Frequent query discovery: a unifying ILP approach to association rule mining

Luc Dehaspe and Hannu Toivonen

Report CW 258, March 2, 1998

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Frequent query discovery: a unifying ILP approach to association rule mining

Luc Dehaspe[*] *and Hannu Toivonen*[†]

Report CW 258, March 2, 1998

Department of Computer Science, K.U.Leuven

Abstract

Discovery of frequent patterns has been studied in a variety of data mining (DM) settings. In its simplest form, known from association rule mining, the task is to find all frequent itemsets, i.e., to list all combinations of items that are found in a sufficient number of examples. A similar task in spirit, but at the opposite end of the complexity scale, is the Inductive Logic Programming (ILP) approach where the goal is to discover queries in first order logic that succeed with respect to a sufficient number of examples.

We discuss the relationship of ILP to frequent pattern discovery. On one hand, our goal is to relate data mining problems to ILP. On another hand, we want to demonstrate how ILP can be used to solve both existing and new data mining problems.

The fundamental task of association rule and frequent set discovery has been extended in various directions, allowing more useful patterns to be discovered. From an ILP viewpoint, however, it can be argued that these settings are all well-controlled subtasks of the full ILP counterpart of the problem. We try to restore the blurred picture by describing the existing approaches using a unified database representation. With the representation, we relate also the DM settings to each other and propose some interesting new areas to be explored. We analyse some aspects of the gradual change in the trade-off between expressivity and efficiency, as one moves from the frequent set problem towards ILP.

Keywords : frequent pattern discovery, inductive logic programming, data mining.

[*]Department of Computer Science, Katholieke Universiteit Leuven
[†]Department of Computer Science, University of Helsinki

1 Introduction

Discovery of frequent patterns has been studied in a variety of data mining (DM) settings. In its simplest form, known from association rule mining [2], the task is to find all frequent itemsets, i.e., to list all combinations of items that are found in a sufficient number of examples. A prototypical application example is in market basket analysis: find out which products tend to be sold together.

A similar task in spirit, but at the opposite end of the complexity scale, is the Inductive Logic Programming (ILP) approach [12, 5, 34], where the goal is to discover queries in first order logic that succeed with respect to a sufficient number of examples [20]. In the application area of market basket analysis, such queries could express relationships between the properties of customers, their market baskets as a whole, and the individual products bought — not just connections between item types bought.

In this paper, we discuss the relationship of this ILP approach to frequent pattern discovery. On one hand, our goal is to relate data mining problems to ILP. On another hand, we want to demonstrate how ILP can be used to solve both existing and new data mining problems.

The fundamental task of association rule and frequent set discovery has been extended in various directions, allowing more useful patterns to be discovered. Alternative problem settings include the use of item type hierarchies [26, 28, 53], the discovery of episodes in event sequences [37, 39], and the search of sequential patterns from series of transactions [4, 55]. These different approaches have been mostly described as parallel extensions to the elementary task, each with their own notational conventions. General considerations of this problem area are few [25, 38], and they have been concerned with concepts, algorithms, and complexity rather than with the issues of expressive power or the exact relationship between different settings.

From an ILP viewpoint it can be argued that these settings are all well-controlled subtasks of the full ILP counterpart of the problem. The use of, e.g., DATALOG allows a clear and uniform formulation of the problems, and a suitable presentation shows connections and differences between the various DM tasks.

We start in Section 2 by describing the general data mining task of discovering frequent first order clauses, and then define the particular DM problems as specializations of that task. Using the common representation, we relate the DM settings to each other and propose some interesting new areas to be explored. In Section 3 we consider algorithms for solving the problems, and analyse some aspects of the gradual change in the trade-off between expressivity and efficiency, as one moves from the frequent itemset problem towards ILP. Some of the new instances of frequent pattern discovery are addressed in Section 4. In Section 5 we demonstrate the potential role of ILP as a benchmark technique for specialized data mining algorithms. We present experimental results in the area of telecommunication alarm sequence analysis. We show how ILP methods can be used to solve problems attacked before, and also consider useful extensions. Finally, in Sections 6 and 7, we touch upon related and future work, and conclude.

2 Description of the data mining task

We use DATALOG [58] to represent data and patterns. There is a straightforward and well-defined correspondence between DATALOG and both relational databases and first order clausal logic. The use of DATALOG allows us to describe a number of data mining tasks in the area of frequent pattern discovery in a clear and uniform manner.

Relational algebra, the theoretical framework of relational databases, has the same expressive power as DATALOG without recursion. For instance, the recursive concept *ancestor* can be defined in DATALOG but not in relational algebra. DATALOG, in turn, is a subset of clausal logic (and PROLOG) that is restricted to function-free definite clauses.

We briefly review the DATALOG concepts used in this paper. These concepts are then used to describe the data mining task of discovering frequent patterns or, in DATALOG terminology, frequent queries. We then show how some existing frequent pattern discovery settings can be derived from this task. To conclude this section on representational issues, we situate the discussed settings for frequent pattern discovery in a multi-dimensional space.

2.1 Background: DATALOG concepts

In DATALOG a *term* is defined as a constant symbol or a variable. To distinguish between them, we will write variables with an initial upper-case letter. An *atom* is an m-ary predicate symbol followed by a bracketed m-tuple of terms. A *definite clause* is a universally quantified formula of the form $B \leftarrow A_1, \ldots, A_n$ $(n \geq 0)$, where B and the A_i are atoms. This formula can be read as "B if A_1 and ... and A_n". If $n = 0$ a definite clause is also called a *fact*. *Ground* clauses are clauses that contain only constants as terms, no variables.

A deductive DATALOG *database* is a set of definite clauses. Often a distinction is made between the extensional part of a DATALOG database, with ground clauses, and the remaining intensional part. The presence of intensional non-ground definitions distinguishes deductive DATALOG databases from relational databases.

A formula $\leftarrow A_1, \ldots, A_n$ without a conclusion part is called a *denial*. Such a formula can also be viewed as a *query ?-* $A_1, \ldots, A_n$: (the resolution based derivation of) the answer to a given query with variables $(X_1, \ldots, X_m)$ binds these variables to terms $(a_1, \ldots, a_m)$, such that the clause is true if each X_i is replaced by a_i. This binding is denoted by $(X_1/a_1, \ldots, X_m/a_m)$. Due to the nondeterministic nature of the computation of answers, a single query Q

may result in many bindings. We will refer to the set of bindings obtained by submitting query Q to a DATALOG database D as $answerset(Q, D)$.

As already pointed out, if recursion is not allowed, these concepts correspond directly to relational database terminology. Predicates map to relations, facts to tuples of a relation, and a query such as

?- transaction(Transaction_id, Itemtype), is_a(Itemtype, beverage)

can be written in SQL as

```
SELECT  TRANSACTION.transaction_id, TRANSACTION.itemtype
FROM    TRANSACTION, IS_A
WHERE   TRANSACTION.itemtype=IS_A.itemtype
AND     IS_A.ancestor='beverage'
```

An overview of strategies to make the relationship between DATALOG and relational databases operational can be found in [58].

2.2 Frequent query discovery

Mannila and Toivonen [38] describe the following generic problem of finding all potentially interesting sentences. Given are a database $\mathbf{r}$, a class $\mathcal{L}$ of sentences (patterns), and a selection predicate q which is used for evaluating whether a sentence $Q \in \mathcal{L}$ defines a potentially interesting pattern in $\mathbf{r}$. The task is to find the theory of $\mathbf{r}$ with respect to $\mathcal{L}$ and q, i.e., the set $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q) = \{Q \in \mathcal{L} \mid q(\mathbf{r}, Q) \text{ is true}\}$. In terms of their framework, we consider the following data mining task.

Definition 1 (frequent query discovery) *Assume*

- $\mathbf{r}$ *is a* DATALOG *database,*
- $\mathcal{L}$ *is a set of* DATALOG *queries, and*
- $q(\mathbf{r}, Q)$ *is true if and only if the frequency of query* $Q \in \mathcal{L}$ *with respect to* $\mathbf{r}$ *is at least equal to the frequency threshold specified by the user.*

The task is to find the set $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ *of frequent queries.*

In this general form, the task would be to discover arbitrary queries that are frequent. In this section we consider the relationship of $\mathcal{L}$, as defined above, to the languages used in well-known data mining tasks. We next define what frequency exactly means in this setting.

Definition 2 (frequency of query Q with respect to database r)
Assume

- $\mathbf{r}$ *is a* DATALOG *database and* Q *is a query,*
- keypred *is a predicate name not used in* Q *or* $\mathbf{r}$*, and*
- KeyVars *is a tuple of user specified* key variables *of* Q*.*

Then the (absolute) frequency of query Q *w.r.t.* $\mathbf{r}$ *is*

$$|answerset(\text{?- keypred(KeyVars)}, \mathbf{r} \cup \{\text{keypred(KeyVars)} \leftarrow Q\})|,$$

i.e., the number of bindings of the key variables with which the query Q *is true.*

Once more, we establish the link with relational database terminology. In SQL syntax the frequency of Q can be obtained with the following query, inspired by [35, 7]:

```
SELECT   count(distinct *)
FROM     SELECT   fields that correspond to KeyVars
         FROM     relations in Q
         WHERE    conditions expressed in Q
```

We next show how restrictions on the DATALOG database $\mathbf{r}$ and the language $\mathcal{L}$ of DATALOG queries lead to some of the existing frequent pattern discovery settings.

2.3 Item sets

In the context of association rule mining [2], the task is to list all frequent combinations of items. For instance, in market basket analysis one wants to find out which products tend to be sold in the same transaction. Several equivalent database representations are possible for such transaction data. They can all be mapped to the case where a single transaction is represented as a set of facts *tr(tid,itemtype$_j$)*$\leftarrow$, where *tid* is the identifier of the transaction and *itemtype$_j$* is the name of a product sold in that transaction. Given this form, we can formulate the setting as follows.

Definition 3 (itemset discovery) *Discovery of frequent itemsets is a special case of frequent query discovery where*

- *the database* $\mathbf{r}$ *consists of one or more facts* tr(tid,itemtype$_j$)$\leftarrow$ *per transaction* tid*,*
- *the queries in* $\mathcal{L}$ *only contain atoms of the form* tr(Tid,itemtype$_j$)*, and*
- KeyVars *equals the transaction identifier Tid.*

For instance, given a DATALOG database

$$\{tr(1, beer) \leftarrow; tr(1, chips) \leftarrow; tr(2, beer) \leftarrow; tr(2, pampers) \leftarrow; tr(2, chips) \leftarrow\},$$

the frequency of ?- *tr(Tid,pampers)* is one, and the frequency of ?- *tr(Tid,beer), tr(Tid,chips)* is two.

2.4 Item hierarchies

In market basket analysis, it is useful to consider hierarchies of item types and to analyze basket contents on various concept levels. Whether a customer bought *budweiser* or *heineken* does not necessarily matter, and in such cases the higher level concept of *beer* is more useful. On the other hand, it might turn out that customers buying *hoegaarden* beer are original and show distinctive shopping patterns. The problem of discovering generalized itemsets on multiple levels of an item type hierarchy has been considered, e.g., in [26, 28, 53]. In our DATALOG representation, item hierarchies can be specified with facts *is_a(itemtype$_j$, ancestor$_k$)$\leftarrow$* which hold for all transactions.

Definition 4 (itemset discovery with item hierarchies) *Discovery of frequent itemset with item hierarchies is a special case of frequent query discovery, where*

- *database* r *contains one or more facts* tr(tid, itemtype$_j$)$\leftarrow$ *per transaction* tid*, as before, but also facts of the form* is_a(itemtype$_j$, ancestor$_k$)$\leftarrow$
- *the queries in* $\mathcal{L}$ *contain atoms of the form* tr(Tid, itemtype$_j$)*, as before, but also atom pairs* (tr(Tid, I), is_a(I, ancestor$_k$))*,*
- KeyVars *equals the transaction identifier Tid.*

In the original formulations of the problem there is the following extra restriction: all itemtype variables *I* in a query are different, i.e., redundancies concerning an item and its own ancestors are pruned. For simplicity, we ignore this.

2.5 Sequential patterns

Consider the case in market basket analysis where the sequence of transactions of each individual customer can be tracked. It is interesting to look for frequent patterns over such sequences of transactions. Sequential patterns [4, 55] are a generalization of frequent itemsets to frequent sequences of itemsets.

To describe the data, we use facts *customer(cust, tid)*$\leftarrow$ that associate each transaction *tid* to the customer *cust* that made it. The order of transactions is represented by a set of clausal definitions for predicate *order/2*.

Definition 5 (sequential pattern discovery) *Discovery of frequent sequential patterns is a special case of frequent query discovery where*

- *database* r *contains one or more facts* tr(tid, itemtype)$\leftarrow$ *per transaction* tid*, as before, but also one or more facts* customer(cust, tid)$\leftarrow$ *per customer* cust*, and a set of clauses for predicate* order/2*,*

- *the queries in $\mathcal{L}$ contain series of atoms of the forms*
  customer(Cust,Tid$_1$),tr(Tid$_1$,itemtype$_{11}$),tr(Tid$_1$,itemtype$_{12}$), ...
  customer(Cust,Tid$_2$),tr(Tid$_2$,itemtype$_{21}$),tr(Tid$_2$,itemtype$_{22}$), ...
  ...
  and customer(Cust,Tid1), customer(Cust,Tid2), order(Tid1,Tid2),
- KeyVars *equals the customer identifier* Cust.

Again, the original formulation is in some ways more restricted than this definition. Each transaction *Tid* in a query is required to have at least one item (i.e., to occur in the first atom type mentioned above) and to have a unique position with respect to the other transactions (e.g., to occur in the latter atom type with all other transactions). On the other hand, the original settings present some extensions not covered here.

2.6 Episodes

Consider now a case where the input consists of a long sequence of items such that remote items in the sequence are not related. Such sequences could arise, for instance, in life-long customer relationships of pharmacies: drugs purchased within weeks may be related, but drugs purchased years or decades apart are related only in special cases. A better motivated problem is perhaps in the analysis of alarms from a telecommunication network [31]. There the task is discover combinations of related alarms that are received from a number of devices in a large network.

Episodes [39] are such patterns. Given a long sequence of items and a window width, one looks at the sequence by sliding a window of the given width, and analyzes the combinations of items visible in the window. We discuss two variants of episodes. The first variant, parallel episodes, is a simple adaptation of frequent itemsets to such sequential cases with windowing. The second variant, general episodes is a pattern type with a number of ILP like elements and with a substantially increased expressive power.

2.6.1 Parallel episodes

For the definition of the episode discovery task, we use facts *window(wid, tid)*$\leftarrow$ as a means to represent that window *wid* contains transaction *tid*. In the case of episodes, each transaction consists of exactly one item. In the alarm domain, transactions or items have a number of properties in addition to the item type, such as the urgency or the sender of the alarm. These are all given in facts *tr(tid, itemtype*$_j$*,* a_2*, ...,* a_n*)*$\leftarrow$.

Definition 6 (discovery of parallel episodes) *Discovery of frequent parallel episodes is a special case of frequent query discovery where*

- *database* **r** *contains per time window* wid *(1) both one or more facts* window(wid, tid)←, *and, (2) per transaction* tid *in the window, one fact* tr(tid, itemtype$_j$, a$_2$, …, a$_n$)←,
- *the queries in $\mathcal{L}$ consist of one or more atom series of the form* window(Wid,Tid$_1$), tr(Tid$_1$,V$_1$,…,V$_n$), V$_i$=constant, window(Wid,Tid$_2$), tr(Tid$_2$,V$_1$,…,V$_n$), V$_j$=constant,
- KeyVars *equals the window identifier Wid.*

Parallel episodes could be defined as itemsets by a simple transformation of the representation. However, the present way of describing episodes is more appropriate for further analysis of the problem field.

2.6.2 General episodes

Our final example of data mining tasks in the area of frequent patterns are general episodes. The formulation here imitates the ones given in [37, 39]. Unlike the examples discussed above, this one has not been implemented in a full scale before.

The setting for general episodes we consider here has the following differences from parallel episodes. Properties of item types and properties of transactions/items are represented as a set of facts *unary$_i$(a$_j$)←*; and finally, binary relations on transactions/items as a third set of facts *binary$_i$(a$_j$, a$_k$)←*. Clauses *order(tid1, tid2)←* for representing a partial ordering on transactions are a special case of binary relations and thus unnecessary for the specification of the problem: they can be defined, for instance in terms of a binary relation on the time attribute a_t associated with transactions.

Definition 7 (discovery of general episodes) *Discovery of frequent general episodes is a special case of frequent query discovery where*

- *database* **r** *contains per time window* wid, *as before, (1) one or more facts* window(wid, tid)← *and (2) per transaction* tid *in the window, one fact* tr(tid, itemtype$_j$, a$_2$, …, a$_n$)←, *and*
  sets of facts unary$_i$(a$_j$)←, *and* binary$_j$(a$_j$, a$_k$)←,
- *the queries in $\mathcal{L}$ contain one or more atom series of the form*
  window(Wid, Tid), tr(Tid, I, …, V$_n$), unary$_i$(I), *and*
  window(Wid, Tid), tr(Tid, V$_1$, …, V$_n$), unary$_i$(V$_j$), *and*
  window(Wid, Tid1), tr(Tid1, V$_{11}$, …, V$_{1n}$),
  window(Wid, Tid2), tr(Tid2, V$_{21}$, …, V$_{2n}$), …
  binary$_i$(V$_j$, V$_k$)
- KeyVars *equals the window identifier Wid.*

	IS	IH	SP	PE	GE	DQ
Many items per transaction	+	+	+			+
Item type properties		+	+	+	+	+
Many (ordered) transactions per example			+	+	+	+
Item or transaction attributes				+	+	+
Binary item properties					+	+

Table 1: Dimensions of frequent pattern types. Legend: IS = itemsets, IH = itemsets with item hierarchies, SP = sequential patterns, PE = parallel episodes, GE = general episodes, DQ = (full) DATALOG queries.

2.7 Dimensions of the pattern discovery task

As a summary and a conclusion of the task review above, consider Table 1. The table lists five of the properties where the tasks differ; these properties are directly reflected by the existence of different types of atoms in the language $\mathcal{L}$. A cell contains a plus if the pattern type can deal or can easily be extended to deal with the given feature. Note that the table is coarse: "item type properties" means a concept hierarchy for most of the cases, and only some can handle other properties associated with item types.

According to Table 1, the most obvious gaps to fill are to either extend sequential patterns to include item and transaction attributes and binary properties, or to extend episodes to the case where a transaction can contain a number of items. Recall that general episodes have not been implemented in full, but the column GE rather lists a proposed setting for data mining. There exists a number of interesting gaps also within the filled rows.

In the following section we look at the problems and their algorithmic solutions in more detail, and in the next section we discuss some of the gaps invisible in Table 1.

3 Frequent pattern discovery algorithms

Design of algorithms for frequent pattern discovery has, indeed, turned out to be a popular topic in data mining (for a sample of algorithms, see [2, 3, 36, 51, 57]). Practically all algorithms are on some level based on the same idea of levelwise search, known from the APRIORI algorithm [3]. We first review the generic levelwise search method and its central properties, and recall how this method fits in the two-phased discovery of frequent and confident rules. Next, we introduce the algorithm WARMR [20] for finding frequent queries. Following the structure of the previous section we then look at algorithms that solve restricted variants.

Each time we show how WARMR can be tuned to simulate these algorithms and explain how and why the specialised algorithms can do a lot better than WARMR in terms of efficiency.

3.1 Background: the common elements

3.1.1 The levelwise algorithm

The levelwise algorithm [38] is based on a breadth-first search in the lattice spanned by a specialization relation $\preceq$ between patterns, cf. [40], where $p1 \preceq p2$ denotes pattern "$p1$ is more general than pattern $p2$", or "$p2$ is more specific than pattern $p1$".

The method looks at a level of the lattice at a time, starting from the most general patterns. The method iterates between candidate generation and candidate evaluation phases: in *candidate generation*, the lattice structure is used for pruning non-frequent patterns from the next level; in the *candidate evaluation* phase, frequencies of candidates are computed with respect to the database. Pruning is based on monotonicity of $\preceq$ with respect to frequency: if a pattern is not frequent then none of its specialisations are frequent. So while generating candidates for the next level, all the patterns that are specialisations of infrequent patterns can be pruned. For instance, in the APRIORI algorithm for frequent itemsets, candidates are generated such that all their subsets (i.e., generalizations) are frequent.

The levelwise approach has two crucial useful properties [38]:

- The database is scanned at most $k+1$ times, where k is the maximum level (size) of a frequent pattern. All candidates of a level are tested in single database pass. This is an important factor when mining large databases.
- The time complexity is in practice linear in the size of the result, and the number of examples.

3.1.2 Two-phased discovery of frequent and confident rules

Frequent patterns are commonly not considered useful for presentation to the user as such. Their popularity is mainly based on the fact that they can be efficiently post-processed into rules that exceed given confidence and frequency threshold values. The best known example of this two-phased strategy is the discovery of association rules [2], and closely related patterns include episodes [39] and sequential patterns [4]. For all these patterns, the threshold values offer a natural way of pruning weak and rare rules.

In terms of the DATALOG concepts introduced in Section 2, an *association rule R* is an expression of the form $A_1, \ldots, A_k \Rightarrow A_{k+1}, \ldots, A_n$, where A_i are atoms. This formula should be read as "if query ?- $A_1,\ldots,A_k$ succeeds then query ?- $A_1,\ldots,A_n$ succeeds also". The *confidence* of association rule R can be computed as the ratio of the frequencies of queries ?- $A_1,\ldots,A_n$ and ?- $A_1,\ldots,A_k$. The *frequency* (or *support*) of association rule R is the frequency of query ?- $A_1,\ldots,A_n$.

As observed in [2], confident and frequent rules can be found effectively in two steps. In the first step one determines the set of all frequent queries ?- $A_1,\ldots,A_n$, and in the second produces rules $A_1, \ldots, A_k \Rightarrow A_{k+1}, \ldots, A_n$ whose confidence exceeds the given threshold. If all frequent queries and their frequencies are known as a result of the first step, then this easy second step is guaranteed to output all frequent and confident rules.

3.2 Query discovery with WARMR

3.2.1 Specialisation relation

The subset specialization relation used in most frequent pattern discovery settings is not appropriate for structuring a space of DATALOG queries. For instance, not every atom of query ?- *parent(X,Y)* occurs as such in query ?- *parent(tom,M),female(M)* while we still would like to consider the former a generalization of the latter. Therefore, we need a strictly stronger variant of the subset relation coined θ-subsumption by Plotkin [48]. $Query1$ θ-subsumes a $Query2$ if and only if there exists a (possibly empty) binding of the variables of $Query1$, such that every atom of the resulting query occurs in $Query2$. In the example, the binding $(X/tom, Y/M)$ has the desired effect.

In theory, testing θ-subsumption is NP-complete, but in some practical cases, as discussed in [29], θ-subsumption can be tested efficiently.

3.2.2 Language bias

With association rules the definition of language $\mathcal{L}$ is straightforward: $\mathcal{L}$ is simply 2^I, where I is the set of items. Srikant, Vu, and Agrawal [56] describe a technique to impose (and exploit) user-defined constraints on combinations of items, but otherwise the definition of $\mathcal{L}$ has received little attention in the frequent pattern discovery literature. In ILP on the other hand this issue has been studied extensively in the subfield of declarative language bias. This is motivated by huge, often infinite, search spaces, that require a tight specification of interesting patterns. Several formalisms have been proposed for adding language bias information in a declarative manner to the search process (for an overview, see [1, 45]). The WARMR algorithm [20] for finding frequent queries accepts specifications of the form $rmode(n : (A_1, \ldots, A_n))$, where the A_i are atoms[1]. Typically, $n = 1$ and the specification will be of the form $rmode(n : atom)$. This

[1]Alternatively, WARMR's language bias can be specified in DLAB format [19] as in CLAUDIEN [16] and ICL [18].

format, originally proposed for PROGOL [42] and later adapted to TILDE [8], indicates which atoms can be added to a query, the maximal number of times the atom can be added ($n > 0$), and the modes and types of the variables in it. A variable V in input mode, denoted with $+V$, has to occur somewhere to the left in the query, whereas a variable in output mode, denoted with plain V, should not occur to the left. Typing of variables can be used to constrain the occurrence of input variables, such that for instance atom $beer(X)$ will not be added to ?- *customer(X),buys(X,Y)* but atom $beer(Y)$ will.

3.2.3 Candidate generation

To generate candidates, WARMR employs a classical specialisation operator under θ-subsumption [48, 44]. A specialisation operator ρ maps queries $\in \mathcal{L}$ onto sets of queries $\in 2^{\mathcal{L}}$, such that for any $Query1$ and $\forall\, Query2 \in \rho(Query1)$, $Query1$ θ-subsumes $Query2$. The operator used in WARMR essentially adds conjunctions to the query as allowed by rmodes and type specifications.

Mode declarations on variables, may cause an atom to be added for the first time only deep down the lattice. This, and the fact that conjunctions of several atoms can be added in a single refinement step, complicates pruning significantly. We can no longer require that all subsets of a candidate are frequent as some of the subsets might simply not be in $\mathcal{L}$. Instead, WARMR requires candidates not to θ-subsume any infrequent query.

As an (expensive) option we can also require that candidates are mutually inequivalent under θ-subsumption, or that candidates do not θ-subsume any of a set of user defined uninteresting queries (e.g. previous search results).

3.2.4 Candidate evaluation

In the candidate evaluation phase the frequencies of a set of queries are computed in a single database pass. Therefore, Definition 2 of frequency of a single query is not directly applicable, as it would require one pass per candidate. The WARMR algorithm rather considers one tuple K of key values at a time and for each candidate Q_i runs the query ?- *keypred(K)* in database $\mathbf{r}^K \cup \{keypred(K) \leftarrow Q_i\}$, where $\mathbf{r}^K \subseteq \mathbf{r}$ only contains clauses that are "linked" to key K. If query Q_i succeeds, an associated counter q_i is incremented. One strategy to make the selection of $\mathbf{r}^K$ operational, is to make sure key K is explicitly added to each clause in the database, such that the relation *same_key* defines a partition on $\mathbf{r}$.

For most practical cases, $\mathbf{r}^K$ is very small compared to $\mathbf{r}$, and can be loaded in main memory even if $\mathbf{r}$ cannot. This has the crucial advantage that evaluation of candidates Q_i can be done (relatively) efficiently.

The composition and the loading of $\mathbf{r}^K$ can be optimized in two steps. First, if a fixed portion $\mathbf{r}^B$ reoccurs as a subset of many $\mathbf{r}^K$'s, we can load the common $\mathbf{r}^B$ once, and iteratively load only the specific $\mathbf{r}^K \setminus \mathbf{r}^B$. In ILP jargon, $\mathbf{r}^B$

corresponds to background knowledge. For instance, in market basket analysis, background knowledge $\mathbf{r}^B$ might consist of: (1) ground facts about commodities, suppliers, and the supermarket's floor plan, and (2) clausal rules that capture general marketing principles, economic laws and so forth.

Second, in cases where the repeated composition of $\mathbf{r}^K$ is still too costly, e.g. if many clauses have to selected from many different predicates, a preprocessing step can be considered where all the $\mathbf{r}^K$'s are composed once and written to a (set of) flat file(s), see [9] for an experimental evaluation.

In addition, the evaluation of candidates itself can be optimized by organizing and rewriting the candidates in such a way that backtracking is minimized. For instance, when the atom $tr(Tid, beer)$ in query ?- *tr(Tid,I), is_a(I,toy), tr(Tid,beer)* fails, there is no point in looking for alternative bindings for I. In PROLOG notation, the *cut* operator should be inserted to suppress backtracking: ?- *tr(Tid,I), is_a(I,toy), !, tr(Tid,beer).*

In general, all work in ILP faces the theoretical result that evaluation of a query is an NP complete problem. However, queries with up to k atoms, where each atom contains with at most j terms, can be evaluated in polynomial-time with respect to a relational database, cf. [17].

We now look at the different settings for frequent pattern discovery from three angles. In the language bias paragraph we show how WARMR can be tuned to simulate the restricted setting. In the candidate generation and evaluation paragraphs we review the basic ideas and principles underlying the much faster algorithms that have been proposed within the restricted settings.

3.3 Item set discovery with APRIORI

Language bias. With language bias specifications

$rmode(1 : rmode(1 : tr\_id(Tid)),$
$rmode(1 : rmode(1 : tr(+Tid, itemtype_1)), rmode(1 : tr(+Tid, itemtype_2)), \ldots$

WARMR simulates the APRIORI algorithm [3] for finding frequent itemsets. Remember the +sign indicates the *Tid*'s in predicate $tr/2$ should be input variables, i.e. occur before in the query.

Unlike WARMR, APRIORI exploits the fact that queries that only contain atoms $tr(Tid, itemtype)$ can be mapped to sets of itemtypes, and that for itemsets, θ-subsumption is equivalent to the subset relation.

Candidate generation for frequent itemsets is efficient: it only involves subset search and testing, and the time used can be neglected in practice.

Candidate evaluation of itemsets can also be implemented efficiently. There is no need for backtracking at all, which allows an extreme form of query reorganisation, cf. the *hash-trees* described in [3]. The composition and loading step is typically optimized by preprocessing $\mathbf{r}$ such that every transaction $\mathbf{r}^K$ corresponds to one line in a flat file.

3.4 Item hierarchies

Language bias. To simulate item hierarchies with WARMR the language bias has to contain specifications of the form

$rmode(1 : rmode(1 : tr\_id(Tid)),$
$rmode(1 : rmode(1 : tr(+Tid, itemtype_1)), rmode(1 : tr(+Tid, itemtype_2)), \ldots$
$rmode(\infty : rmode(1 : tr(+Tid, I)),$
$rmode(\infty : rmode(1 : is\_a(+I, ancestor_1)),$
$rmode(\infty : rmode(1 : is\_a(+I, ancestor_2)), \ldots$

The item hierarchy can be encoded with atom constraints. As before, this setting can be expressed in terms of itemsets, and efficient algorithms rely on the subset relation rather than θ-subsumption.

Candidate generation. An item hierarchy imposes extra structure on the search space of itemsets. Frequency of items is monotone in the item hierarchy: a more general item is at least as frequent as a more specific one. Two basic techniques have been considered for dealing with item hierarchies. In the straightforward bottom-up approach [28, 53], candidate generation is the same as with itemsets, but counts are propagated up in the hierarchy. In the top-down approach [26], candidate generation takes the new specialization relation into account: more specific items are only added if a general item is already there.

An interesting point here is that although the top-down approach is better capable of limiting the search space, it may be less efficient in practice. The bottom-up approach probably is faster, since the extra work (if compared to the basic setting) can probably be performed by consuming only CPU and main memory. The top-down approach, in turn, may consider a smaller total number of candidate sets, but it more easily leads to a larger number of database passes.

Candidate evaluation is the same as with itemsets, except that, in the preprocessing step, every transaction $\mathbf{r}^K$ is computed as the union of the item types and their ancestor item types.

3.5 Sequential pattern discovery

Language bias. With basically the following language bias, WARMR discovers sequential patterns:

$rmode(1 : customer\_id(Cust)), rmode(\infty : (customer(+Cust, Tid)),$
$rmode(\infty : rmode(1 : tr(+Tid, itemtype_1)), rmode(\infty : tr(+Tid, itemtype_2)), \ldots$
$rmode(\infty : order(+Tid1, +Tid2)))$

Candidate generation. Although the patterns are more general than simple itemsets, and the subset relation is not appropriate for structuring the space of sequential patterns, a similar and efficiently computable specialization relation still exists [53].

Candidate evaluation in the GSP algorithm for mining sequential patterns [55] adapts the hash-tree structure of [3] to efficiently reduce the number of candidates that have to be checked in a sequence of itemsets. However, in the check-phase itself, backtracking over transactions in the sequence cannot be avoided, cf. the "backward phase" in GSP.

In the loading phase, one sequence of transactions $\mathbf{r}^K$ is read at a time.

3.6 Episode discovery

As argued in Section 2.6.1, parallel episodes can be transformed efficiently to simple itemsets in a preprocessing step. Therefore the observations made in Section 3.3 also hold here. As described in [39], additional efficiency is obtained via an incremental candidate evaluation technique. This technique is based on the observation that subsequent windows (or transactions, in itemset terminology) are similar to each other. We now consider the case of general episodes.
Language bias. To have WARMR discover general episodes as defined in Definition 7, the following rmode specifications have to be provided (recall a + sign denotes an input variable):

$rmode(1 : window\_id(Wid)),$
$rmode(\infty : window(+Wid, Tid)), rmode(\infty : unary_i(+V_j)),$
$rmode(\infty : tr(+Tid, I, V_1, \ldots, V_n)), rmode(\infty : binary_i(+V_j, +W_j))$

Candidate generation. The task of discovering episodes can for a large part be transformed to finding frequent sets, plus taking the order into account. The specialization relation between totally or trivially ordered patterns is easy to compute, and almost exactly same candidate generation methods can be used as for frequent sets [39, Algorithm 3]. For general episodes however the task is more difficult.

Candidate evaluation. In the candidate testing for episodes, advantage can be taken from the overlapping contents of successive window positions. Additionally, the queue structure of the window contents can also be utilized. The idea is to store full and partial bindings of variables so that minimal updates are necessary

	IS	IH	SP	PE	GE	WARMR
Incremental candidate evaluation				+	+	
Subset relation between item types only	+			+		
All backtracking suppressed	+	+		+		
Bindings can be stored	+	+	+	+	+	
Levelwise search	+	+	+	+	+	+

Table 2: Dimensions of pattern discovery algorithms. Legend: IS = itemsets, IH = itemsets with item hierarchies, SP = sequential patterns, PE = parallel episodes, GE = general episodes.

when the window slides. For the simple cases of one item per transaction and no attributes there are very efficient special solutions [11, 39], where an explicit representation of bindings is not necessary. The methods can be extended for the binary predicates, but the growth in the number of different atoms probably means that (1) less efficient indexing techniques have to be used and that (2) there is less shared information between candidate patterns to take advantage of. For relations with multiple item variables or shared variables, partial binding combinations may need to be stored — and this can require too much space and time to be useful.

Loading $\mathbf{r}^K$ is done incrementally as the window slides: the transactions leaving the window to the left are retracted from $\mathbf{r}^K$, and those entering the window to the right are added to $\mathbf{r}^K$.

3.7 Dimensions of the pattern discovery algorithms

We conclude, as we did with the section on task descriptions, with a summary in Table 2 of dimensions that characterize and relate the different pattern discovery algorithms. A plus in a cell here denotes the specialised algorithm exploits or uses the feature in the first column.

The relevant – though not very surprising – observation here is that Table 2 is roughly complementary with Table 1: settings with many plusses in one table tend have few plusses in the other. Thus, the combination of these two tables provides a fairly balanced picture of the well known trade-off between expressivity and efficiency in the context of association rule mining. It also demonstrates there is no dichotomy item sets - queries (APRIORI-WARMR), but rather a gradual change in the trade-off between expressivity and efficiency, with a number of "intermediate" problems that have received considerable attention. Finally, the two tables provide a blueprint for a single integrated system that uses Table 1 to determine the minimal level of expressivity required and Table 2 to fire the maximally efficient algorithm available within that setting. In such a system, WARMR would be the "catch-all" method.

In the next section we point at some unexplored settings for frequent pattern discovery and propose WARMR as a first, but not necessarily ultimate approach.

4 New instances of frequent query discovery

We first present two instances of frequent query discovery that have not been addressed before: one obvious "gap" in Table 1, and one more complex setting which requires an extra row in that table. For both settings we show how they can be handled with WARMR. To conclude, we propose the ILP approach as a benchmark method for these and other unexplored variants of frequent pattern discovery.

4.1 Many items per transaction, with properties

Reconsider Table 1, and notice both the features "many items per transaction" and "item or transaction attributes" have been addressed in isolation, but never in combination. In fact, settings where these features are addressed in isolation can be mapped: a set of facts $\{tr(tid,i_1)\leftarrow;\ldots;tr(tid,i_n)\leftarrow\}$ maps to a singleton $\{tr(tid,i_1,\ldots,i_n)\leftarrow\}$. However, when a database combines both features, a mapping to one of the existing problems does not seem to be possible without loss of information.

Imagine, for instance, a database with facts *tr(tid,itemtype,size,promoted)*← that denote for each item in the transaction: the size of the package, and whether or not the item is in promotion. A sample of such a database is shown below [2].

tr(1,beer,sixpack,no)←	*tr(2,beer,crate,yes)*←
tr(1,chips,familysize,yes)←	*tr(2,chips,box,no)*←
...	...

Let us now look at two possible strategies for transforming this setting to itemsets. First, one could blow up the number of itemtypes and introduce an item *itemtype_size_promoted* for all combinations that occur. A first objection to this solution is that, especially with a high number of (many-valued) properties, this transformation will result in an exponential number of infrequent items. Moreover, even if this transformation is practicable, it would disallow the discovery of frequent combinations of the original item types.

As as second attempt we could add the individual properties as extra itemtypes, as is done with item hierarchies and in [31]. This, indeed, allows the discovery of patterns such as

?- *tr(Tid,sixpack),tr(Tid,promoted),tr(Tid,chips)*,

[2]Notice this setting has some similarity with the "multiple-instance problem" known from attribute-value learning [21].

"sixpacks, promoted things and chips". However, we lose the facility to discover something about combinations of properties, such as "promoted things in six-packs": properties *sixpack* and *promoted* are tested independently and cannot be linked to the same itemtype.

To summarize the problem, we would like itemtypes and their properties to occur both in isolation, and in any combination, e.g.

?- *tr(Tid,beer,sixpack,yes),tr(Tid,chips,familysize,P),tr(Tid,pampers,S,Q)*,

"promoted beer in a six pack, chips in a family sized package, and pampers".

With WARMR similar rules could be discovered by choosing the language bias essentially as follows:

$$\begin{array}{l} rmode(1: tr\_id(Tid)), rmode(\infty : tr(+Tid, I, S, P)), \\ rmode(\infty : eq(+I, beer)), rmode(\infty : eq(+I, chips)), \ldots \\ rmode(\infty : eq(+S, sixpack)), rmode(\infty : eq(+S, familysize)), \ldots \\ rmode(\infty : eq(+P, yes)), rmode(\infty : eq(+P, no)) \end{array}$$

where *eq* is an equality test. This bias could be easily extended to handle the case where a transaction as a whole may have properties in addition to the properties of items. In the supermarket domain, the properties of transaction may contain information about the context of shopping — such as the time or the location — about the customer, or aggregate information about the basket — such as the total value or the number of items in the basket. The task is to find frequent patterns of item and transaction properties, such as

?- *tr(Tid,cigarettes),paid_with(Tid,cash)*,

"baskets containing cigarettes and paid in cash", or

?- *cust_age(Tid,senior),tr(Tid, I, S, promoted)*,

"senior customers going for bargains". To find these two rules with WARMR, we only have to add

$$\begin{array}{l} rmode(\infty : paid\_with(+Tid, cash)), rmode(\infty : paid\_with(+Tid, credit\_card)), \ldots \\ rmode(\infty : cust\_age(+Tid, junior)), rmode(\infty : cust\_age(+Tid, senior)), \ldots \end{array}$$

to the language bias.

4.2 Related item properties and background knowledge

We also consider a case which can no longer be described as a combination of dimensions of Table 1. This setting is characterised by the presence of arbitrary relations between item properties and arbitrary background knowledge.

For an example, we can slightly modify one of the patterns shown in the previous paragraph:

?- *tr(Tid,beer,sixpack,yes),tr(Tid,chips,S,P),tr(Tid,pampers,S,Q)*,

"promoted beer in a six pack, and chips and pampers in the same type of package". Notice the S variable shared between the chips and the pampers items. And finally, an example with arbitrary background knowledge:

?- *tr(Tid,beer),customer(Tid,C),employee(Tid,E),sister(E,C)*

"beer bought by a sister of one of the employees present at that time in the shop".

The first rule can be found if WARMR's bias is extended with:

$rmode(\infty : tr(+Tid, I, +S, P))$,

the second rule can be found with

$rmode(1 : tr\_id(Tid))$,
$rmode(\infty : tr(+Tid, beer), rmode(\infty : tr(+Tid, chips), \ldots$
$rmode(\infty : customer(+Tid, N)), rmode(\infty : employee(+Tid, N))$,
$rmode(\infty : sister(+N1, +N2), rmode(\infty : uncle(+N1, +N2)), \ldots$

as language bias specifications.

4.3 ILP as a benchmark approach

It is not inconceivable that for the two settings above, and for any other setting addressed with WARMR, specialised algorithms can be developed that will outperform WARMR by several orders of magnitude, as is the case with the existing algorithms we discussed in Section 3. However, a generic tool such as WARMR could be complementary with these specialised algorithms and would offer several advantages both to users and developers.

To the users WARMR offers mainly two types of flexibility. First, the user can jump from one setting to another with just minor changes to the language bias. Individual pattern types that turn out to be of particular interest can then be mined in a second stage with specialised algorithms. An additional danger with using a specialised algorithm as a first approach is that any information which cannot be used within this method is bound to be ignored or even cut away in a preprocessing step, as illustrated in Section 4.1.

A second type of flexibility comes with the possibility to add background knowledge. Background knowledge has at least two functions in the process of knowledge discovery in databases: it can be used to (1) add information in the form of general rules, but also (2) to change with minor effort the view on the data, without going through the typically laborious preprocessing of the raw data themselves. Again, once the experiments converge on some specific setting, efficiency can be cranked up by reorganisation of the data into some very specific format.

On the other hand, for the developers of specialised algorithms WARMR can function as a benchmark, and as a verification/validation method: the special algorithm should run significantly faster, and produce the same output. In the next section on experiments, we present some preliminary indications of WARMR's performance.

5 Experiments: alarm analysis with WARMR

In this section we present experimental results with WARMR in the task of frequent query discovery. We explore some cases where previous data mining tools are not expressive enough, and consider the use of ILP as an exploration and benchmark technique for the possible development of specialized data mining tools.

For the experiments we run a prototypical implementation of WARMR, written in MASTERSPROLOG, on a Sun Ultra 2 m1170. In general, the subsequent databases $\mathbf{r}^k$ are loaded from an *Oracle7*^{TM} database. MASTERSPROLOG has been connected to this relational database to obtain full DATALOG facilities. In [20], for instance, an experiment is discussed where the database contains a 3-million word tagged corpus of Wall Street Journal news paper articles, and subsequent sentences are loaded, and parsed, one by one. For the current experiments with alarm analysis however, the whole database fits in main memory and is loaded once from a flat file.

5.1 Example application

The experimental data originates from a fault management database of a mobile communication network. The problem of discovering recurrent combinations of alarms from such databases has been considered in [24, 27, 31, 39]. Closely related data mining problems have been considered, e.g., in [6, 22, 41, 46, 50, 55, 47, 59].

The dataset consists of a sequence of 46662 alarms omitted by the network elements such as base stations and transmission devices during a period of one month. The time granularity of the data is one second. The average frequency of alarms is approximately 1500 alarms/day, or 1 alarm/minute, but since alarms tend to occur in bursts, the busiest second contains 50 alarms.

There are 180 different alarm types, which can be further classified into 10 overlapping classes. Each instance of an alarm has one of 4 urgency levels. The alarms in the dataset have been received from 2012 network management objects of 9 different types. These objects represent units of different granularities, and they form a containment hierarchy. This hierarchy gives essential information about the nature of the relationships of the objects.

The discovery task we consider is to find those combinations of alarms that are frequent. This problem is the one considered in episode discovery, but here, to the best of our knowledge, we implement a much more expressive variation than has been done before. We consider alarms with different combinations of properties, and we also consider cases where the alarms are connected, e.g., in the object hierarchy or in some other way. We cannot see any way of transforming this task to episode or sequential pattern discovery task without losing information.

5.2 WARMR inputs

Database and background knowledge. Following are some of the most important predicates used to represent the alarm data. The most obvious ones, such as *alarmtype(alarm, alarmtype)*←, relate each alarm to an occurrence time, an alarm type, several alarm classes, etc. A background clause with the head *precedes(alarm1, alarm2)* allows temporal order tests between alarms *alarm1* and *alarm2*. Some new clauses are defined in the background knowledge based on the occurrence time, to add potentially useful information such as *officehour(alarm)*.

In a similar manner, the database contains clauses *sender(alarm, object)*← that indicate the sender of each alarm; background knowledge includes clauses that derive predicates such as *ancestor(object, ancestor)*, *sibling(object1, object2)*, and *same_object(objecttype, alarm1, alarm2)*. The last one tells whether alarms *alarm1* and *alarm2* were sent from within a same object of type *objecttype*.

Finally, to represent and define windowing, we specified in the background knowledge a clause that derives *in_window(alarm1, interval, alarm2)* if *alarm2* occurs within time *interval* from *alarm1*. In the experiments we considered windows of width 120 seconds that start from an alarm.

Language bias. The most extensive language bias used in the experiments looks essentially as follows (we only show rmodes, not the typing information, and lookahead and constraint specifications):

$$\begin{array}{l}
rmode(1 : window(W)),\\
rmode(\infty : in\_window(+W, 120, A)), rmode(\infty : precedes(+A, +B)),\\
rmode(\infty : officehour(+A)), rmode(\infty : weekend(+A)),\\
rmode(\infty : weekday(+A, mon)), rmode(\infty : weekday(+A, tue)), \ldots\\
rmode(\infty : urgency(+A, 1)), rmode(\infty : urgency(+A, 2)), \ldots\\
rmode(\infty : alarm\_type(+A, 1001)), rmode(\infty : alarm\_type(+A, 2270)), \ldots\\
rmode(\infty : alarm\_class(+A, switch)), rmode(\infty : alarm\_class(+A, trans)), \ldots\\
rmode(\infty : sender\_element(+A, 95)), rmode(\infty : sender\_element(+A, 96)), \ldots\\
rmode(\infty : same\_alarmtype(+A, +B)), rmode(\infty : same\_class(+A, +B)),\\
rmode(\infty : same\_sender(+A, +B)), rmode(\infty : same\_urgency(+A, +B)),\\
rmode(\infty : sender(+A, Obj)), rmode(\infty : same\_objecttype(+Obj1, +Obj2)),\\
rmode(\infty : object\_type(+Obj, bcf)), rmode(\infty : object\_type(+Obj, trx)), \ldots\\
rmode(\infty : ancestor(+Obj1, +Obj2)), rmode(\infty : sibling(+Obj1, +Obj2)), \ldots
\end{array}$$

5.3 Results and discussion

Quality of queries The first criterion for success in data mining is that something useful is found. The frequent queries that were discovered show patterns that are more informative than the ones discovered with episodes, and episodes have been found useful in the alarm analysis task [31]. We thus expect the new queries to be even more useful; this will be confirmed by an expert evaluation of the patterns.

A specific task we considered was to describe the windows following alarms of a specific alarm type. Problems reported by this alarm are difficult to track; here the goal is to discover patterns of alarms from related objects that might help in explaining the important alarms. We present one of the patterns, in a more informative association rule format:

window(A),in_window(A,120,B),alarm_class(B,bsc_message)
⇒ (f:0.15,c:0.77)
in_window(A,120,C),alarm_class(C,trans),same_urgency(A,C),same_urgency(C,B)

i.e. with 77% confidence, and 15% frequency: "if a window contains an alarm of class *bsc-message* then it will also contain an alarm of class *trans* such that all alarms referred to will have the same urgency".

In a more general setting, where the window could start on alarms of any type, the following pattern was discovered:

window(A),in_window(A,B),sender(B,O),objecttype(O,bcf),
ancestor(O,P),objecttype(P,bsc),
in_window(A,D),alarm_class(D,bst_message)
⇒ (f:0.27,c:0.68)
precedes(B,D),urgency(B,2),alarm_class(B,bst_message)

i.e. with 68%confidence, and 27% frequency: "**if** there is in the window an alarm sent by an object of type *bcf*, and an ancestor of that object is of type *bsc*, and there is also an alarm of class *bst_message*, **then** the window will also contain two alarms of that kind where the first (the one with the *bsc* ancestor) precedes the second (the one with the *bst_message*) and moreover where the first has urgency 2 and class *bst_message*".

Performance In order to make the role and complexity of the task clear, consider the following increasingly complex cases. Table 3 contains the results of some first experiments with WARMR on the full dataset, for cases 1, 2, and 4.

- **case 1** Data: just single items with attributes. Patterns: frequent sets of attribute-values pairs. This task can be transformed to the discovery of frequent sets (each attribute-value pair is an item).

	case 1 (F=0.1)				**case 2** (F=0.25)				**case 4** (F=0.25)			
L	CGT	CET	NOC	NOFS	CGT	CET	NOC	NOFS	CGT	CET	NOC	NOFS
1	1	50	215	22	1	156	249	13	1	167	267	18
2	14	63	190	19	180	7828	1543	19	24	1433	238	38
3	19	1689	8	6	390	1190	18	15	112	6113	170	67
4	4	502	1	1	420	399	8	7	336	16897	233	114
5	0	0	0	0	235	349	1	1	904	103221	286	159
6	0	0	0	0	34	0	0	0	1983	90162	271	225

Table 3: Results of three runs with WARMR on alarm analysis. Legend: F = frequency threshold, L = level, CGT = candidate generation time (CPUs), CET = candidate evaluation time (CPUs), NOC = number of candidates, NOFS = number of frequent sets.

- **case 2** Data: several items (or events or transactions) each with attributes. Patterns: frequent collections of sets of attribute-value pairs. This task can be transformed to the discovery of sequential patterns.
- **case 3** Data: as above. Patterns: as above, but with binary predicates connecting several items. This case cannot be transformed to any sensible form without losing information. This task can lead to a combinatorial explosion (except when the binary predicates only set a total order on the items, which can be handled efficiently).
- **case 4** Data: as above, but several transactions each with several items. Patterns: as above, but also connecting items to transactions and transactions to each other. Another explosion is possible.

Discussion. There is an increase in complexity from case 3 onwards in case there are many alarms in a window, as can be seen in ther results for case 4 Table 3: evaluation of candidates at level 5 takes 103221 CPUs (28h). At that level queries are evaluated with up to twelve atoms, of which up to three in_window/3 atoms, and two precedes/2 atoms. Evaluation of such queries can take a lot of time, due to backtracking over all alarms in the window in case the query does not succeed immediately. Windows with few alarms are still processed efficiently, but in windows with bursts of several hundreds of alarms, evaluation can take up 10 mins and more. This is an example of the combinatorial explosion when there are relations with several alarm (item) variables or shared variables between alarms. Some clever incremental method for processing the windows could help, at least in specific situations such as a total order.

The problem could also be alleviated by reorganizing and rewriting the queries such that fruitless backtracking is prevented (as already suggested in

Section 3.2.4). In the current version of WARMR all queries are tested in a serial fashion, addition of query optimization techniques is planned for the near future.

6 Discussion and Related work

We here touch upon some related work. We restrict ourselves to research not explicitly addressed elsewhere in the paper. For an overview of ILP work in the context of knowledge discovery in databases, we refer to [23].

6.1 Logical paradigm: learning from interpretations

The definition of frequent query discovery and the (relatively) efficient candidate evaluation in WARMR is rooted in the *learning from interpretations* paradigm, introduced by De Raedt and Džeroski [17] and related to other inductive logic programming settings in [13]. Indeed, the $\mathbf{r}^K$'s, described in Section 3.2.4 as partitions of the database, can be formalized in first order logic as *Herbrand interpretations.* Every $\mathbf{r}^K$ in which a query succeeds is then a *Herbrand model* of that query.

The learning from interpretations paradigm has proven to be particularly suitable for the design of upgrades to popular attribute-value learning techniques. In that respect, APRIORI - WARMR is only one of the more recent additions to a sequence of similar upgrades [15]: (EXPLORA [32])-CLAUDIEN [16], CN2 [10]-ICL [18], C4.5 [49]-TILDE [8], and [33]-C0.5 [14].

6.2 Clausal discovery

Association rules $A_1 \wedge \ldots \wedge A_k \Rightarrow A_{k+1} \wedge \ldots \wedge A_n$, as introduced in Section 3.1.2, can easily be confused with a clauses $A_1 \wedge \ldots \wedge A_k \rightarrow A_{k+1} \vee \ldots \vee A_n$: both are interpreted as if-then rules with atoms A_i. We first clarify the relation between both formats, and then relate WARMR to clausal discovery engines. For the first part we will take the example of CLAUDIEN [16], because (1) it is the only algorithm to discover also non-definite clauses, and (2) its notions of confidence and frequency are close to our definitions of these concepts for association rules. In both formats we will refer to $A_1, \ldots, A_k$ as the body of the rule and to $A_{k+1}, \ldots, A_n$ as the head.

Let us start from a clause c. To test whether clause c is true w.r.t. a database $\mathbf{r}^K$, we can submit a query ?- *body(c),not(head(c))* to $\mathbf{r}^K$. Roughly speaking, if the query fails, the clause is true and is said to hold in $\mathbf{r}$, otherwise, if the query succeeds, the clause is false and does not hold. We can then define the confidence of c as the conditional probability that the whole clause is true, given that $body(c)$ is true, cf. *global accuracy* in [16].

Let us also consider an association rule a. The confidence of this rule is defined as conditional probability that $body(a) \wedge head(a)$ is true given that $body(a)$ is true.

If we write both conditional probabilities as ratio's, we have in both cases the the probability that the body is true in the denominator. Now let us assume $body(a) = body(c) = body$, focus on the nominator, and write quantification explicitly. In the case of clauses, we have joint probability $p(\exists body \wedge \forall(head(c) \leftarrow body)))$. In the case of association rules, we have joint probability $p(\exists body \wedge \exists(body \wedge head(a)))$. Since $\exists body$ is a generalisation of $\exists(body \wedge head(a))$, we can reduce the latter probability to $p(\exists(body \wedge head(a)))$.

Let us finally verify under which circumstances

$$p(\exists body \wedge \forall(head(c) \leftarrow body)) = p(\exists body \wedge head(a))$$

holds. Under these circumstances the confidence of a and c is identical and we can safely confuse them. We can reformulate the equation above as an equivalence

$$\exists body \wedge \forall(head(c) \leftarrow body) \Leftrightarrow \exists(body \wedge head(a))$$

and first look at the case where $head(c)$ and $head(a)$ contain the same atoms. An immediate observation is that the equivalence cannot hold if the heads contain more than one atom. Recall $head(c)$ is defined as a *disjunction* of atoms, whereas $head(a)$ is a conjunction of atoms. So let us now consider the case where $head(c) = head(a) = head\_atom$. Notice the equivalence relation then holds if *body* can succeed at most once, i.e. is deterministic. This is the case for instance if *body* is a purely propositional formula. With a deterministic body, we can switch between universal quantification and existential quantification in the equivalence above, and rewrite it as:

$$\exists body \wedge \exists(head\_atom \leftarrow body) \Leftrightarrow \exists(body \wedge head\_atom)$$

which holds.

There is however a more general, indirect way to map the confidence of clauses and association rules. Assume $head(a) = \neg head(c)$, so that we have

$$\exists body \wedge \forall(head(c) \leftarrow body)$$
$$\exists(body \wedge \neg head(c))$$

These two formulae are both false if *body* fails, but otherwise they are contradictory. From this, and from our definitions of confidence *conf*, we can derive that $conf(c) = 1 - conf(a)$.

So we can conclude that, with the *global accuracy* confidence measure of CLAUDIEN, association rules a can be mapped to clauses c if

1. they have the same deterministic body and an identical single atom in the head: in that case $conf(a) = conf(c)$

2. they have the same body, and the head of the one equals the negation of the head of the other: in that case $conf(a) = 1 - conf(c)$.

The discovery of clauses is handled for instance by KNOWLEDGE MINER [52], CLAUDIEN [16], MIDOS [61], RDT [30], MOBAL [35], OCD [60], and PROGOL in learning from positives only mode [43]. We now consider the key differences between these algorithms and WARMR. First, WARMR is the first ILP system to employ the efficient levelwise search method, and mine frequent queries. For levelwise search, a quality criterion is required that is monotone with respect to the specialisation relation, cf. [38]. WARMR searches a space of queries, where frequency is such a monotone quality criterion. Traditional clausal discovery engines such as CLAUDIEN search a space of clauses, where neither confidence nor frequency has the desired properties. However, these engines mostly have an any-time character, and incorporate heuristics to direct the search immediately to regions where highly confident, and frequent rules can be expected. In that sense clausal discovery engines are complimentary to WARMR, which performs an exhaustive search for frequent queries, and only in a post-processing step can discover rules that meet both the frequency and the confidence standards set by the user.

An additional advantage with frequent query discovery is that efficient sampling methods for mining association rules, such as those described in [57], are likely to carry over directly to the ILP setting.

6.3 Association rules with numerical intervals

Some extra complexity arises if items or transactions have numerical properties, and queries in $\mathcal{L}$ contain atoms that test numerical intervals. Solutions to some of the problems have been proposed, in the context of association rules, in [54, 62]. Both concentrate on heuristics for finding suitable intervals ([54]) or 2D areas ([62]) of numerical attributes for association rules.

6.4 Future work

We briefly summarize some directions to pursue the research on frequent query discovery. First, an efficient general method should be developed for query reorganisation to minimize backtracking. Second, based on Table 1 and Table 2, a user-friendly generic system could be developed that automatically selects the most efficient algorithm available. This could be done on the basis of an analysis of the user inputs, i.e. the database and the language bias. Fourth, Table 1 uncovers a number of "gaps" that could be filled with some very useful specialized algorithms. Fifth, many optimizations and techniques for mining and postprocessing frequent patterns and association rules have been proposed. Some of these, such as the sampling techniques described in [57], could probably be plugged into WARMR without much effort.

7 Conclusions

We have presented a unifying ILP approach to frequent pattern discovery and, by extension, to association rule discovery. The new task of frequent query discovery has been defined and some of the popular frequent pattern tasks have been reformulated as instantiations of frequent query discovery. In doing that, a number of dimensions of the practice of frequent pattern discovery have emerged, cf. Table 1.

In a second part, we have introduced the frequent query discovery engine WARMR and repeated the exercise of the first part on the level of algorithms. This has resulted in a second table with algorithmic dimensions, cf. Table 2, that, together with the first table, sheds some new light on the expressivity-for-efficiency trade off.

In the remaining sections, we have considered the application of WARMR to some new frequent pattern discovery tasks, and to the "real-world" problem of telecommunication alarm sequence analysis. The main motivation behind these sections was to position WARMR as a flexible "catch-all" tool that can be used both by users and developers as a first approach to a new task. In that spirit, a stand alone version of WARMR is freely available for academic purposes upon request to Luc Dehaspe.

Acknowledgements

The authors of this paper are listed in alphabetic order. Luc Dehaspe is supported by ESPRIT Long Term Research Project No 20237, ILP^2. Hannu Toivonen is supported by the Academy of Finland. This paper was conceived while Hannu Toivonen was visiting the Department of Computer Science, Katholieke Universiteit Leuven.

The authors are grateful to Luc De Raedt for comments on the paper, and for many fundamental ideas and discussions, to Hendrik Blockeel, Bart Demoen, and Wim Van Laer for their share in the implementation of WARMR. Data for the telecommunication alarm sequence analysis experiments is provided by Juka Leino from Nokia Telecommunications.

References

[1] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative Bias for Specific-To-General ILP Systems. *Machine Learning*, 1995. To appear.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*, pages 207 – 216, Washington, D.C., USA, May 1993. ACM.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE'95)*, pages 3 – 14, Taipei, Taiwan, Mar. 1995.

[5] F. Bergadano and D. Gunetti, editors. *'Inductive Logic Programming: from Machine Learning to Software Engineering.* The MIT Press, 1995.

[6] C. Bettini, X. S. Wang, and S. Jajodia. Testing complex temporal relationships involving multiple granularities and its application to data mining. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'96)*, pages 68 – 78, Montreal, Canada, June 1996.

[7] H. Blockeel and L. De Raedt. Relational knowledge discovery in databases. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 199–212. Springer-Verlag, 1996.

[8] H. Blockeel and L. De Raedt. Top-down induction of logical decision trees. In *Proceedings of the Ninth Dutch Conference on Artificial Intelligence (NAIC-97)*, 1997.

[9] H. Blockeel, L. De Raedt, N. Jacobs, and B. Demoen. Scaling up inductive logic programming. Submitted, 1998.

[10] P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, 3(4):261–284, 1989.

[11] G. Das, R. Fleischer, L. Gasieniec, D. Gunopulos, and J. Kärkkäinen. Episode matching. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM '97)*, pages 12 – 27, Aarhus, Denmark, June 1997.

[12] L. De Raedt, editor. *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications.* IOS Press, 1996.

[13] L. De Raedt. Induction in logic. In R. Michalski and W. J., editors, *Proceedings of the 3rd International Workshop on Multistrategy Learning*, pages 29–38, 1996.

[14] L. De Raedt and H. Blockeel. Using logical decision trees for clustering. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 133–141. Springer-Verlag, 1997.

[15] L. De Raedt, H. Blockeel, L. Dehaspe, and W. Van Laer. Three companions for first order data mining. In N. Lavrač and S. Džeroski, editors, *Inductive Logic Programming for Knowledge Discovery in Databases*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1998. To appear.

[16] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[17] L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[18] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 5th Workshop on Algorithmic Learning Theory*, volume 997 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 1995.

[19] L. Dehaspe and L. De Raedt. DLAB: A declarative language bias formalism. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems (ISMIS96)*, volume 1079 of *Lecture Notes in Artificial Intelligence*, pages 613–622. Springer-Verlag, 1996.

[20] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 125–132. Springer-Verlag, 1997.

[21] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[22] C. Dousson, P. Gaborit, and M. Ghallab. Situation recognition: Representation and algorithms. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 166 – 172, Chambery, France, Aug. 1993.

[23] S. Džeroski. Inductive logic programming and knowledge discovery in databases. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 118–152. The MIT Press, 1996.

[24] R. M. Goodman and H. Latin. Automated knowledge acquisition from network management databases. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management, II*, pages 541 – 549. Elsevier Science Publishers B.V (North-Holland), Amsterdam, The Netherlands, 1991.

[25] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, pages 209 – 216, Tucson, Arizona, 1997. ACM.

[26] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 420 – 431, Zürich, Switzerland, 1995.

[27] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. Knowledge discovery from telecommunication network alarm databases. In *Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, pages 115 – 122, New Orleans, Louisiana, Feb. 1996. IEEE Computer Society Press.

[28] M. Holsheimer, M. Kersten, H. Mannila, and H. Toivonen. A perspective on databases and data mining. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 150 – 155, Montreal, Canada, Aug. 1995. AAAI Press.

[29] J. Kietz and M. Lübbe. An efficient subsumption algorithm for inductive logic programming. In *Proceedings of the 11th International Conference on Machine Learning*. Morgan Kaufmann, 1994.

[30] J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive logic programming*, pages 335–359. Academic Press, 1992.

[31] M. Klemettinen, H. Mannila, and H. Toivonen. Rule discovery in telecommunication alarm data. *Journal of Network and Systems Management*, June/July 1998.

[32] W. Klösgen. Explora: A multipattern and multistrategy discovery assistant. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. The MIT Press, 1996.

[33] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.

[34] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[35] G. Lindner and K. Morik. Coupling a relational learning algorithm with a database system. In Y. Kodratoff, G. Nakhaeizadeh, and G. Taylor, editors, *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, Heraklion, Crete, Greece, 1995.

[36] H. Lu, R. Setiono, and H. Liu. Neurorule: A connectionist approach to data mining. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 478 – 489, Zürich, Switzerland, 1995.

[37] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 146 – 151, Portland, Oregon, Aug. 1996. AAAI Press.

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

[39] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259 – 289, 1997.

[40] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[41] R. A. Morris, L. Khatib, and G. Ligozat. Generating scenarios from specifications of repeating events. In *Second International Workshop on Temporal Representation and Reasoning (TIME-95)*, Melbourne Beach, Florida, Apr. 1995.

[42] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13, 1995.

[43] S. Muggleton. Learning from positive data. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 225–244. Stockholm University, Royal Institute of Technology, 1996.

[44] S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[45] C. Nédellec, H. Adé, F. Bergadano, and B. Tausend. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 82–103. IOS Press, 1996.

[46] T. Oates and P. R. Cohen. Searching for structure in multiple streams of data. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, pages 346 – 354, San Francisco, CA, July 1996. Morgan Kaufmann.

[47] B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 351–354, 1996.

[48] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

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

[50] R. Sasisekharan, V. Seshadri, and S. M. Weiss. Data mining and forecasting in large-scale telecommunication networks. *IEEE Expert, Intelligent Systems & Their Applications*, 11(1):37 – 43, 1996.

[51] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 432 – 444, Zürich, Switzerland, 1995.

[52] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. The MIT Press, 1996.

[53] R. Srikant and R. Agrawal. Mining generalized association rules. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 407 – 419, Zürich, Switzerland, 1995. Morgan Kaufmann.

[54] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'96)*, pages 1 – 12, Montreal, Canada, 1996.

[55] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Advances in Database Technology—5th International Conference on Extending Database Technology (EDBT'96)*, pages 3 – 17, Avignon, France, 1996.

[56] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD'97)*, pages 67 – 73. AAAI Press, 1997.

[57] H. Toivonen. Sampling large databases for association rules. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 134 – 145, Mumbay, India, Sept. 1996. Morgan Kaufmann.

[58] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville, MD, 1988.

[59] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In R. Snodgrass and M. Winslett, editors, *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'94)*, pages 115 – 125, Minneapolis, MI, June 1994. ACM.

[60] I. Weber. Discovery of first-order regularities in a relational database using offline candidate determination. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 288–295. Springer-Verlag, 1997.

[61] S. Wrobel. An algorithm for multi-relational discovery of subgroups. In J. Komorowski and J. Zytkow, editors, *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD '97)*, pages 78 – 87. Springer-Verlag, 1997.

[62] K. Yoda, T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Computing optimized rectilinear regions for association rules. In D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD'97)*, pages 96 – 103. AAAI Press, 1997.