})$, and $(Pa, \mathtt{DIR})(Si, \mathtt{KB})$. The direct generalizations of $(Pa, \mathtt{DRIVE})(Pa, \mathtt{DIR})$ are $(Pa, \mathtt{DRIVE})$ and $(Pa, \mathtt{DIR})$.

4.2 Constructing Direct Generalizations

In the algorithm later on, we need to perform the following task: Given a relation scheme S, a roll-up scheme $(L, \preceq)$, and a genscheme G of S, find all direct specializations and generalizations of G. The following theorem gives a characterization of the relation $\multimap\!\lhd$ in terms of $\multimap\!\prec$, which can be used to compute the direct specializations and generalizations of a given genscheme.

Theorem 2 *Let S be a relation scheme over the roll-up scheme $(L, \preceq)$. Let $G, H \in S^{\diamond}$. $G \multimap\!\lhd H$ iff for some pair $(A, l) \in G$, $H = \lfloor (G - \{(A, l)\}) \cup \{(A, l') \mid l \multimap\!\prec l'\} \rfloor$.*

Proof. See Appendix B. □

That is, every direct generalization of a genscheme G can be obtained by first replacing some element (A, l) of G by the set $\{(A, l') \mid l \multimap\!\prec l'\}$, and then applying the $\lfloor \cdot \rfloor$-operator. For example, let us construct the direct generalizations of the genscheme $(Pa, \texttt{PATH})(Size, \texttt{MB})$. Since no level covers MB, we directly obtain $(Pa, \texttt{PATH})$ as one direct generalization. For the other direct generalization, we replace $(Pa, \texttt{PATH})$ by $(Pa, \texttt{DRIVE})(Pa, \texttt{DIR})$. The application of $\lfloor \cdot \rfloor$ does not induce any changes, so the second direct generalization is $(Pa, \texttt{DRIVE})(Pa, \texttt{DIR})(Size, \texttt{MB})$.

For a second example, consider the genscheme $(Item, \texttt{BRAND})(Item, \texttt{CATEGORY})$. Since BRAND is only covered by INDUSTRY, one direct generalization is $\lfloor (Item, \texttt{CATEGORY})(Item, \texttt{INDUSTRY}) \rfloor$, which yields $(Item, \texttt{CATEGORY})$. Hence, $(Item, \texttt{BRAND})(Item, \texttt{CATEGORY}) \multimap\!\lhd (Item, \texttt{CATEGORY})$.

4.3 Stratification

Each roll-up lattice can be layered in strata. More precisely, a unique number can be assigned to every genscheme of a roll-up lattice, as follows: (a) the top is numbered 1, and (b) if $G \multimap\!\lhd H$ and H is numbered n, then G is numbered $n + 1$. In Figure 3, the number assigned to the genschemes is indicated at the right. For example, the number of $(Pa, \texttt{DRIVE})(Si, \texttt{KB})$ is 4. Such numbering divides the roll-up lattice into different strata. In the algorithms later on, this stratification will be used to traverse the roll-up lattice startumwise.

Definition 7 Let S be a relation scheme. A roll-up lattice $S^{\diamond}$ has a *stratification* iff for some $n \in \mathbb{N}$, there exists a numbered partition $P_1, \ldots, P_n$ of $S^{\diamond}$, such that (a) $P_1 = \{\top\}$, and (b) for each $G, H \in S^{\diamond}$, if $G \multimap\!\lhd H$ and $H \in P_i$, then $G \in P_{i+1}$. □

Theorem 3 *Let S be a relation scheme over the roll-up scheme $(L, \preceq)$. The roll-up lattice $S^{\diamond}$ has a unique stratification.*

Proof. See Appendix C. □

Note incidentally that not every complete lattice has a stratification.

5 RUDMINE

In this section, we consider the problem of finding all RUDs that hold in a database. We first define *support* and *confidence* of RUDs, and then we introduce the problem RUDMINE.

5.1 Notions of Satisfaction: Support and Confidence

In data mining, one is typically not only interested in rules that fully hold, but also in rules that almost hold. Therefore we define some relaxed notions of satisfaction that characterize the "interestingness" of the rules.

Definition 8 Let R be a relation over the relation scheme S. Let G be a genscheme of S.

- The *support* of the genscheme G in R, denoted $sp(G, R)$, measures the fraction of pairs of tuples that are G-equivalent:

$$sp(G, R) := \frac{|\{\ \{t, s\} \subseteq R \mid t \neq s \wedge t \sim_G s\ \}|}{|\{\ \{t, s\} \subseteq R \mid t \neq s\ \}|} \ .$$

- The *support* of a RUD $G \to H$ in R, denoted $sp(G \to H, R)$, is simply the support of the genscheme G in R.

- The *confidence* of a RUD $G \to H$ in R is defined as follows.

$$cf(G \to H, R) := \frac{sp(G \wedge H, R)}{sp(G, R)} \ .$$

□

Our notions of support and confidence are natural adaptations of the notions with the same name used in association rule mining [AIS93]. Notice that the confidence of a RUD $G \to H$ is the conditional probability that two distinct tuples are H-equivalent provided that they are already G-equivalent; the support is the probability that the conditioning event occurs. 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 minor 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.

Theorem 4 gives an effective way for computing support and confidence.

Theorem 4 *Let G, H be genschemes, and R a relation. Let $\widetilde{n}$ denote the number $n(n-1)$, for each positive natural number n. Then:*

$$\begin{aligned} sp(G \to H, R) &= \frac{\sum_{S \in (R/\sim_G)} \widetilde{|S|}}{\widetilde{|R|}} \\ cf(G \to H, R) &= \frac{\sum_{S \in (R/\sim_G)} \sum_{T \in (S/\sim_H)} \widetilde{|T|}}{\sum_{S \in (R/\sim_G)} \widetilde{|S|}} \end{aligned}$$

FILES	$(Pa, \texttt{PATH})$	$(Fn, \texttt{FNAME})$	$(Si, \texttt{B})$
	`C:\Research\`	`Paper.doc`	41 590
	`C:\Teaching\`	`Course.doc`	65 024
	`C:\Research\`	`Paper.ps`	1 327 081
	`C:\Teaching\`	`Course.ps`	134 302
	`D:\Private\`	`Resume.doc`	754 348
	`D:\Private\`	`Letter.doc`	251 738
	`D:\Teaching\`	`Program.exe`	1 040 960
	`D:\Teaching\`	`Handouts.ps`	370 079
	`D:\Teaching\`	`Slides.ps`	418 325
	`D:\Teaching\`	`Spreadsheet.ps`	93 428

Figure 4: Equivalence classes induced by $(Pa, \texttt{DRIVE})$ (double lines) and $(Pa, \texttt{DRIVE})(Fn, \texttt{EXTENSION})$ (single lines).

Proof. Straightforward. □

Consider, for example, the relation *FILES* of Figure 2, and the RUD $(Pa, \texttt{DRIVE}) \to (Fn, \texttt{EXTENSION})$. Figure 4 shows the equivalence classes induced by $(Pa, \texttt{DRIVE})$ (double lines) and $(Pa, \texttt{DRIVE})(Fn, \texttt{EXTENSION})$ (single lines). There are two equivalence classes induced by $(Pa, \texttt{DRIVE})$ with 4 and 6 elements respectively. So the support of the RUD under consideration is $\frac{\widetilde{4}+\widetilde{6}}{\widetilde{10}} = \frac{7}{15}$. Both equivalence classes are further partitioned giving five equivalence classes induced by $(Pa, \texttt{DRIVE})(Fn, \texttt{EXTENSION})$. The confidence of the RUD under consideration is $\frac{\widetilde{2}+\widetilde{2}+\widetilde{2}+\widetilde{1}+\widetilde{3}}{\widetilde{4}+\widetilde{6}} = \frac{2}{7}$.

5.2 RUDMINE

The problem RUDMINE is motivated by the following example. Assume a relation R over the relation scheme

$$(\mathit{Item}, \texttt{ITEM})(\mathit{Day}, \texttt{DAY})(\mathit{Store}, \texttt{STORE})(\mathit{Price}, \texttt{CENT}) \ .$$

An analyst wants to find out whether there are temporal or spatial patterns governing the price. He decides that prices in integral Euros are sufficiently accurate to begin with. That is, he is looking for "strong" RUDs of the form $G \to (\mathit{Price}, \texttt{EURO})$ where G is any genscheme that does not contain the attribute *Price*. An example is the RUD

$$(\mathit{Item}, \texttt{ITEM})(\mathit{Day}, \texttt{YEAR})(\mathit{Store}, \texttt{AREA}) \to (\mathit{Price}, \texttt{EURO}) \ ,$$

expressing that the Euro price of an item does not vary within a year and area. By a strong RUD, we mean a RUD of which the support and the confidence exceed specified threshold values.

Definition 9 A *RUDMINE problem* is a quintet (S, H, s^*, c^*, R) where S is a relation scheme, H is a genscheme of S, s^* and c^* are real numbers between 0 and 1, and R is a relation over S.

The *answer* to the RUDMINE problem (S, H, s^*, c^*, R) is the set of all genschemes G of S satisfying: $sp(G \to H, R) \geq s^*$, $cf(G \to H, R) \geq c^*$, and G and H have no attributes in common. □

So RUDMINE asks which RUDs with a fixed right-hand genscheme attain certain specified support and confidence thresholds. Fixing the right-hand side in a RUDMINE problem is intuitive: in the previous example, the analyst would not be interested in RUDs where the level in the right-hand is coarser than `EURO`.

It is not hard to prove that the associated decision problem—i.e., the problem asking whether a given RUDMINE problem has a nonempty solution—is **NP**-hard. The proof uses a reduction from 3SAT along the lines of [WM98]. The exponentiality is in the number of attributes; the problem can be solved in polynomial time in the number of tuples. An algorithm for solving RUDMINE is described next.

6 Algorithm

6.1 Stratumwise Search

The input of the algorithm is a RUDMINE problem (S, H, s^*, c^*, R). Clearly, since no genscheme in the solution can have attributes in common with H, every genscheme in the solution must be a genscheme of S', where S' is the relation scheme obtained from S by omitting all attributes that occur in H. We know that all genschemes of S' are organized in the stratified roll-up lattice ${S'}^{\diamond}$ (Theorem 3). The main idea is to traverse ${S'}^{\diamond}$ stratumwise, starting at the top $\top$ and moving towards the bottom $\bot$, and retain all genschemes of ${S'}^{\diamond}$ that are solutions, while avoiding visiting genschemes that cannot possibly be solutions based on the information obtained so far.

The outline of the algorithm is given in Figure 5, and is an application of levelwise search [MT97]. After the initialization, the lattice is traversed stratumwise in the outer while-loop. The *evaluation phase* of the k^{th} iteration determines which candidate genschemes at stratum k are solutions. The subsequent *candidate generation phase* generates the genschemes at stratum $k+1$ that can possibly turn out to be solutions in the next iteration. The following monotonicity property will be used for pruning in the candidate generation phase: if a genscheme G has insufficient support, then any genscheme F with $F \trianglelefteq G$ will also have insufficient support.

Theorem 5 *Let F, G be genschemes of the relation scheme S. If $F \trianglelefteq G$, then for all relations R over S, $sp(F, R) \leq sp(G, R)$.*

PROOF. Straightforward. □

Hence, a genscheme F at level $k+1$ cannot possibly be a solution if some of its direct specializations G at level k fails the support threshold. The genschemes at level k with sufficient support are collected in L_k. Moreover, it is clear that every solution G at level $k+1$ must be covered by a genscheme with sufficient support at level k ($k > 1$). The candidate generation phase can thus be elaborated as shown in Figure 6.

The generation of direct specializations and generalizations is based on the property expressed by Theorem 2. There can be two distinct genschemes F_1 and F_2 in L_k that both cover the same genscheme G; that is, $G \in spec(F_1)$ and $G \in spec(F_2)$. In that case we need to avoid considering G twice, hence the addition "—unless G already tested—" in the algorithm of Figure 6. In practice, this can be achieved by using a spanning tree of the roll-up lattice; in this spanning tree, G will be a child of either F_1 or F_2, but not both. We illustrate this by an example.

```
Input:
    RUDMINE-problem $(S, H, s^*, c^*, R)$
Output:
    The solution to the problem.
Method:
 1. $C_1 = \{\top\}$
 2. $k = 1$
 3. while $C_k \neq \{\}$ loop
 4.         % Evaluation phase
 5.         $L_k = \{\}$
 6.         for $G \in C_k$ loop
 7.             if $sp(G \to H, R) \geq s^*$ then
 8.                 $L_k = L_k \cup \{G\}$
 9.                 if $cf(G \to H, R) \geq c^*$ then output $G$
10.                 end-if
11.             end-if
12.         end-loop
13.         % Candidate generation phase
14.         $C_{k+1}$ = all genschemes at stratum $k+1$ except those that cannot possibly be
                    solutions, given all information obtained so far.
15.         $k = k + 1$
16. end-loop
```

Figure 5: Stratumwise algorithm for solving RUDMINE.

```
13.   % Candidate generation phase
14.1  $C_{k+1} = \{\}$
14.2  for $F \in L_k$ loop
14.3      for $G \in spec(F)$—unless $G$ already tested—loop
14.4          if every $H \in gen(G)$ is in $L_k$ then
14.5              $C_{k+1} = C_{k+1} \cup \{G\}$
14.6          end-if
14.6      end-loop
14.7  end-loop
```

Figure 6: Elaboration of the candidate generation phase.

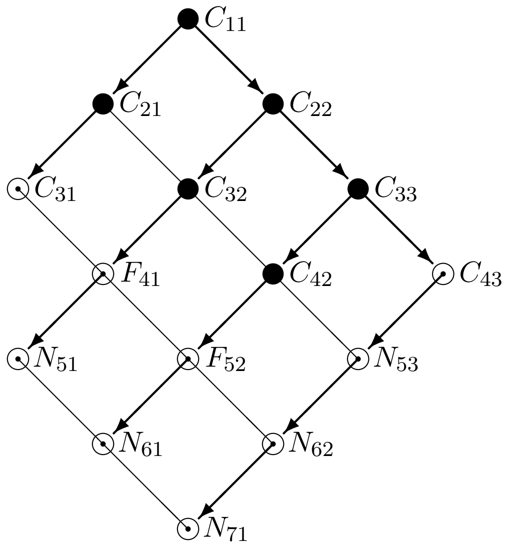


Figure 7: Roll-up lattice. The full-circled genschemes satisfy the support threshold; the open-circled genschemes falsify the support threshold. The vectors constitute the spanning tree. Nodes labeled C_{ij} are considered candidate solutions at level i. The nodes F_{41} and F_{52} are no candidates because they are covered by some genscheme that is no solution. Nodes labeled N are never visited because they are not covered by a solution in the spanning tree.

Consider the roll-up lattice in Figure 7. The thick vectors indicate the spanning tree. We start with $C_1 = \{C_{11}\}$, and obtain $L_1 = \{C_{11}\}$ as C_{11} turns out to satisfy the support threshold. The first candidate generation phase yields $C_2 = \{C_{21}, C_{22}\}$. In the subsequent evaluation phase, both C_{21} and C_{22} turn out to satisfy the support threshold, hence $L_2 = \{C_{21}, C_{22}\}$. The next candidate generation phase yields $C_3 = \{C_{31}, C_{32}, C_{33}\}$. Note that C_{32} is generated only once, along the vector from C_{22} pointing to it; that is, C_{32} is not generated from C_{21} because this edge of the roll-up lattice is not part of the spanning tree. Since C_{31} fails the support threshold, we obtain $L_3 = \{C_{32}, C_{33}\}$. In the subsequent candidate generation phase, the genscheme F_{41} is considered no candidate because C_{31} is no solution. Hence, $C_4 = \{C_{42}, C_{43}\}$. Since C_{43} has insufficient support, we obtain $L_4 = \{C_{42}\}$. In the subsequent candidate generation phase, the genscheme F_{52} is considered no candidate because F_{41} is no solution. Hence, $C_5 = \{\}$ and the algorithm terminates.

A spanning tree can be defined easily by using a lexical order on genschemes. Among all direct specializations of a given genscheme G, the one that is the smallest according to this lexical order will point to G in the spanning tree.

6.2 Counting

In this subsection we explain how the functions $sp(\cdot,\cdot)$ and $cf(\cdot,\cdot)$ can be evaluated efficiently.

By Theorem 4, for calculating the support and the confidence of a rule $G \to H$, we need the number of elements in each class of the partitions induced by $\sim_G$ and $\sim_{G\wedge H}$. In order to calculate the number of elements in each class, we use *histograms*. The histogram of a genscheme G stores the number of tuples in each partition induced by $\sim_G$.

In Figure 8, the histograms of both $(Pa, \mathtt{DRIVE})$ and $(Pa, \mathtt{DRIVE})(Fn, \mathtt{EXTENSION})$ over the relation *FILES* are given. Note the difference with Figure 4, where the partitions induced by these two genschemes are given. The histograms exhibit a hierarchical structure. For the

$(Pa,$ DRIVE$)$	COUNT
C:	4
D:	6

(a)

$(Pa,$ DRIVE$)$	$(Fn,$ EXTENSION$)$	COUNT
C:	.doc	2
	.ps	2
D:	.doc	2
	.exe	1
	.ps	3

(b)

Figure 8: The histograms of the genschemes $(Pa,$ DRIVE$)$ and $(Pa,$ DRIVE$)(Fn,$ EXTENSION$)$.

histogram of Figure 8 (b), for example, the main entries are `C:` and `D:`. The main entry `C:` is further subdivided into subentries `.doc` and `.ps`. And so on. In our implementation, we take advantage of this hierarchical structure and store histograms as trees. The hierarchical structure is further exploited to reduce storage requirements by sharing main entries that are common to two histograms. For example, the two histograms of Figure 8 could share the entries for the common subscheme $(Pa,$ DRIVE$)$.

The right-hand genscheme is fixed in a RUDMINE problem. As a consequence, this right-hand genscheme would appear in all histograms that are constructed for computing confidences. To avoid this duplication of right-hand side data, we start by *fragmenting* the input dataset according to the fixed right-hand genscheme. For example, for the RUDMINE problem $(S, (Fn,$ EXTENSION$), s^*, c^*, FILES))$, where S is the relation schema of *FILES*, the relation *FILES* is first fragmented according to the fixed genscheme $(Fn,$ EXTENSION$)$. This is shown in Figure 9 (*left*). From this fragmented input relation, we can construct fragments of the histogram of $G \wedge (Fn,$ EXTENSION$)$, for any genscheme G. For $G = (Pa,$ DRIVE$)$, the three fragments of the histogram of $(Pa,$ DRIVE$)(Fn,$ EXTENSION$)$ are shown in Figure 9 (*right*). These fragments can then be merged to obtain all figures needed to compute the support and confidence of the RUD $(Pa,$ DRIVE$) \rightarrow (Fn,$ EXTENSION$)$. The figure shows one merge operation; in practice, merge can be done two histograms at a time. The main advantage of this fragmentation technique is a better usage of main memory. The original (unfragmented) histograms can become very large as one goes down the roll-up lattice, and may not fit into main memory. With fragmentation, we only need to store smaller fragments, and we can compute fragments in main memory until it is filled.

Note incidentally that although the examples considered so far show RUDs with singleton right-hand sides, the algorithm allows any number of attributes at the right-hand side of a RUD.

7 Experiments

A prototype of our algorithm has been implemented in Visual C++ 6.0 and its performance has been tested. In this section, we describe experiments on four different datasets: COUNTRIES, FILES, KDD98CUP, and SYNTH. The main characteristics of the datasets are given in Table 1. The COUNTRIES dataset is rather small, but is included because of the many natural roll-up functions and the interest of discovered rules. Although the FILES dataset has only seven attributes, it is associated with a rich roll-up lattice. The KDD98CUP dataset is large, but the number of roll-ups is limited. The SYNTH dataset is synthetically generated using the data generator of the IBM QUEST-group for transactional data [AMS$^+$96], as explained later on. Together the four datasets show a wide variety of characteristics. We first describe the content and the discovered RUDs for each dataset, and then we provide

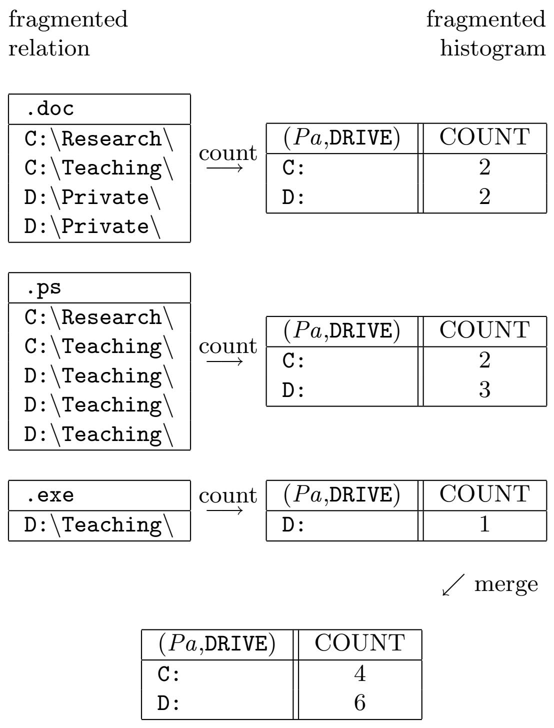


Figure 9: Technique used for computing support and confidence.

Dataset	Number of attributes	Number of tuples
COUNTRIES	12	195
KDD98CUP	100	97 000
FILES	7	45 000
SYNTH	200	100 000

Table 1: Size of the datasets.

performance figures.

7.1 Datasets and Mined RUDs

7.1.1 COUNTRIES

Each tuple of this dataset contains geographical, social, political, and other information pertaining to a single country. It is based on the CIA factbook [FAC01], global statistics [van01], and the MONDIAL database [May99]. The attributes and the most important roll-ups are:

Country The name of the country. Roll-ups to sub-continent (`SUBCONT`) and continent (`CONT`) are natural.

Pop Population size. This attribute can be rolled up to the nearest 100 000 (`100th`), 1 000 000 (`1mil`), or 10 000 000 (`10mil`).

InfMor Infant mortality per 1000 births; rolls up to nearest 10, 20 (`20`), 30 (`30`).

Res Natural resources of the country; roll-ups to resources like `GAS`, `OIL`, `FISH`, `LUMBER`, `COAL`, `IRON`, indicate whether the resource is present or not.

Econ Economical situation. This attribute contains data about the occupation of the inhabitants of the country. The occupations are split into three categories: agriculture, industry, and service. This attribute can be rolled up to the percentage of people working in service (`SERV`), industry (`INDU`), and agriculture (`AGRI`). These percentages again can be rolled up to the nearest 10%, 20%, 30%,... We can also order the three categories by importance (`ORDER`) or select the main occupation (`MOCC`).

Geo Geographical information; roll-ups to landscape elements like `SEA`, `DES` (for desert), `MOUNTAIN`, indicate whether the element is present in the country.

Rel Religious situation in the country. This attribute rolls up to the percentages of muslims, catholics, hindus. The percentages again roll up to the nearest 10%, 20%,...

PopGr Relative population growth in percent. Rolls up to .1%, 1% (`1%`), and to `SIGN`, indicating whether the growth is positive or negative.

Area Area of the country in square miles, rolls up to 100 000 sq. miles (`100thsq.m.`), and 1 000 000 sq. miles (`1milsq.m.`).

GDP Gross product of the country. This can again be rounded to 10 000s (`10th`) and 100 000s (`100th`).

PScore Score for different political parameters such as press freedom (`PRESS`), civil rights (`CIVIL`), amount of democracy (`DEMO`) and corruption (`CORR`). The percentages roll up to 10%, 20%. Civil rights rolls up to a qualitative ranking (`CIVILHL`).

HDI Score from the human development index. This score includes percentages for life expectation (`LIFE`), GDP per inhabitant (`GDP`), education (`EDU`), and a total score (`TOT`).

Due to the many roll-ups, the roll-up lattice contained more than 36×10^{15} genschemes. Table 2 shows some RUDs of interest found in this dataset. The first three RUDs, for example, indicate that countries that score similarly w.r.t. the presence of gas and population size (up to `10mil`), also score similarly w.r.t. gross product (up to `100th`). The last rule

$$(\mathit{GDP}, \texttt{100th})(\mathit{Area}, \texttt{100thsq.m.}) \rightarrow (\mathit{Pop}, \texttt{10mil})$$

means that countries with the same GDP at the `100th` level and the same area at the `100thsq.m.` level, are likely (probability of 89%) to have the same population size (up to `10mil`). This is a strong rule, because the probability that two randomly chosen countries agree on population at level `10mil`, is only 40%.

7.1.2 FILES

The dataset FILES contains characteristics of files on a number of PCs running Windows NT. These data are fairly easy to obtain, roll-ups are natural, and our familiarity with the dataset facilitated interpreting the output of the algorithm. Each file is described by one record in the dataset. Attributes include:

Fn is the file name. The level `EXTENSION` denotes the file name's extension. For example, the file name `document.txt` rolls up to `.txt` in `EXTENSION`.

G	$\rightarrow$	H	$sp(G \rightarrow H)$	$cf(G \rightarrow H)$	$cf(\{\} \rightarrow H)$
$(Res, \texttt{GAS})$	$\rightarrow$	$(GDP, \texttt{100th})$	57%	70%	59%
$(Pop, \texttt{10mil})$	$\rightarrow$	$(GDP, \texttt{100th})$	40%	87%	59%
$(Res, \texttt{GAS})(Pop, \texttt{10mil})$	$\rightarrow$	$(GDP, \texttt{100th})$	29%	90%	59%
$(Econ, \texttt{MOCC})(InfMor, 30)$	$\rightarrow$	$(HDScore, \texttt{TOT50\%})$	14%	85%	58%
$(Econ, \texttt{MOCC})(InfMor, 30)(Geo, \texttt{DES})$	$\rightarrow$	$(HDScore, \texttt{TOT50\%})$	12%	88%	58%
$(Econ, \texttt{MOCC})(InfMor, 30)(Rel, \texttt{Mus})(Geo, \texttt{DES})$	$\rightarrow$	$(HDScore, \texttt{TOT50\%})$	10%	90%	58%
$(Econ, \texttt{SERV20\%})(InfMor, 30)$	$\rightarrow$	$(HDScore, \texttt{GDP50\%})$	12%	82%	53%
$(Econ, \texttt{MOCC})(InfMor, 30)(Geo, \texttt{DES})$	$\rightarrow$	$(HDScore, \texttt{GDP50\%})$	12%	88%	53%
$(Econ, \texttt{MOCC})(Econ, \texttt{AGRI20\%})(InfMor, 30)$	$\rightarrow$	$(HDScore, \texttt{GDP50\%})$	10%	88%	53%
$(Econ, \texttt{MOCC})(Rel, \texttt{Mus})(Geo, \texttt{DES})$	$\rightarrow$	$(HDScore, \texttt{EDU50\%})$	21%	81%	62%
$(Country, \texttt{CONT})$	$\rightarrow$	$(HDScore, \texttt{LIFE50\%})$	21%	71%	56%
$(InfMor, 30)$	$\rightarrow$	$(HDScore, \texttt{LIFE50\%})$	29%	73%	56%
$(PopGr, \texttt{SIGN})(InfMor, 30)$	$\rightarrow$	$(HDScore, \texttt{LIFE50\%})$	23%	73%	56%
$(Econ, \texttt{MOCC})(PScore, \texttt{DEMO50\%})(HDScore, \texttt{GDP50\%})$	$\rightarrow$	$(InfMor, 30)$	11%	64%	29%
$(Country, \texttt{CONT})(HDScore, \texttt{GDP50\%})$	$\rightarrow$	$(PopGr, \texttt{1\%})$	14%	40%	24%
$(HDScore, \texttt{EDU10\%})$	$\rightarrow$	$(PopGr, \texttt{1\%})$	15%	40%	24%
$(InfMor, 20)$	$\rightarrow$	$(PopGr, \texttt{1\%})$	19%	41%	24%
$(PScore, \texttt{CORR20\%})(Pop, \texttt{10mil})$	$\rightarrow$	$(GDP, \texttt{mil})$	25%	100%	91%
$(Econ, \texttt{AGRI20\%})(InfMor, 20)$	$\rightarrow$	$(PScore, \texttt{DEMO50\%})$	13%	64%	36%
$(Econ, \texttt{AGRI20\%})(Area, \texttt{1milsq.m.})(Rel, \texttt{Mus})$	$\rightarrow$	$(PScore, \texttt{CIVILRNK})$	21%	70%	49%
$(Geo, \texttt{SEA})(InfMor, 30)$	$\rightarrow$	$(PScore, \texttt{PRESS50\%})$	10%	72%	49%
$(Econ, \texttt{AGRI20\%})(Rel, \texttt{Mus})(InfMor, 30)$	$\rightarrow$	$(PScore, \texttt{PRESS50\%})$	14%	76%	49%
$(Pop, \texttt{10mil})$	$\rightarrow$	$(Area, \texttt{1milsq.m.})$	40%	87%	73%
$(Pop, \texttt{mil})$	$\rightarrow$	$(Area, \texttt{1milsq.m.})$	7%	95%	73%
$(Area, \texttt{100thsq.m.})$	$\rightarrow$	$(Pop, \texttt{10mil})$	22%	78%	40%
$(GDP, \texttt{10th})$	$\rightarrow$	$(Pop, \texttt{10mil})$	20%	86%	40%
$(GDP, \texttt{100th})(Area, \texttt{100thsq.m.})$	$\rightarrow$	$(Pop, \texttt{10mil})$	18%	89%	40%

Table 2: RUDs for the COUNTRIES dataset.

Pa is the path to the file. The level DRIVE denotes the disk, and MAIN the main directory. For example, the path `C:\winnt\profiles\desktop\` rolls up to `C:` in DRIVE and to `winnt` in MAIN.

Access is the last time the file was accessed.

Write is the last time the file was modified.

Attrib is an array of file properties, indicating whether or not the file is hidden, a subdirectory, a system file, and so on. The level SD only retains the boolean property indicating whether or not the file is a directory.

The roll-up lattice contained up to 64 800 genschemes. Table 3 shows some RUDs found in this dataset.

As expected, many RUDs had low support. The support of a RUD with left-hand side $(Pa, \texttt{DIR})$, for example, 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. Some caution is in order when interpreting confidence measures, however. For example, RUDs with $(Pa, \texttt{DRIVE})$ at the right-hand side happened to have a high confidence independent of the left-hand side, just because the confidence of $\{\} \rightarrow (Pa, \texttt{DRIVE})$ is high. The confidence of $\{\} \rightarrow (Pa, \texttt{DRIVE})$ is the probability that any two files reside on the same drive, which is high if there is only a small number of

G	$\rightarrow$	H	$sp(G \rightarrow H)$	$cf(G \rightarrow H)$	$cf(\{\} \rightarrow H)$
$(Pa, \texttt{MAIN})$	$\rightarrow$	$(Fn, \texttt{EXTENSION})$	0.10	0.26	0.056
$(Access, \texttt{WEEKDAY})$	$\rightarrow$	$(Pa, \texttt{DRIVE})$	0.23	0.57	0.37
$(Pa, \texttt{MAIN})$	$\rightarrow$	$(Access, \texttt{WEEKDAY})$	0.10	0.62	0.23
$(Pa, \texttt{MAIN})$	$\rightarrow$	$(Write, \texttt{YEAR})$	0.10	0.65	0.26
$(Pa, \texttt{DRIVE})(Create, \texttt{WEEKDAY})$	$\rightarrow$	$(Write, \texttt{YEAR})$	0.14	0.53	0.26
$(Pa, \texttt{DRIVE})(Write, \texttt{YEAR})(Access, \texttt{YEAR})$	$\rightarrow$	$(Attrib, \texttt{SD})$	0.11	0.96	0.90

Table 3: RUDs for the dataset FILES.

drives. The confidence of a RUD $G \rightarrow H$ has therefore to be compared with the confidence of $\{\} \rightarrow H$ to verify its statistical significance.

The RUD $(Pa, \texttt{MAIN}) \rightarrow (Write, \texttt{YEAR})$ says that files in the same main directory tend to be last modified in the same year. This RUD has a confidence of 65% while the probability that two randomly chosen files are last modified in the same year is only 26%.

7.1.3 KDD98CUP

The KDD98CUP dataset contains information about donations; the dataset is available in the UCI KDD archive [Bay99]. Every record describes a donor. In a preprocessing phase, we removed attributes having the same value in more than 95% of the tuples. In the experiments, we varied the number of attributes from 10 to 100. Attributes include:

Dob is the date of birth. Classical temporal roll-ups apply, for example, to `YEAR`.

Income is the household income of the donor.

Wealth is a rating of the donors wealth status.

Hit is the number of times the donor has responded to a mail order offer other than PVA's. Each positive *Hit*-value rolls up to 'Yes' in the level `YN`, while the value 0 rolls up to 'No.' So `YN` indicates whether or not the donor has responded to an offer.

Dsrc is the source of overlay data, and indicates which third-party data source the donor matched against.

Books is the number of times the donor has responded to mail order offers for books. Like for the attribute *Hit*, each positive *Books*-value rolls up to 'Yes' in `YN`, while the value 0 rolls up to 'No'.

Numchld is the number of children.

Table 4 shows some RUDs found in this dataset. There are a number of relatively strong RUDs with *Wealth* at the right-hand side. For example, the RUD $(Hit, \texttt{YN})(Books, \texttt{YN})\ Dsrc \rightarrow Wealth$ with a confidence of 63% and a support of 18% says that donors having the same response behavior to other mailings and especially to mailings for books, and who match against the same third-party data source, tend to have the same wealth rating.

7.1.4 SYNTH

The SYNTH dataset is generated using the IBM QUEST-group data generator. This data generator originally generates data for transaction databases. We transform the transactional data into a relational table with 200 attributes $A_1, \ldots, A_{200}$ as follows.

G	$\rightarrow$	H	$sp(G \rightarrow H)$	$cf(G \rightarrow H)$	$cf(\{\} \rightarrow H)$
$(Dob, \texttt{YEAR})$	$\rightarrow$	*Wealth*	0.07	0.50	0.27
$(Income)$	$\rightarrow$	*Wealth*	0.14	0.46	0.27
$(Hit, \texttt{YN})$	$\rightarrow$	*Wealth*	0.51	0.47	0.27
$(Books, \texttt{YN})$	$\rightarrow$	*Wealth*	0.65	0.38	0.27
Dsrc	$\rightarrow$	*Wealth*	0.32	0.41	0.27
$(Hit, \texttt{YN})(Books, \texttt{YN})$ *Dsrc*	$\rightarrow$	*Wealth*	0.18	0.63	0.27
Income $(Books, \texttt{YN})$	$\rightarrow$	*Wealth*	0.10	0.60	0.27
Income	$\rightarrow$	*Numchld*	0.14	0.81	0.76

Table 4: RUDs for the dataset KDD98CUP.

1. We generate a transaction database with 1000 different items.
2. We group the 1000 items in 200 groups of 5. Let $I_1^i, I_2^i, \ldots, I_5^i$ be the five items associated with the i^{th} attribute.
3. The domain of each attribute A_i is the set of integers $\{0, 1, 2, \ldots, 2^5 - 1\}$. For each transaction, we construct a tuple t such that the j^{th} bit in the binary representation of $t(A_i)$ indicates whether I_j^i is present (value one) or absent (value zero) in the transaction under consideration $(1 \leq j \leq 5, 1 \leq i \leq 200)$.
4. As roll-up functions, we use bit-wise AND with randomly selected integers in $\{0, 1, 2, \ldots,$ $2^5 - 1\}$; e.g., the roll-up function $R_{\texttt{b}00101}$ is associated with the number 5 is defined as follows: $R_{\texttt{b}00101}(x)$ is the bit-wise AND between the binary representation of x and $\texttt{b}00101$. For example, $R_{\texttt{b}00101}(12) = \texttt{b}00101$ AND $\texttt{b}01100 = \texttt{b}00100 = 8$. We do not use all these functions, for each attribute we randomly take a selection of these 512 roll-up functions.

The resulting synthetic dataset has a large number of attributes and roll-up functions, and a complex roll-up lattice. The RUDs discovered in the SYNTH dataset are not shown because they are artificial and lack meaning.

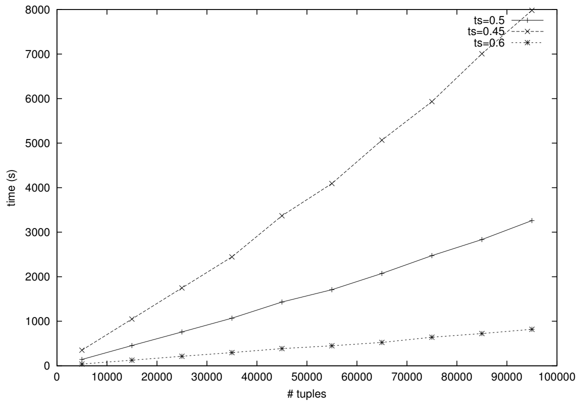
7.2 Performance Evaluation

It was investigated how the execution time behaves in function of the size of the dataset, the size of the roll-up lattice, and the support threshold. Unless stated otherwise, the upper half of each figure shows a graph for KDD98CUP (a) followed by a graph for FILES (b). The lower half correspond to the SYNTH dataset restricted to 100 attributes (c) and 200 attributes (d) respectively.

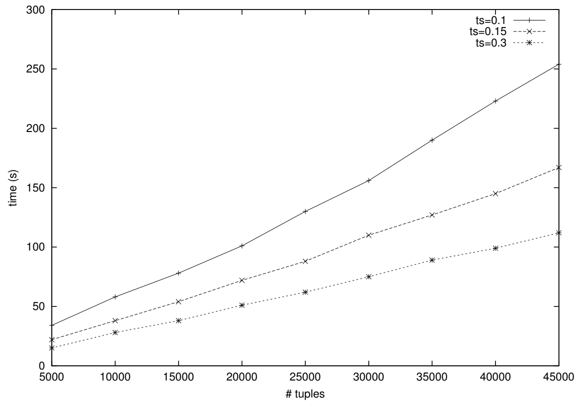
7.2.1 Scale-up

Figure 10 shows the execution time as we increase the number of tuples of the input dataset, for different levels of threshold support. In Figure 10 (a), i.e., for the KDD98CUP data, the fixed right-hand genscheme was *Wealth*; for Figure 10 (b), i.e., for the FILES data, the fixed right-hand genscheme was $(Fn, \texttt{EXTENSION})$. The fixed right-hand genscheme for the experiments with the SYNTH-dataset was always the same attribute. The algorithm turns out to scale linearly for the three datasets. Note that the execution time is independent of the threshold confidence, as the confidence is not used for pruning.

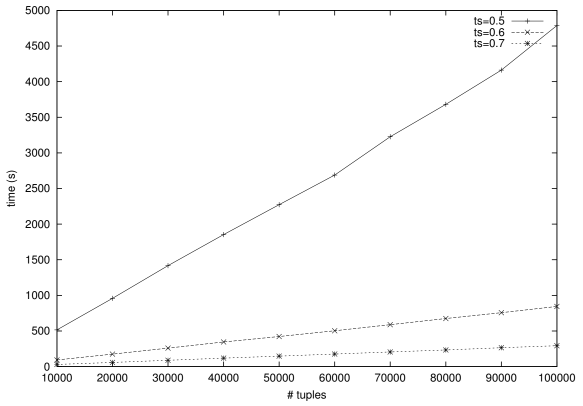
The support thresholds for the KDD98CUP data may seem rather high (45% and higher). However, it can be easily verified that the support for a genscheme (A, l) cannot drop below



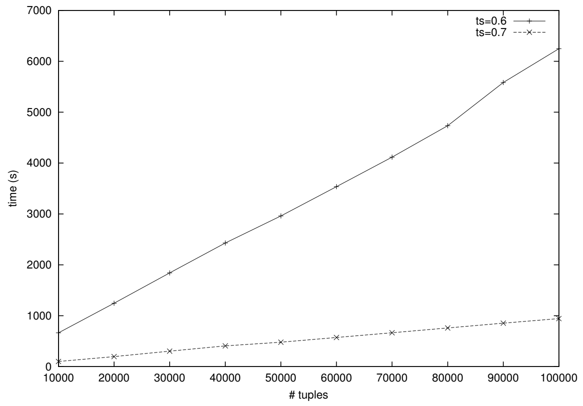
(a)



(b)



(c)



(d)

Figure 10: Execution time in function of the number of tuples.

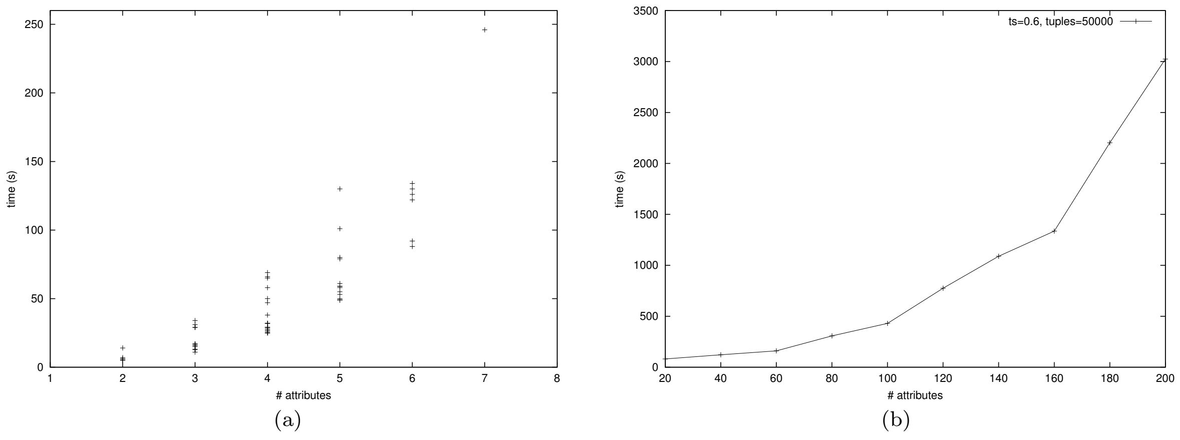


Figure 11: Execution time in function of the number of attributes, for FILES and SYNTH datasets.

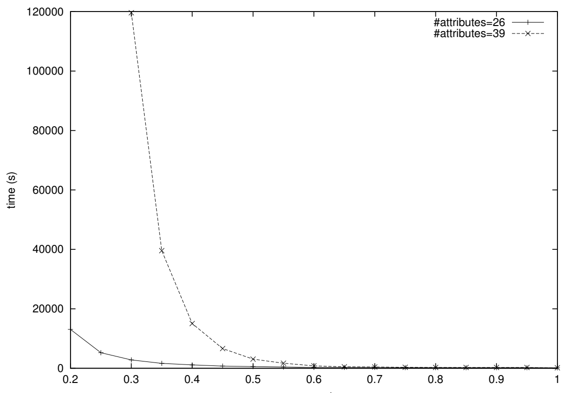
$\frac{\frac{|R|}{n}-1}{|R|-1} \approx \frac{1}{n}$, where R is the relation under consideration and n is the number of distinct A-values in R. This minimal support is reached if the genscheme partitions R in n classes each containing $\frac{|R|}{n}$ tuples that agree on A. Since in the KDD98CUP dataset, n is smaller than 5 for most attributes, high support thresholds are reasonable. The supports in the experiments with the SYNTH-dataset are high for the same reason.

Figure 11 shows the execution time as we increase the number of attributes that can appear at the left-hand side. The fixed right-hand side in Figure 11 (a) was $(Fn, \mathtt{EXTENSION})$. To obtain the figures for the FILES dataset, we ran the algorithm on each dataset that can be obtained from FILES by retaining only a subset of the attributes (always retaining the right-hand attribute Fn, of course). We used all 45000 tuples and a support threshold of 0.1. The graph shows that, for a given number of attributes, the execution time can vary considerably depending on which set of attributes is retained. Attributes with a small number of distinct values are generally easier to handle.

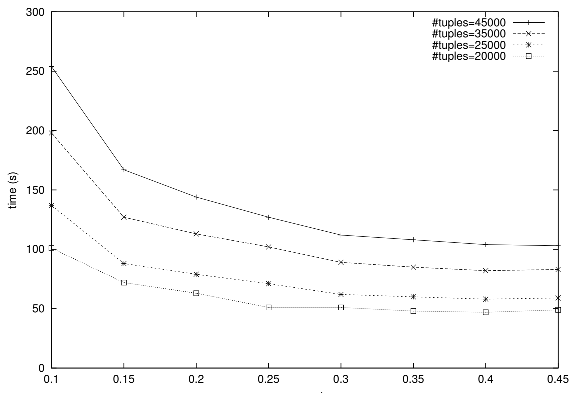
For the graph in Figure 11 (b), in the i^{th} experiment, we only retain the first $20 \times i$ attributes ($1 \leq i \leq 10$). Unlike for the FILES dataset, there is no significant difference between attributes of the SYNTH dataset, as was to be expected since the attributes are synthetic.

7.2.2 Pruning

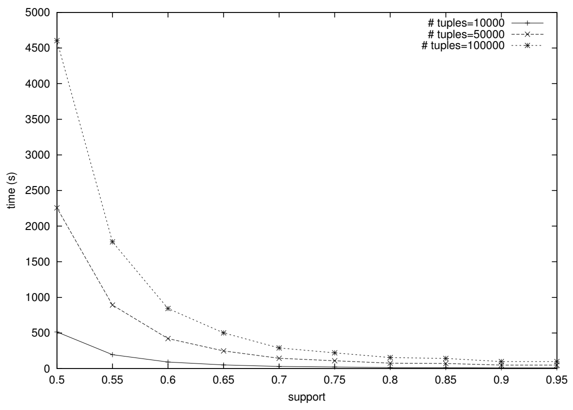
The size of the roll-up lattice is exponential in the number of attributes. In the worst case, our algorithm has to evaluate every genscheme of the roll-up lattice. Therefore, it is significant to ask how many nodes of the roll-up lattice have to be evaluated in practice. Figure 12 shows that the execution time drops rapidly as the support threshold increases. Figure 13 gives the explanation: increasing the support results in more genschemes of the roll-up lattice being pruned away. The graph shows that the pruning strategy used is quite effective. Significantly, the genschemes that are pruned away are near the bottom of the roll-up lattice; these genschemes have large histograms that would otherwise be expensive to evaluate. Figure 14 shows the execution time in function of the number of genschemes that are considered in the evaluation phase of the algorithm. The execution time grows little faster than linear, even for the SYNTH dataset, which has a rich roll-up lattice. The explanation for the super-linearity is



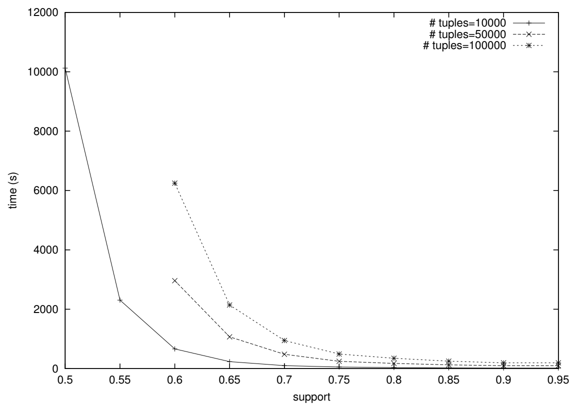
(a)



(b)

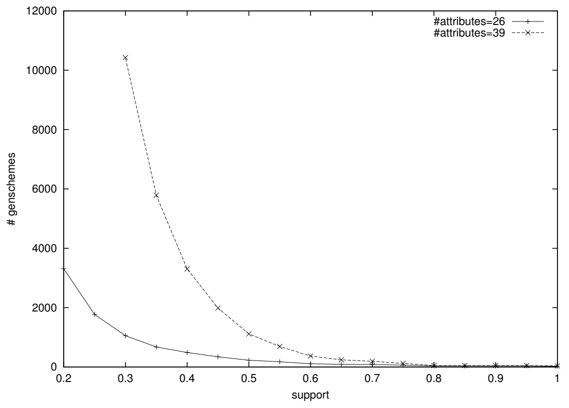


(c)

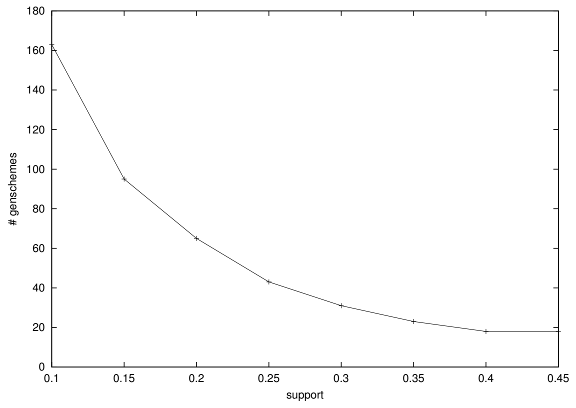


(d)

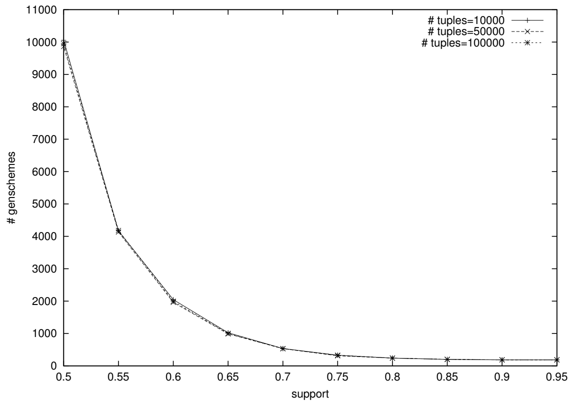
Figure 12: Execution time in function of the support threshold.



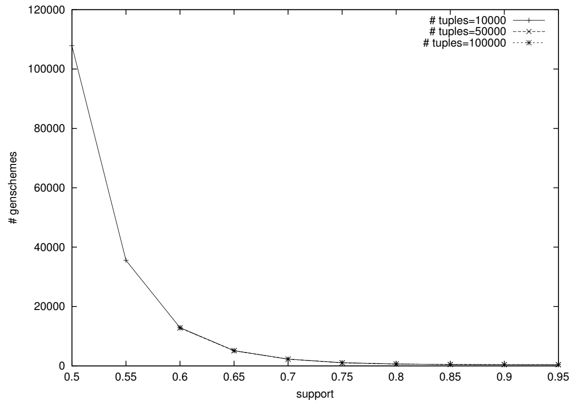
(a)



(b)

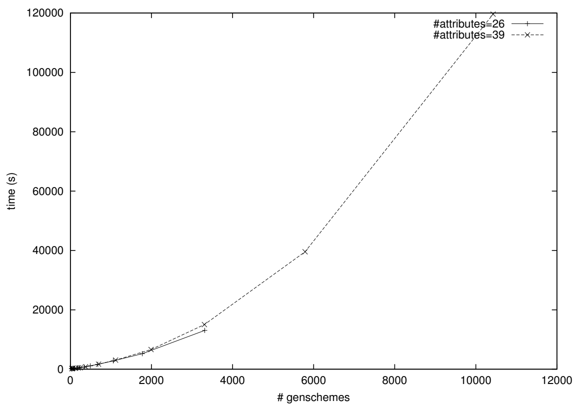


(c)

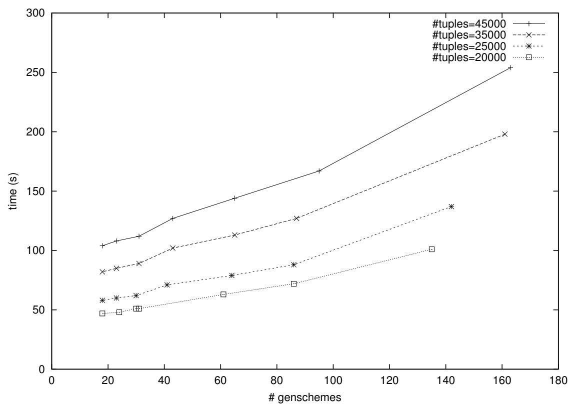


(d)

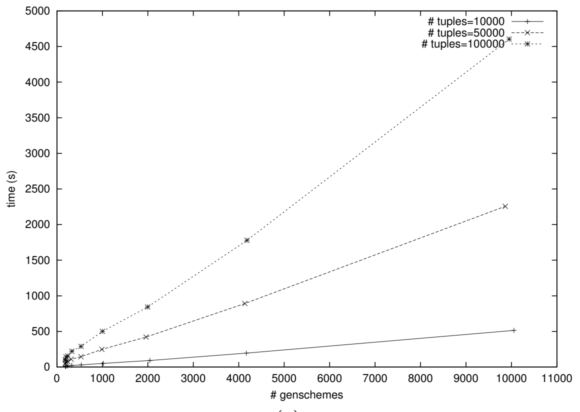
Figure 13: Number of genschemes considered in the evaluation phase in function of the support threshold.



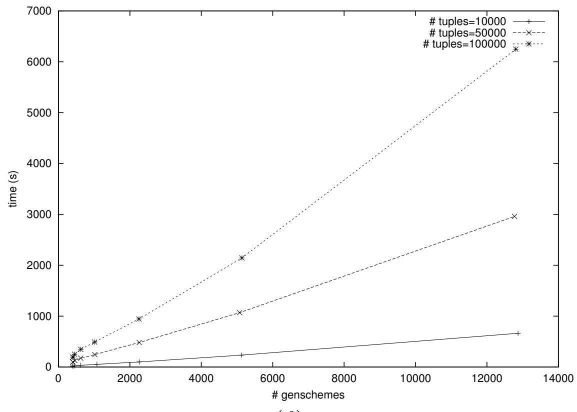
(a)



(b)



(c)



(d)

Figure 14: Execution time in function of the number of genschemes considered in the evaluation phase.

that while evaluating ever more genschemes, we are moving towards the bottom of the roll-up lattice where the genschemes tend to have ever larger histograms.

8 Related Work

We relate our work to theoretical approaches on lattice traversal, to mining functional dependencies, and to OLAP.

8.1 Lattice Traversal

Numerous data mining techniques involve traversing a lattice (or poset) of candidate patterns in a systematic way; our algorithm for mining RUDs is no exception. It is interesting to position our solution relative to the theoretical framework provided in [vNC98]. Assume a lattice $(P, \succeq)$, with top element $\top$, capturing some generalization order on patterns: $p \succeq q$ means that p is more general than q. Elementary specialization steps are executed by applying a so-called *downward refinement operator*: a mapping $\rho : P \rightarrow 2^P$ such that for every $p \in P$ and $q \in \rho(q)$, $p \succeq q$. Starting from an initial pattern, the search space is traversed from general to specific patterns by recursively applying ρ until a solution is found. For this search to be effective, the downward refinement operator ρ must satisfy a number of desirable properties. Notably, the downward refinement operator ρ is called:

- *Locally finite* if $\rho(p)$ is finite and computable for each $p \in P$. Clearly, a refinement operator without this property is of no practical use.
- *Complete* if for all $p, q \in P$ such that $p \succ q$, there exists a finite sequence $p = p_0, p_1, \ldots, p_n = q$ such that $p_i \in \rho(p_{i-1})$ $(i \in \{1, 2, \ldots, n\})$. When incomplete refinement operators are used, it is not guaranteed that all patterns in the search space can be attained. Nevertheless, if every "downward" search starts at the top element $\top$, then the notion of completeness can be relaxed: in that case it suffices that any pattern be derivable from $\top$ to guarantee the derivability of all patterns in the search space. Note that the algorithm outlined in Figure 5 starts at the top element of the roll-up lattice.
- *Proper* if for every $p \in P$ and $q \in \rho(p)$, $p \succ q$. Properness is interesting because it prevents us from infinite loops.

A downward refinement operator satisfying each of these three properties is called *ideal* in [vNC98]. It can be easily verified that our *spec*$(\cdot)$ operator is ideal in this respect. Nevertheless, this notion of ideality does not capture every desirable aspect of refinement. For example, it may be a waste of time and memory to visit the same pattern twice. We have avoided such double computations by imposing a spanning tree on the roll-up lattice. When using a spanning tree, $p \succ q$ does not imply that q is reachable form p along the spanning tree; nevertheless, q will be attainable from the top element. For example, in Figure 7, C_{21} is more general than C_{32}, but there is no path from C_{21} to C_{32} in the spanning tree.

The exploitation of (anti-)monotonicity properties to prune parts of the lattice is common in many data mining tasks; it is the main principle underlying apriori [AS94], and has been generalized and formalized in [MT97].

8.2 RUDs and Functional Dependencies

The discovery of FDs dates back to the eighties [MR86, MR87, BMT89, KMRS92], and has gained renewed interest recently [HKPT99, LPL00, NC01]. A comprehensive overview of FD mining can be found in [NC01]. There is an apparent relationship between FDs and RUDs, which goes as follows. Let S be a relation scheme over the roll-up scheme $(L, \preceq)$, and suppose we want to mine RUDs from a relation R over S, where a roll-up instance is implicitly understood. One can "flatten" S and the associated roll-up scheme into a classical relation scheme (i.e., a set of attributes), denoted $\mathit{flat}(S)$. Correspondingly, R and the associated roll-up instance are "flattened" in a relation, denoted $\mathit{flat}(R)$. More precisely, the construction is as follows: $\mathit{flat}(S)$ is the smallest set of attributes such that for every $(A, l) \in S$, for every level $l' \in L$ such that $l \preceq l'$, $\mathit{flat}(S)$ contains a new attribute that we will denote $A_{l'}$. Correspondingly, $\mathit{flat}(R)$ is the smallest set of tuples such that for every tuple $t \in R$, $\mathit{flat}(R)$ contains a tuple t' such that for every $(A, l) \in S$, for every level $l' \in L$ such that $l \preceq l'$, $t'(A_{l'}) = I_l^{l'}(t(A))$. The problem of mining RUDs in R then corresponds to the problem of mining FDs in $\mathit{flat}(R)$: if R satisfies the RUD $(A, l) \to (B, l')$, then $\mathit{flat}(R)$ satisfies the FD $A_l \to B_{l'}$; the extension to RUDs containing more than two attributes, is straightforward.

However, two caveats are in place. First, it can be easily verified that if $A_l, A_{l'}$ with $l \prec l'$ are two distinct attributes in $\mathit{flat}(S)$, then $\mathit{flat}(R)$ must necessarily satisfy the FD $A_l \to A_{l'}$. Although FDs between distinct attributes are not trivial in general, triviality of $A_l \to A_{l'}$ follows from our flattening construction. However, since existing algorithms for mining FDs are not developed to capture roll-up semantics, they must and will "discover" such trivialities. Our RUDMINE problem resembles the problem of discovering previously unknown FDs from relations when a set of known FDs has already been given. We are not aware of FD mining algorithms that exploit FDs that are given in advance. Second, different strength measures for FDs have been used by different authors, and FD mining algorithms obviously depend on the measure used. For example, an FD mining algorithm developed for the discovery of "exact" FDs is unlikely to be capable of discovering approximate FDs according to some relaxed notion of satisfaction. For both reasons, an experimental comparison of our RUD mining algorithm with pure FD mining algorithms seems inappropriate.

We believe that our notions of support and confidence are natural approximation measures for dependencies, as they allow a straightforward probabilistic interpretation: the confidence of a RUD $G \to H$ is the conditional probability that two distinct tuples are H-equivalent provided that they are already G-equivalent; the support is the probability that the conditioning event occurs. Our confidence measure is closely related to the approximation measure g_1 for FDs in [KM95]. Another natural approximation measure for FDs is in terms of the tuples that need to be deleted to bring a relation in accordance with the FDs [HKPT99].

8.3 OLAP

RUDs are intimately related to roll-up operations in OLAP. A motivation of RUD mining is to tell users which roll-up operations are beneficial. Our notion of *roll-up lattice* generalizes the lattice framework for OLAP provided in [HRU96]. Many computational issues involved in the algorithm to solve a RUDMINE problem are similar to those in computing data cubes. For 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.

9 Conclusions and Future Work

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. We have shown that the roll-up lattice in our framework can be divided into *strata*. This allows levelwise search.

RUDs have a promising 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 in the schema size, but polynomial in the number of tuples.

We have implemented and tested an algorithm for mining RUDs. Experimental results show that the execution time is linear in the number of tuples. If the threshold support is set low, the program may have to evaluate each genscheme of the roll-up lattice, whose size is exponential in the number of attributes. The algorithm proceeds in an apriori-like way, traversing the roll-up lattice stratumwise and using support as a pruning criterion. The lattice traversal is optimized by proceeding along a spanning tree. Histograms are used to efficiently calculate confidence and support values; several optimizing techniques, like sharing and fragmentation of histograms, have been implemented.

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. Some initial results in this direction are presented in [BW01].

A more incremental approach to RUD mining may be useful for OLAP. Mined RUDs can be used first to select good data cube dimensions, and should then be maintained incrementally during further modifications of the cubes. Incremental mining has already been addressed for FDs [Bel95, Sch93].

Another interesting research topic is the use of RUDs in (multi-dimensional) 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.

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.

[AMS$^+$96] R. Agrawal, H. Manilla, 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*, chapter 12, pages 307–328. AAAI Press/The MIT Press, 1996.

[AS94] A. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, 1994.

[Bay99] Stephen Bay. The UCI KDD archive, `http://kdd.ics.uci.edu/`. University of California, Irvine, Dept. of Information and Computer Sciences, 1999.

[BDE$^+$98] C. Bettini, C.E. Dyreson, W.S. Evans, R.T. Snodgrass, and X.S. Wang. A glossary of time granularity concepts. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*, number 1399 in LNCS State-of-the-art Survey, pages 406–413. Springer-Verlag, 1998.

[Bel95] S. Bell. Discovery and maintenance of functional dependencies by independencies. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 27–32. AAAI Press, 1995.

[BMT89] D. Bitton, J. Millman, and S. Torgersen. A feasibility and performance study of dependency inference. In *Proc. IEEE Int. Conf. on Data Eng.*, pages 635–641, Los Angeles, CA, 1989.

[BW01] R. Bassée and J. Wijsen. Neighborhood dependencies for prediction. In *Proc. Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mininig, PAKDD 2001*, LNAI 2053, pages 562–567. Springer, 2001.

[CT97] Luca Cabibbo and Riccardo Torlone. Querying multidimensional databases. In *Sixth Int. Workshop on Database Programming Languages*, pages 253–269, 1997.

[DP90] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Camebridge University Press, 1990.

[FAC01] The world factbook 2001, `http://www.cia.gov/cia/publications/factbook`, 2001.

[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.

[HKPT99] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[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.

[KM95] J. Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149:129–149, 1995.

[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.

[LPL00] Stephane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and Armstrong relations. In *Proc. 7th Int. Conf. on Extending Database Technology (EDBT 2000)*, LNCS 1777, pages 350–364. Springer, 2000.

[May99] Wolfgang May. Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999.

[MR86] Heikki Mannila and Kari-Jouko Räihä. Design by example: An application of armstrong relations. *Journal of Computer and System Sciences*, 33(2):126–141, October 1986.

[MR87] Heikki Mannila and Kari-Jouko Räihä. Dependency inference. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *Proc. Int. Conf. Very Large Data Bases*, pages 155–158, Brighton, England, 1–4 September 1987. Morgan Kaufmann.

[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.

[NC01] Noel Novelli and Rosine Cicchetti. FUN: An efficient algorithm for mining functional and embedded dependencies. In *ICDT*, volume 1973 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 2001.

[Sch93] J. C. Schlimmer. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *Proc. Int. Conf. on Machine Learning*, pages 284–290, 1993.

[van01] Johan van der Heyden. Global statistics, `http://www.geohive.com/`. GeoHive, 2001.

[vNC98] Patrick R. J. van der Laag and Shan-Hwei Nienhuys-Cheng. Completeness and properness of refinement operators in inductive logic programming. *Journal of Logic Programming*, 34(3):201–225, 1998.

[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. *ACM Trans. on Database Systems*, 24(1):127–176, 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.

[WN99] J. Wijsen and Raymond T. Ng. Temporal dependencies generalized for spatial and other dimensions. In *Proc. Int. Workshop on Spatio-Temporal Database Management (STDBM'99)*, LNCS 1678, pages 189–203. Springer, 1999.

[WNC99] J. Wijsen, R.T. Ng, and T. Calders. Discovering roll-up dependencies. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 213–222, San Diego, CA, 1999.

A Proof of Theorem 1

Proof. Let G, H be genschemes of a relation scheme S. We write $G \oplus H$ for the smallest set containing (A, l) if G contains (A, g) for some level $g \preceq l$, and H contains (A, h) for some level $h \preceq l$.

Let $S^{\diamond}$ be the set of all genschemes of S ordered by $\trianglelefteq$. It suffices to show two things: First, that $(S^{\diamond}, \trianglelefteq)$ is a partially ordered set; and second, that $\inf\{G, H\}$ and $\sup\{G, H\}$ are defined for all genschemes G and H of S. From this and the obvious fact that $S^{\diamond}$ is finite, it follows that $(S^{\diamond}, \trianglelefteq)$ is a complete lattice.

We first show that $(S^{\diamond}, \trianglelefteq)$ is a partially ordered set. Proving reflexivity and transitivity is straightforward. For antisymmetry, assume G and H are genschemes of S with $G \trianglelefteq H$ and $H \trianglelefteq G$. We need to show $G = H$, i.e., $G \subseteq H$ and $H \subseteq G$. We show $G \subseteq H$; the proof of $H \subseteq G$ is symmetrical. Let $(A, l) \in G$. It suffices to show $(A, l) \in H$. Since $H \trianglelefteq G$, H contains (A, h) for some level h with $h \preceq l$. Since $G \trianglelefteq H$, G contains (A, g) for some level g with $g \preceq h$. So G contains both (A, l) and (A, g) with $g \preceq h \preceq l$. By the definition of genscheme, $g = l$, hence $h = l$. Consequently, H contains (A, l).

We next show that $\inf\{G, H\}$ and $\sup\{G, H\}$ are defined for all genschemes G, H of S. We show that $\inf\{G, H\} = \lfloor G \cup H \rfloor$. Clearly, $\lfloor G \cup H \rfloor \trianglelefteq G$ and $\lfloor G \cup H \rfloor \trianglelefteq H$. Let F be a genscheme of S with $F \trianglelefteq G$ and $F \trianglelefteq H$. We need to show $F \trianglelefteq \lfloor G \cup H \rfloor$. Let $(A, l) \in \lfloor G \cup H \rfloor$. Hence, $(A, l) \in G$ or $(A, l) \in H$. Since $F \trianglelefteq G$ and $F \trianglelefteq H$, F contains (A, g) for some level g with $g \preceq l$. Since (A, l) is an arbitrary member of $\lfloor G \cup H \rfloor$, it is correct to conclude that $F \trianglelefteq \lfloor G \cup H \rfloor$. Finally, we show that $\sup\{G, H\} = \lfloor G \oplus H \rfloor$. Clearly, $G \trianglelefteq \lfloor G \oplus H \rfloor$ and $H \trianglelefteq \lfloor G \oplus H \rfloor$. Let F be a genscheme of S with $G \trianglelefteq F$ and $H \trianglelefteq F$. We need to show $\lfloor G \oplus H \rfloor \trianglelefteq F$. Let $(A, l) \in F$. Since $G \trianglelefteq F$ and $H \trianglelefteq F$, G contains (A, g) for some level $g \preceq l$, and H contains (A, h) for some level $h \preceq l$. Hence, $G \oplus H$ contains (A, l). Hence, $\lfloor G \oplus H \rfloor$ contains (A, f) for some level $f \preceq l$. Since (A, l) is an arbitrary member of F, it is correct to conclude that $\lfloor G \oplus H \rfloor \trianglelefteq F$. This concludes the proof. □

B Proof of Theorem 2

Lemma 1 *If $h \prec l$, $(A, l) \in G$, and $G \trianglelefteq H$, then $(A, h) \notin H$.*

Proof. Assume $h \prec l$, $(A, l) \in G$, and $G \trianglelefteq H$. Suppose $(A, h) \in H$. Since $G \trianglelefteq H$, G contains (A, g) for some level $g \preceq h$. From $g \preceq h$ and $h \prec l$, it follows $g \prec l$. But then G contains both (A, g) and (A, l) with $g \prec l$, a contradiction. We conclude by contradiction that $(A, h) \notin H$. □

Lemma 2 *If $F \trianglelefteq G \trianglelefteq H$, then H contains no element of $F - G$.*

Proof. Assume $F \trianglelefteq G \trianglelefteq H$. Assume the existence of an element (A, l) such that $(A, l) \in H$, $(A, l) \in F$, but $(A, l) \notin G$. Since $G \trianglelefteq H$, G must contain (A, h) for some level $h \preceq l$. Since $(A, l) \notin G$, $h \neq l$, hence $h \prec l$. From $h \prec l$, $(A, l) \in F$, and $F \trianglelefteq G$, it follows by Lemma 1 that $(A, h) \notin G$, a contradiction. We conclude by contradiction that H contains no element of $F - G$. $\square$

Definition 10 Let G be a genscheme of a relation scheme S. Let $(A, l) \in G$. We write $G^{\lhd(A,l)}$ for the genscheme $\lfloor(G - \{(A, l)\}) \cup \{(A, l') \mid l \multimap\!\!\prec l'\}\rfloor$. $\square$

Lemma 3 *Let $(A, l) \in G$. Then $G \lhd G^{\lhd(A,l)}$.*

Proof. Since $G \neq G^{\lhd(A,l)}$ is obvious, it suffices to show $G \unlhd G^{\lhd(A,l)}$. Assume $(C, h) \in G^{\lhd(A,l)}$. It suffices to show that G contains (C, g) for some level $g \preceq h$. If $C \neq A$, then $(C, h) \in G$ follows from the definition of $G^{\lhd(A,l)}$. If $C = A$, then either $(A, h) \in G$ or $l \multimap\!\!\prec h$, and the desired result follows immediately. $\square$

Lemma 4 *Let $(A, l), (C, g) \in G$. If $(C, g) \notin G^{\lhd(A,l)}$, then $(A, l) = (C, g)$.*

Proof. Assume $(A, l), (C, g) \in G$ and $(C, g) \notin G^{\lhd(A,l)}$. $C = A$ follows immediately from the definition of $G^{\lhd(A,l)}$. Suppose $l \neq g$. Then $G^{\lhd(A,l)}$ contains (A, h) for some level $h \prec g$, such that the application of $\lfloor\cdot\rfloor$ in the computation of $G^{\lhd(A,l)}$ has the effect of removing (A, g). From $h \prec g$, $(A, g) \in G$, and $G \unlhd G^{\lhd(A,l)}$ (Lemma 3), it follows by Lemma 1 that $(A, h) \notin G^{\lhd(A,l)}$, a contradiction. We conclude by contradiction that $l = g$. This concludes the proof. $\square$

Lemma 5 *If $(A, l) \in G$ and $l \prec h$, then $G^{\lhd(A,l)}$ contains (A, g) for some level $g \preceq h$.*

Proof. Assume $(A, l) \in G$ and $l \prec h$. Then there exists some level l' such that $l \multimap\!\!\prec l' \preceq h$. If $(A, l') \in G^{\lhd(A,l)}$, then the desired result obtains vacuously. Next assume $(A, l') \notin G^{\lhd(A,l)}$. Hence, $G^{\lhd(A,l)}$ must contain some element (A, g) with $g \prec l'$, such that the $\lfloor\cdot\rfloor$ operator in the computation of $G^{\lhd(A,l)}$ has the effect of removing (A, l'). It follows that $G^{\lhd(A,l)}$ contains (A, g) with $g \preceq h$. $\square$

Lemma 6 *If $G \multimap\!\!\lhd H$ then $H = G^{\lhd(A,l)}$ for some $(A, l) \in G$.*

Proof. Assume $G \multimap\!\!\lhd H$. $G \not\subseteq H$, or else $H \unlhd G$. Let $(A, l) \in G - H$. We prove $G^{\lhd(A,l)} = H$. Since $G \multimap\!\!\lhd H$ and $G \lhd G^{\lhd(A,l)}$ (Lemma 3), it suffices to show $G^{\lhd(A,l)} \unlhd H$. Let $(C, h) \in H$. Since $G \unlhd H$, G contains a pair (C, g) for some level $g \preceq h$. It suffices to show that $G^{\lhd(A,l)}$ contains a pair (C, f) with $f \preceq h$. If $G^{\lhd(A,l)}$ contains (C, g) then the desired result follows immediately. Next assume $(C, g) \notin G^{\lhd(A,l)}$. By Lemma 4, $(C, g) = (A, l)$. Since $(A, h) \in H$, $(A, l) \notin H$, and $g = l \preceq h$, we conclude $l \prec h$. By Lemma 5, $G^{\lhd(A,l)}$ will contain some pair (A, f) with $f \preceq h$. $\square$

The proof for Theorem 2 can now be given.

Proof. *Only-if part.* Immediately by Lemma 6. *If part.* Suppose that $(A, l) \in G$. Suppose *not* $G \multimap\!\!\lhd G^{\lhd(A,l)}$. Since $G \lhd G^{\lhd(A,l)}$ (Lemma 3), there exists a generalization scheme F with $G \multimap\!\!\lhd F \lhd G^{\lhd(A,l)}$. By Lemma 6, $F = G^{\lhd(C,g)}$ for some $(C, g) \in G$. Clearly, $(A, l) \neq (C, g)$, or else $F = G^{\lhd(A,l)}$, a contradiction. Suppose $(C, g) \in G^{\lhd(A,l)}$. Since $F \lhd G^{\lhd(A,l)}$, F contains a pair (C, f) with $f \preceq g$. We have $f \neq g$, since $(C, g) \notin F$, hence $f \prec g$. By Lemma 1, $(C, f) \notin F$, a contradiction. We conclude by contradiction that $(C, g) \notin G^{\lhd(A,l)}$. By Lemma 4, $(C, g) = (A, l)$, again a contradiction. We conclude by contradiction that $G \multimap\!\!\lhd G^{\lhd(A,l)}$. $\square$

C Proof of Theorem 3

The following lemma implies that in the roll-up lattice, any path towards the top has to respect all levels of the roll-up scheme. More precisely, if $G_1 \multimap\!\!\lhd G_2 \multimap\!\!\lhd \ldots \multimap\!\!\lhd G_n \multimap\!\!\lhd \top$ are genschemes, and $(A,l) \in G_1$ and $l \prec h$, then (A,h) occurs in some G_i ($i \in [1..n]$).

Lemma 7 *If $G_1 \multimap\!\!\lhd G_2 \multimap\!\!\lhd \ldots \multimap\!\!\lhd G_n$ are genschemes, and $(A,l) \in G_1$, and $l \prec h$, then either $(A,h) \in G_i$ for some $i \in [1..n]$, or G_n contains (A,g) for some level $g \preceq h$, or both.*

Proof. Proof by induction on n. The base case $n = 1$ is trivial. For the induction step, assume $G_1 \multimap\!\!\lhd G_2 \multimap\!\!\lhd \ldots \multimap\!\!\lhd G_k$ are genschemes, $(A,l) \in G_1$, and $l \prec h$. Further assume $(A,h) \notin G_i$ for all $i \in [1..k]$. It suffices to show that G_k contains (A,g) for some level $g \preceq h$. Since $(A,h) \notin G_i$ for all $i \in [1..k-1]$ is obvious, it follows by the induction hypothesis that G_{k-1} contains (A,f) for some level f with $f \prec h$. Two cases can occur.

1. $G_k = G_{k-1}{}^{\lhd(A,f)}$. By Lemma 5, G_k contains (A,g) for some level $g \preceq h$.
2. $G_k \neq G_{k-1}{}^{\lhd(A,f)}$. By Lemma 6, $G_k = G_{k-1}{}^{\lhd(C,e)}$ for some $(C,e) \in G_{k-1}$. If $(A,f) \notin G_k$, then $(A,f) = (C,e)$ by Lemma 4, hence $G_k = G_{k-1}{}^{\lhd(A,f)}$, a contradiction. We conclude by contradiction that $(A,f) \in G_k$, where $f \preceq h$.

□

Lemma 8 *If $G \multimap\!\!\lhd H$ then $G - H$ is a singleton and—let $G - H$ be the singleton $\{(A,l)\}$:*

$$H = G^{\lhd(A,l)} \ .$$

Proof. Assume $G \multimap\!\!\lhd H$. By Lemma 6, $H = G^{\lhd(A,l)}$ for some $(A,l) \in G$. Clearly, $(A,l) \in G - H$. Suppose the existence of (C,g) with $(C,g) \in G$ but $(C,g) \notin H$. By Lemma 4, $(A,l) = (C,g)$. This concludes the proof. □

Lemma 9 *Consider the generalization schemes: $G = G_1 \multimap\!\!\lhd G_2 \multimap\!\!\lhd \ldots \multimap\!\!\lhd G_n = H$ and $G = H_1 \multimap\!\!\lhd H_2 \multimap\!\!\lhd \ldots \multimap\!\!\lhd H_m = H$. Then $\bigcup_{i=1}^{n} G_i = \bigcup_{j=1}^{m} H_j$.*

Proof. We prove $\bigcup_{i=1}^{n} G_i \subseteq \bigcup_{j=1}^{m} H_j$. The proof runs by induction on n. The base case $n = 1$ is obvious. For the induction step, assume $\bigcup_{i=1}^{k-1} G_i \subseteq \bigcup_{j=1}^{m} H_j$. Let $(A,l) \in G_k$. It suffices to show $(A,l) \in \bigcup_{j=1}^{m} H_j$. If $(A,l) \in G_i$ for some $i \in [1..k-1]$, then the desired result obtains obviously. We next assume $(A,l) \notin G_i$ for all $i \in [1..k-1]$. By Theorem 2, it is correct to assume that $G_k = G_{k-1}{}^{\lhd(A,g)}$ for some $(A,g) \in G_{k-1}$ with $g \multimap\!\!\prec l$. Since $G_{k-1} \trianglelefteq G_k \trianglelefteq H$ and $(A,g) \in G_{k-1} - G_k$, it follows by Lemma 2 that $(A,g) \notin H$. On the other hand, since $(A,g) \in G_{k-1}$, it follows by the induction hypothesis that $(A,g) \in H_j$ for some $j \in [1..m]$. Hence, we can assume the existence of some $p \in [1..m-1]$ such that $(A,g) \in H_p$ but $(A,g) \notin H_{p+1}$. Since $H_p \multimap\!\!\lhd H_{p+1}$, $H_{p+1} = H_p{}^{\lhd(A,g)}$ by Lemma 8. If $(A,l) \in H_{p+1}$, the desired result obtains vacuously. Next assume $(A,l) \notin H_{p+1}$. Then since $g \multimap\!\!\prec l$, we can assume the existence of some level f with $(A,f) \in H_p$ and $f \prec l$ so that the application of $\lfloor \cdot \rfloor$ in the computation of H_{p+1} has the effect of removing (A,l). Since

- $H_p \multimap\!\!\lhd H_{p+1} \multimap\!\!\lhd \ldots \multimap\!\!\lhd H_m = H$,

- $(A, f) \in H_p$, and
- $f \prec l$,

it follows by Lemma 5 that either $(A, l) \in H_j$ for some $j \in [p..m]$, or H contains (A, e) for some $e \preceq l$, or both. Assume some level e with $e \prec l$. Since $(A, l) \in G_k$ and $G_k \trianglelefteq H$, it follows by Lemma 1 that H does not contain (A, e). It follows $(A, l) \in H_j$ for some $j \in [p..m]$. To conclude, $\bigcup_{i=1}^{n} G_i \subseteq \bigcup_{j=1}^{m} H_j$. By symmetry, $\bigcup_{i=1}^{n} G_i = \bigcup_{j=1}^{m} H_j$. □

Theorem 6 *Consider the following chain of genschemes* $G_1 \prec\!\!\lhd G_2 \prec\!\!\lhd \ldots \prec\!\!\lhd G_n$. *Then* $|\bigcup_{i=1}^{n} G_i| = |G_n| + n - 1$.

Proof. We show that $|\bigcup_{i=k}^{n} G_i| = |G_n| + n - k$ for each $k \in [1..n]$. Proof by induction on decreasing k. The base case $k = n$ is trivial. For the induction step, $|\bigcup_{i=k-1}^{n} G_i| = |G_{k-1} \cup \bigcup_{i=k}^{n} G_i|$. By Lemma 8, $G_{k-1} - G_k$ is a singleton (say $\{(A, l)\}$), and $G_k = {G_{k-1}}^{\lhd(A,l)}$. Moreover, by Lemma 2, $(A, l) \notin G_j$ for all $j \geq k$. Hence, $|\bigcup_{i=k-1}^{n} G_i| = 1 + |\bigcup_{i=k}^{n} G_i|$. By the induction hypothesis, $|\bigcup_{i=k}^{n} G_i| = |G_n| + n - k$. Hence, $|\bigcup_{i=k-1}^{n} G_i| = |G_n| + n - (k - 1)$. □

Corollary 1 *Consider the generalization schemes:* $G = G_1 \prec\!\!\lhd G_2 \prec\!\!\lhd \ldots \prec\!\!\lhd G_n = H$ *and* $G = H_1 \prec\!\!\lhd H_2 \prec\!\!\lhd \ldots \prec\!\!\lhd H_m = H$. *Then* $m = n$.

Proof. By Lemma 9, $\bigcup_{i=1}^{n} G_i = \bigcup_{j=1}^{m} H_j$. Hence $|\bigcup_{i=1}^{n} G_i| = |\bigcup_{j=1}^{m} H_j|$. By Theorem 6, $|H| + n - 1 = |H| + m - 1$. Hence, $m = n$. □

Finally, we obtain a proof for Theorem 3.
Proof. Consider the family of sets defined as follows:

- $P_1 = \{\top\}$
- for each $i \geq 2$, P_i is the smallest set containing $G \in S^{\diamond}$ if there exist i genschemes $G_1, \ldots, G_i$ such that $G = G_1 \prec\!\!\lhd G_2 \prec\!\!\lhd \ldots \prec\!\!\lhd G_{i-1} \prec\!\!\lhd G_i = \top$.

Since $G \trianglelefteq \top$ for each $G \in S^{\diamond}$, every genscheme will be in at least some P_i. Moreover, by Corollary 1, no $G \in S^{\diamond}$ can be in P_i and P_j unless $i = j$. So the non-empty sets $P_1, \ldots, P_n$ are a partition of $S^{\diamond}$. To see that this partition is a stratification, assume $F \prec\!\!\lhd G$ and $G \in P_i$. Hence, there exist i genschemes $G_1, \ldots, G_i$ such that $G = G_1 \prec\!\!\lhd G_2 \prec\!\!\lhd \ldots \prec\!\!\lhd G_{i-1} \prec\!\!\lhd G_i = \top$. Since $F \prec\!\!\lhd G_1 \prec\!\!\lhd G_2 \prec\!\!\lhd \ldots \prec\!\!\lhd G_{i-1} \prec\!\!\lhd G_i = \top$, it follows $F \in P_{i+1}$. □