A perspective on inductive databases

Luc De Raedt
Institut für Informatik, Albert-Ludwigs-University,
Georges Koehler Allee 79, D-79110 Freiburg, Germany
deraedt@informatik.uni-freiburg.de

ABSTRACT

Inductive databases tightly integrate databases with data mining. The key ideas are that data and patterns (or models) are handled in the same way and that an inductive query language allows the user to query and manipulate the patterns (or models) of interest.

This paper proposes a simple and abstract model for inductive databases. We describe the basic formalism, a simple but fairly powerful inductive query language, some basics of reasoning for query optimization, and discuss some memory organization and implementation issues.

Keywords

inductive databases, inductive querying, constraint-based mining

1. INTRODUCTION

Ever since the start of the field of data mining, it has been realized that the data mining process should be supported by database technology. In recent years, this idea has been formalized in the concept of inductive databases [28]. Inductive databases are databases that, in addition to data, also contain patterns, i.e., generalizations extracted from the data. Within the inductive database framework knowledge discovery in databases is modelled as an interactive process in which users can query data as well as patterns. To this aim a so-called inductive query language is used.

A number of specialized inductive query languages have been proposed and implemented, e.g., MINE RULE [35], MSQL [29], DMQL [22] and XMine [10]. Most of these inductive query languages extend an existing database query (SQL or XML) language with some primitives to support data mining. The combination of a data mining algorithm (usually some variant of Apriori [1]) with a language such as SQL offers some interesting querying abilities. Other researchers have argued that data mining primitives should be as close as possible to those of traditional query languages [20; 6].

The work on inductive query languages has been complemented by several approaches to constraint based mining [3; 23; 24; 38; 31; 16]. Constraint based mining provides the user with certain primitives (such as frequency and syntactic constraints) to specify the patterns of interest. This line of research has focussed on discovering useful constraints that can be processed efficiently.

Despite these many interesting contributions, we are still far away from a deep understanding of inductive databases. It appears to the author that several important components of inductive databases (such as traditional databases and constraint based mining techniques) are already well understood, but that the overall picture is still unclear. Indeed, with a few notable exceptions [5; 28; 18; 16], few works have addressed architectural or design issues in inductive databases. A simple theory that gives a clear answer to simple questions such as "What is an inductive database?" and "What is the functionality of an inductive database?" seems to be missing and yet of central importance to the further development of this idea.

Although it is probably too early to give a final answer to the above questions, this paper wants to contribute a partial answer to this question that is grounded in database theory and logic. The answer consists of a simple but fairly powerful inductive database concept that clearly exemplifies the nature as well as the functionality of an inductive database. In formulating this answer, we start from first principles rather than from complex state-of-the-art techniques and algorithms, so that we do not get lost in technical details and are, we hope, able to address the true design issues in inductive databases. The resulting design does not pretend to be realistic and is – from a practical perspective – overly simple. It is merely meant to give insight into the possible nature and properties of inductive databases and these principles are generally applicable. One specific context in which they are successfully being applied is that of the molecular feature miner MolFea [13; 31], in which the user can query for molecular patterns or fragments of interest using primitives such as frequency and generality.

The paper is organized as follows: Section 2 introduces the data and pattern components of inductive databases and illustrates them using MolFea; Section 3 introduces the inductive query language for specifying patterns of interest; in Section 4 it is shown how queries can be used to create, update and modify data and pattern sets; Section 5 is concerned with reasoning about queries with the aim of query evaluation and optimization; Section 6 contains some ideas and challenges concerned with memory organization and data structures, and finally, in Section 7, we conclude.

2. INDUCTIVE DATABASES

Inductive databases do not only store data but also patterns: the patterns are first class objects. Thus an inductive database $I(\mathcal{D}, \mathcal{P})$ consists of a data component $\mathcal{D}$ and a pattern component $\mathcal{P}$. We assume that both the data and the pattern components $\mathcal{D}$ and $\mathcal{P}$ are sets of sets. This assumption is motivated by an analogy with traditional relational databases. A relational database can be considered a set of relations where each relation is a set of tuples. So, relational databases are - as our data component - sets of sets. The assumption is further justified because in data mining one is often coping with different data sets. Indeed, one distinguishes the training from the test set, the set of positives from the set of negatives, the set of instances in a given cluster, the correctly classified examples from the incorrectly classified ones, etc. This justifies the assumption that $\mathcal{D}$ is a collection of data sets. Each data set $D \in \mathcal{D}$ contains different examples or instances, which will be denoted by e or i.

Everything that was said about the data component also applies to the pattern component. Indeed, during the knowledge discovery process, one will often work with different sets of patterns, each of which may reside in the inductive database. These different sets may correspond to different hypotheses constructed on different data sets during cross-validation, to hypotheses constructed under various parameter settings (e.g., levels of frequency), to user supplied patterns, post-processed patterns, etc. Pattern sets will be denoted by P and their elements, i.e., the patterns, by $p, p_i, \ldots$.

In order to illustrate the key concepts of our inductive database design, we will employ the pattern domain of strings. This pattern domain, as many of the other ideas presented in this paper, are motivated by the domain specific inductive database MolFea [31]. We will therefore first briefly review the MolFea setting and then abstract from MolFea in defining the pattern domain of strings. At the same time, we hope that the MolFea experiences and setting will convince the reader of the practical relevance of the presented inductive database concepts.

MolFea is a domain specific inductive database for mining features of interest in sets of molecules. The examples in MolFea are thus molecules, and the patterns are molecular fragments. More specifically, in [31] we employed the 2D (graph) structure of molecules, and linear sequences of atoms and bonds as fragments. An example molecule named Azidothymidine (AZT), a commonly used drug against HIV, is illustrated in Figure 1. Two interesting molecular fragments discovered using MolFea are:

```
N=N=N-C-C-C-n:c:c:c=O
N=N=N-C-C-C-n:c:n:c=O
```

In these fragments, `'C'`, `'N'`, `'Cl'`, etc. denote elements[1], and `'-'` denotes a single bond, `'='` a double bond, `'#'` a triple bond, and `':'` an aromatic bond. The two fragments occur in AZT because there exist labelled paths in the graph AZT that correspond to these fragments.

Let us now introduce the string pattern domain, which is an abstraction of MolFea. This pattern domain should also be useful for other applications in bioinformatics involving, e.g., DNA/RNA or proteins. In the string pattern domain, examples as well as patterns are strings expressed in a language $\mathcal{L}_\Sigma$ over an alphabet Σ. Furthermore, a pattern p matches or covers an example e if and only if p is a substring of e, i.e. the symbols of p occur at consecutive positions in e. An

[1]Elements involved in aromatic bonds are written in lowercase.

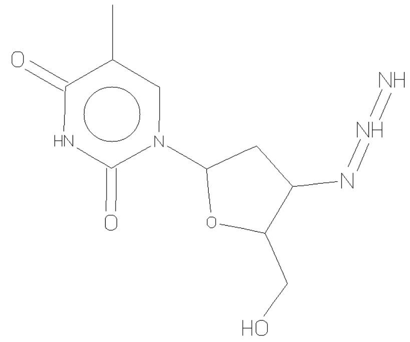

Figure 1: Chemical Structure of Azidothymidine

example toy database in this context could be:

- $e_1 = aabbcc$; $e_2 = abbc$; $e_3 = bb$; $e_4 = abc$; $e_5 = bc$; $e_6 = cc$
- $D_1 = \{e_1, e_2, e_3\} = \{aabbcc, abbc, bb\}$;
  $D_2 = \{e_4, e_5, e_6\} = \{abc, bc, cc\}$;
  $D_3 = D_1 \cup D_2$
- $p_1 = abb$; $p_2 = bb$; $p_3 = cc$
- $P_1 = \{p_1, p_2, p_3\}$.
- $\mathcal{D} = \{D_1, D_2, D_3\}$ and $\mathcal{P} = \{P_1\}$

Instead of using the pattern domain of strings, one could also employ other domains such as the data miner's favorite item sets. Then, if $\mathcal{I}$ is the set of items considered, examples e as well as patterns p are subsets of $\mathcal{I}$. Data and pattern sets are then sets of item-sets.

3. AN INDUCTIVE QUERY LANGUAGE

One of the crucial reasons behind the success of relational databases is that relational query languages, such as the relational algebra, are fairly powerful but yet reasonably simple.[2]

What should an inductive query language look like? We will assume that the result of an inductive query is either a pattern set or a data set. This assumption guarantees the so-called *closure* property [28; 6]. The closure property is again justified by analogy with relational databases, where the result of a query is always a relation; one can argue that the reason for the success of relational algebra was due to the closure property and not to its inherent strength.

Given that we distinguish data sets from pattern sets, we need two types of queries: those that generate data sets and those that generate pattern sets. Queries that start from both data sets as well as pattern sets are sometimes called *cross-over* operations [6].

Motivated by the use of logic for both databases and constraint based mining as well as by the intimate relationship between logic and set theory, we propose to use logic to design inductive query languages. Using logic should facilitate the definition of the semantics of the query language,

[2]It is easy to give examples of queries that one cannot express using relational algebra; just consider aggregates. Thus having simple examples that fall outside the query language does not necessarily mean that the language is fatally flawed.

should allow to reason about queries and their execution, and should also result in declarative querying possibilities.
In our example inductive query language, we allow for queries q that either contain no variables or contain exactly one variable τ as in $q(\tau)$. Queries without variables will be interpreted as true or false; queries $q(\tau)$ with a single variable τ will be interpreted as sets $\{i \mid q(i)\ is\ true\}$; queries with two or more variables are not allowed. Let us now define a simple though powerful query language (inspired on MolFea) for our inductive database. Again, we wish to stress that the list of primitives sketched is by no means exhaustive, it is meant for illustrative purposes and it can easily be extended with other primitives, with other pattern domains (such as item sets [1], episodes [34], Datalog queries [12], etc.), and even with primitives for other data mining tasks (such as clustering [42]).

- Let g and s be strings. Then g *is more general than* s, notation $g \preccurlyeq s$, if and only if g is a substring of s. E.g., on our earlier example, $p_2 \preccurlyeq p_1$ evaluates to true, and $p_3 \preccurlyeq p_1$ to false. This primitive − defined in the context of strings − applies to virtually any pattern domain.

- The $\preccurlyeq$ relation can now be used in primitive constraints of the type $p \preccurlyeq p', \tau \preccurlyeq p$, $\neg(\tau \preccurlyeq p)$, $p \preccurlyeq \tau$, and $\neg(p \preccurlyeq \tau)$, where τ denotes the target pattern and p and p' specific patterns. E.g., $\tau \preccurlyeq ab$ yields as solutions the set of substrings of ab, i.e., $\{\epsilon, a, b, ab\}$.

- Let p be a pattern and D a data set, i.e., a set of examples. Then $freq(p, D) = card\{e \in D \mid p \preccurlyeq e\}$, where $card(S)$ denotes the cardinality of the set S. So, $freq(p, D)$ denotes the number of instances in D covered by p, i.e., the frequency of p in D. E.g., $freq(p_3, D_2) = 1$.

- The $freq(p, D)$ construct can be used in the following type of constraints: $freq(p, D) \geq t$, $freq(p, D) \leq t$, $freq(\tau, D) \geq t$ and $freq(\tau, D) \leq t$ where t is a numerical threshold, τ is the queried pattern or string, p is a specific pattern and D is a data set. Notice that $freq$ is usually involved in cross-over operations. E.g., $freq(\tau, D_2) \geq 2$ yields the set of all substrings of bc.

- Given that we work with sets it will be useful to employ traditional set operations: $i \in I$, $i \notin I$, $\tau \in I$ as well as $\tau \notin I$ where τ is the queried pattern, i an element and I a set.

- Finally, the language $\mathcal{IL}$ consists of any boolean expression b involving the above introduced primitives. We allow for the usual boolean connectives $\wedge, \vee, \neg$[3].

So, within the sketched language, we can formulate the following queries:

- All traditional set operations can be performed; e.g. the query $(\tau \in D_1) \vee (\tau \in D_2)$ denotes the set $D_1 \cup D_2$. For ease of expression and compactness, we will sometimes use the (simpler) set notation in queries.

- Traditional minimal frequency queries can be performed, e.g., $freq(\tau, D_1) \geq 2$.

- One can also obtain the set of examples in D_3 covered by a given pattern p_2 using $(\tau \in D_3) \wedge (p_2 \preccurlyeq \tau)$; given the example inductive database listed above, this query returns the set D_1.

- A complex query such as
$$(freq(\tau, D_{pos}) \geq n) \wedge\ (freq(\tau, D_{neg}) \leq m)$$
asks for the set of patterns that are frequent on the positive examples D_{pos} and infrequent on the negatives in D_{neg}. On our toy database, the query
$$(\tau \in P_1) \wedge (freq(\tau, D_1) \leq 0) \wedge (freq(\tau, D_2) \geq 1)$$
would yield the set $\{cc\}$.

The data mining primitives are an extension of those employed in the MolFea system for mining molecular features [31]. The key differences are that MolFea does not support the set oriented primitives and that it only allows for conjunctive queries. Here, arbitrary boolean queries are supported.
Notice that even though the inductive query language is simple, the range of queries that can be expressed is quite large. Two features are especially important and distinguish our language from other ones: the use of arbitrary boolean queries (introduced in [16]) and the ability to formulate minimum as well as maximum frequency constraints over multiple data sets (introduced in [13]). For building efficient query solvers, it is crucial that the key primitives (frequency and generality) satisfy the monotonicity or anti-monotonicity property[4]. A constraint c is *anti-monotonic* (resp. *monotonic*) w.r.t. generality whenever
$$\forall\ patterns\ s, g : (g \preccurlyeq s) \wedge (s \in sol(c)) \rightarrow (g \in sol(c))$$
(resp. $(g \in sol(c)) \rightarrow (s \in sol(c))$). Anti-monotonic (resp. monotonic) constraints have the property that whenever a pattern s satisfies the constraint, all its generalizations (resp. specializations) will also satisfy the constraint. The basic anti-monotonic constraints in our framework are: $(\tau \preccurlyeq p)$ and $freq(\tau, D) \geq m$, the basic monotonic ones are $(p \preccurlyeq \tau)$ and $freq(\tau, D) \leq m$. Furthermore the negation of a monotonic constraint is anti-monotonic and vice versa. It would be relatively straightforward to extend our language and solvers with other primitives that are monotonic or anti-monotonic. One important challenge for inductive querying concerns the use of primitives that are neither monotonic nor anti-monotonic (cf. [40]). As an example constraint consider $acc(\tau, D_{pos}, D_{neg}) \geq 0.8$. The answer set of this constraint contains all patterns τ that have a minimum accuracy of 80 per cent on the data sets D_{pos} and D_{neg}. Here, we could define the accuracy as
$$acc(\tau, D_{pos}, D_{neg}) = \frac{freq(\tau, D_{pos})}{freq(\tau, D_{pos}) + freq(\tau, D_{neg})}$$
(provided that the frequencies are not equal to 0).
Efficiently evaluating queries within inductive database languages such as $\mathcal{IL}$ is one of the most important challenges in inductive databases. From a logical point of view, inductive database queries can - as their traditional counter parts - be decomposed into their constituents, which correspond - for $\mathcal{IL}$ - to the primitives outlined above. The

[3] If desired, one might also use $\Longrightarrow$ and $\Longleftrightarrow$.

[4] Interesting variants and extensions of these notions are studied in a recent paper by [40].

problem is then to combine them in such a way that 1) the available solvers can compute the answers, and 2) the computational resources needed in this process are minimized. To illustrate this problem, consider the query $(freq(\tau, D) \geq t_d) \wedge (freq(\tau, E) \geq t_e)$. There are various ways of finding the solution set to this query. First, one could compute $(freq(\tau, D) \geq t_d)$ and then remove those elements not satisfying $(freq(\tau, E) \geq t_e)$ (or the other way around). Secondly, one could compute the answers to each of the conjuncts independently and then compute the intersection. Thirdly one could compute the result using the overall conjunction directly (e.g., using a level wise search using the conjunction as the constraint). Which of these is to be preferred will depend on the data sets, thresholds and algorithms available. The problem of deciding among these is similar to that in traditional query optimization. It seems therefore likely that similar solutions could work as well. One might e.g. want to use a cost estimates for comparing different execution plans for a given query. It remains - so far - an open question as to what cost estimates would be useful and appropriate for inductive queries. At the same time, it should be clear that reasoning about inductive queries (cf. Section 5) and efficient data structures and memory organization (cf. Section 6) will be of key importance to develop efficient query solvers.

4. OPERATIONS

So far, we have mainly addressed architectural and querying aspects of inductive databases. We have not yet addressed how to create and modify an inductive database. Again, it is useful to exploit the analogy with existing databases to the maximal possible extent. As in SQL, we therefore use creation, deletion and update operations as well as regular sets and view sets. This results in the operations sketched below, which are also inspired by ongoing work on the MineRULE system by Rosa Meo (personal communication). Again, we assume a perfect symmetry among the data and the pattern components. So even though we only list the operations on the data component, every operation on the data has an equivalent one on patterns.

create data set D **as** q_D**:** inserts the new data set D obtained by evaluating q_D into the data component $\mathcal{D}$ of the inductive database $I(\mathcal{D}, \mathcal{P})$.

create view data set D **as** q_D**:** inserts the new virtual data set D defined by the inductive query q_D into the data component $\mathcal{D}$ of the inductive database $I(\mathcal{D}, \mathcal{P})$.

delete data set D**:** deletes the (view or regular) data set D from the data component $\mathcal{D}$ of the inductive database $I(\mathcal{D}, \mathcal{P})$.

update data set D **(add | delete)** q_D**:** either adds or deletes the examples in q_D to/from the set D.

The provided operations are similar to those supported in a relational database. First, the result of an inductive data query can be stored into a new data set. Second, one can also define virtual or view data sets. View data sets behave as views in a relational database. Basically, the defining query is stored (and possibly materialized as well), and the data set can be queried just as if it were a normal set. Changes or updates to the data and pattern sets after the introduction of the view are dynamically reflected in the view data set. Updates on views can be complex. Thirdly, it is possible to delete and modify a data set. Of course, in any possible realistic inductive database more complex update operations will be available.

Whereas the sketched operations are unsurprising for the data component, they are less straightforward when applied to the pattern component. Let us illustrate and motivate the key concepts on our inductive toy-database. Assume the following statements are given:

create data set D_4 **as** $(\tau \in D_3) \wedge (a \preccurlyeq \tau)$**;**
create pattern view P_2 **as** $(b \preccurlyeq \tau) \wedge freq(\tau, D_4) \geq 2$;
update data set D_4 **delete** $\{abbc\}$;

This sequence of commands has the following effects on our database:

- Immediately after executing the first command and before the update command, D_4 becomes $\{e_1, e_2, e_4\} = \{aabbcc, abbc, abc\}$.
- The value of the pattern view P_2 before the update command is $\{b, ab, bb, bc, abb, bbc, abbc\}$.
- After the update command, the value of D_4 is $\{aabbcc, abc\}$ and the value of the pattern view P_2 becomes $\{b, ab, bc\}$.

This example illustrates a form of incremental data mining where changes to the underlying data sets have an immediate effect on the defined pattern sets. This is realized through the use of pattern views. The use of pattern views is interesting from a practical perspective. Indeed, it is easy to imagine a basket analysis scenario where one always wants to query the patterns that hold in the transaction data of the last week. Dealing efficiently with such incremental data mining and pattern views seems an important challenge for data mining. An example of that illustrates some of the issues in pattern view maintenance, is presented in Section 6.2.

5. REASONING

A theory of inductive databases can only be useful if it offers the possibility to reason about data mining and its processes. This section explores the possibilities of our logical view for reasoning about query evaluation and optimization. Such reasoning processes must start from logical axioms.

The reason why a logical axiomatization of constructs in the language are useful is given by the following two properties:

P1 For all queries q_1 and q_2 we have $(q_1 \Longrightarrow q_2)$ if and only if for all inductive databases I we have: $sol(q_1, I) \subseteq sol(q_2, I)$, where $sol(q_i, I)$ denotes the set of solutions to query q_i in the inductive database.

P2 For all queries q_1 and q_2 we have $(q_1 \Longleftrightarrow q_2)$ if and only if for all inductive databases I we have $sol(q_1, I) = sol(q_2, I)$. (This is, of course, a corollary of **P1**.)

These properties state that logical implication or entailment among queries directly corresponds to the subset relation among its answer sets.

For the sketched inductive query language, the axioms listed below should be useful. The list is only meant to illustrate the kind of axioms that query optimizers could use. Studying alternative axiomatizations and their properties remains an important topic for further research. In this context, Calders and Paredaens have recently contributed an interesting axiomatization of frequent sets, cf. [9].

A1 an axiomatization of finite sets;

A2 an axiomatization of the partial order $\preccurlyeq$, i.e. reflexivity, transitivity, and possibly anti-symmetry;

A3 an axiomatization of the total order $\leq$ for numbers;

A4 For all data sets D_1, D_2 and patterns p: $D_1 \subseteq D_2 \Longrightarrow freq(p, D_1) \leq freq(p, D_2)$; and

A5 For all patterns p_1, p_2 and data sets D: $p_1 \preccurlyeq p_2 \Longrightarrow freq(p_1, D) \geq freq(p_2, D)$.

The axioms **A1** could be used for reasoning about the relationships among the various sets in the inductive database and queries. The axioms **A3** are intended to support reasoning about the numerical thresholds in the frequency constraints. The axioms **A4** and **A5** specify the monotonicity properties of $freq$ for both of its arguments. The above axioms can be used in a variety of different ways. Let us illustrate these on some representative examples.
Consider queries q_3 and q_4 such that **A1-5**$\models q_3 \Longrightarrow q_4$. E.g., q_3 and q_4 could be $freq(\tau, D) \geq 3$ and $freq(\tau, D) \geq 2$, or alternatively $freq(\tau, D) \geq m$ and $freq(\tau, D \cup E) \geq m$. The following situations can be imagined, cf. [2].

- Query q_3 needs to be answered and assume that the solution set q_4 has already been stored as pattern set P_4 of the inductive database. Then q_3 can be solved by evaluating q_3 on all elements of P_4. If the results of q_3 must be stored as a pattern set P_3 in the database, the inductive database management system may store P_4 as the union of the two sets P_3 and $P_4 - P_3$ for compactness reasons.

- Query q_4 needs to be answered and assume that the solution set q_3 has already been stored as pattern set P_3 of the inductive database. The inductive database management system can immediately retrieve the answers P_3 and present them to the user while computing the missing answers in $P_4 - P_3$. This may reduce the computational resources needed for answering the queries under certain conditions. Furthermore, the set P_4 may again be stored as the union of the two subsets P_3 and $P_4 - P_3$ for compactness reasons.

Notice that the very same observations and situations apply to queries that return data sets. Furthermore, this type of reasoning is also required to effectively handle the pattern views introduced earlier. Logical reasoning could be used to decide when and how to update materialized views. In this context, one might – by analogy to the traditional view update problem in databases – also consider the pattern view update problem. The pattern view update problem consists of deciding how the underlying data and pattern sets need to be updated to insert or delete a pattern to/from the pattern view. This corresponds to a kind of *what if* questions.

6. INTERNAL REPRESENTATIONS

Efficient data structures and memory management are crucial for database theory. Therefore, we also discuss some of the memory management issues and challenges for inductive databases.

6.1 Representing Pattern Sets

First, at the level of the data and pattern component, it will be useful to keep track of the subset relation. Indeed, when sets D_1 and D_2 are present in the data base and $D_1 \subseteq D_2$, this fact should be represented explicitly. Storing these properties in a systematic way amounts to representing the whole structure of the data and pattern component. This will be useful in the context of query optimization. Also, other facts about the relationship among different pattern sets may be useful (e.g., when one pattern set is a condensed representation of another one, cf. below).
Secondly, it will be useful to explicitly represent the $\preccurlyeq$ relation among the patterns in the pattern set, especially for pattern sets that are irregular w.r.t. the $\preccurlyeq$ relation (e.g., when they are not convex or representable by border sets, cf. below). Explicitly representing the $\preccurlyeq$ relation will allow the inductive database management system to optimize queries of the type $(\tau \in P) \wedge q(\tau)$ where P is a pattern set and q a query in the language $\mathcal{IL}$ introduced above. Using the internal representation of $\preccurlyeq$, the constraint $\tau \in P$ can be pushed into the solver. E.g., if $q(\tau) = (freq(\tau, D) \geq m)$, a variant of the level wise algorithm could efficiently generate as candidates only those elements that belong to P. The alternative would be to search the space of all patterns, and then filter out those that do not belong to P. For ill-structured or small pattern sets this form of post-processing is likely to be less efficient, cf. also [24].

6.2 Operations on Pattern Sets

It is not only important to develop efficient data structures for representing pattern sets, but also to devise operations on these data structures. Such operations would directly implement the primitive constraints and the logical operations in the inductive database language. Thus they could serve as the elementary operations (a kind of algebra) for query evaluation as well as query optimization. Furthermore, the intermediate pattern sets could be cached for further use in interactive querying sessions. To illustrate this point, let us assume that pattern sets are represented using a directed acyclic graph; Figure 2 shows a graph corresponding to the materialized pattern set $P_2 = \{b, ab, bb, bc, abb, bbc, abbc\}$ for the pattern view $(b \preceq \tau) \wedge freq(\tau, D_4) \geq 2$, where D_4 is initially $\{aabbcc, abbc, abc\}$[5]. In a pattern graph, nodes correspond to patterns and edges to the *generalization* relation. The labels of the nodes denote properties of the patterns. The most important property is the membership flag with values "+" and "-". The label "+" (resp. "-") denotes that the corresponding pattern belongs (resp. does not belong) to the pattern set represented by the graph. So, a pattern graph may also contain information about patterns outside the pattern set it represents, e.g. $a \in P_2$ because of the la-

[5]Usually, one would not employ graph structures for these purposes, but some special purpose tree structure such as FP-trees [24] for item sets or version space trees for string patterns [16]. However, in order to simplify the exposition and to focus on the key ideas, we choose graphs rather than these more complex data structures.

bel "-". Furthermore, from Figure 2, one can directly read that b belongs to P_2 and also that (within P_2) it has three minimal specializations, i.e. ab, bb and bc. This kind of information should be useful when searching for patterns in P_2 that satisfy certain constraints (cf. Section 6.1). The other labels on the nodes representing a pattern denote coverage information. Indeed, the second label in a node encodes the indexes of the examples (here in $D_4 = \{e_1, e_2, e_4\}$) that are covered by the pattern, the third one, those (in D_4) that are not covered, where "*" denotes the empty set. E.g., bb covers e_1 and e_2 but not e_4. Given data structures such as pattern graphs, an inductive database language can be implemented using operations on these data structures. As an example consider the pattern graph operations "$\cup$", "$\cap$" and "$-$" corresponding to the logical operations "$\wedge$", "$\vee$" and "$\neg$"[6]. These operations would compute the union, intersection or difference of two pattern graphs. They are useful to decompose queries of the form $q_1 \wedge q_2$; $q_1 \vee q_2$ and $q_1 \wedge \neg q_2$. Indeed, these queries can be answered by first computing the pattern graphs w.r.t. q_1 and q_2 and then performing the appropriate operation on the resulting pattern graphs. Other operations that are likely to be useful could manipulate the labels of the nodes in a pattern graph. The labels w.r.t. coverage could be modified in order to reflect an update in the pattern view (e.g., as the result of deleting or adding an element of the set D_4 used in the definition of the pattern view). In addition, updates to the underlying data set would also trigger changes in the membership label. Interesting and challenging situations occur when new nodes need to be inserted in the pattern graph as the result of an update. (At this point, the reader may want to investigate the effect of deleting the element *abbc* from D_4 on the pattern graph.)

Pattern graphs and their operations are not only useful in the context of query evaluation and optimization, they can also be employed for optimizing interactive querying sessions. Indeed, the intermediate results of one inductive query can often serve to optimize the computation of the answer to the next query.

As far as the author is aware, pattern graphs - in the form presented above - have not yet been used within the field of data mining. Nevertheless, there exist some interesting data structures, such as FP-trees [24; 40] and version space trees [16] that are typically constructed while evaluating the query and that contain several of the components sketched above. One of the remaining challenges - that we are currently studying for version space trees - is to develop efficient implementations of the pattern set operations.

6.3 Condensed Representations

It is sometimes possible to represent pattern sets using condensed representations. A condensed representation $R(P)$ of a pattern set P is a subset $R(P) \subseteq P$ such that the original pattern set P can be reconstructed from $R(P)$[7]. So, $R(P)$

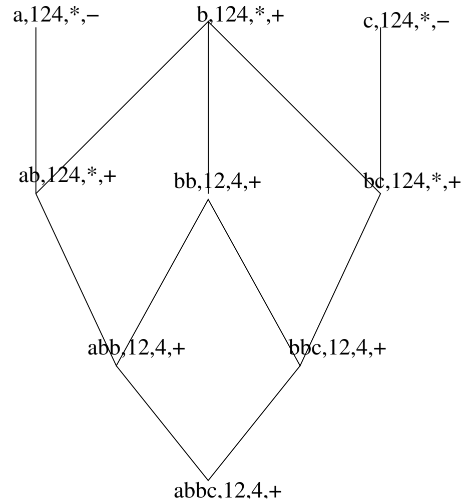

Figure 2: Pattern graph corresponding to P_2.

encodes all relevant information about P in a more compact, i.e. condensed, manner. In the literature one can distinguish two types of condensed representations. On the one hand, there are *local* condensed representations, based on properties of the individual patterns; on the other hand, there are *global* condensed representations based on the properties of the complete pattern set.

Examples of local condensed representations include free item sets, closed item sets, δ-free item sets, etc. See e.g. [8; 39; 41] for more details. An item set I is e.g. free w.r.t. to a data set if and only if there is no rule of the form $i_1 \Rightarrow i_2$ that is valid on the data set where i_1, i_2 are two disjoint subsets of I. Local condensed representations allow one to eliminate redundant patterns from the search space as well as from the solutions. Various algorithms working with local condensed representations have been published in the literature, e.g. [7; 39; 41; 8].

Global representations are often oriented towards representing the border sets, that is the minimal and maximal elements in the set w.r.t. the partial order $\preccurlyeq$. More formally, let $max(P) = \{p \in P \mid \neg\exists q \in P : p \preccurlyeq q\}$, i.e. $max(P)$ contains the maximally specific elements in P, and define $min(P)$ dually, i.e. $min(P) = \{p \in P \mid \neg\exists q \in P : q \preccurlyeq p\}$, i.e. $min(P)$ contains the maximally general or minimally specific elements in P. We can then also define the borders of a pattern set P: $S(P) = max(P)$ and $G(P) = min(P)$[8]. The interesting point about borders is that they can be used as condensed representations. It has been shown [13] that queries q that are a conjunction of anti-monotonic and monotonic constraints are version spaces. This means that they are completely characterized by their sets $S(q)$ and $G(q)$, i.e. $q = \{p \mid \exists s \in S(q), g \in G(q) : g \preccurlyeq p \preccurlyeq s\}$. On our earlier example, the reader may want to verify that the pattern set $P_2 = \{b, ab, bb, bc, abb, bbc, abbc\}$ represented in

[6]Ideally, an operation implementing the complement of a pattern graph would also be provided to compute the answers to a query $\neg q$ on the basis of the pattern graph. However, the resulting pattern graphs would be extremely large (in the case of strings even infinitely large) and therefore impractical. Furthermore, in many cases, as in traditional databases, queries of the form $\neg q$ will not be safe.

[7]Some condensed representations require that additional information (such as exact frequencies) can be reconstructed as well. Other condensed representations require only that the original set (with or without additional information) can be approximated [30].

[8]Sometimes one also uses negative borders [34], they contain patterns not belonging to the pattern set P but are in a sense closest (w.r.t. the generality relation). E.g. the $S^-(P)$ border w.r.t. a minimum frequency threshold contains the maximally general elements that are strictly more specific than an element in $S(P)$.

the pattern graph is completely characterized by $G(P_2) = min(P_2) = \{b\}$ and $S(P_2) = max(P_2) = \{abbc\}$.
Within the field of data mining the use of border sets has been introduced in [34], who propose to keep track of the set $S(P)$ for anti-monotonic constraints, and within machine learning there is the well-known work of [37; 36] on version spaces. In the past few years, a number of algorithms have been published that focuss on the efficient computation of these border sets in the context of data mining, see e.g. [4; 26; 31; 21; 17; 11]. One of the most recent and exciting results is that by [17], who basically show how some local and global condensed representations can be integrated using version spaces.
Whereas our earlier results stated that the solution set of a conjunctive query involving anti-monotonic and monotonic constraints can be represented using a single version space, our more recent results [14; 16] state that the answer set of any boolean query over anti-monotonic and monotonic constraints can be represented as the union of different version spaces. To see why this is the case, rewrite the query in a disjunctive normal form. Each conjunction with the disjunctive normal form will then involve monotonic and anti-monotonic constraints. Hence, the earlier result applies and the conjunction can be represented as a version space, and the original set as the union of such version spaces. This result in turn leads to some interesting questions such as "What is the minimal number of version spaces needed to represent the answers to an inductive query?". This last question is answered in [16]. In the same paper, version space trees are introduced. These combine ideas from pattern graphs and version spaces for the pattern domain of strings. It seems possible to adapt these ideas to FP-trees [24].
Finally, we mention also that operations such as union and intersection on condensed representations have been investigated, cf. [26; 32; 14].

7. CONCLUSIONS

Despite the fact that the presentation of our inductive database framework has been quite informal and presented in the context of string data, there is significant evidence that the line of research sketched in this paper is fruitful from the theoretical as well as from the practical point of view. Much of the present evidence comes from the MolFea system for molecular feature mining, which has effectively been used in a number of real-life applications involving large and complex data sets such as the HIV-data set which contains over 40 000 compounds [31]. Furthermore systems based on the same principles as MolFea have been developed. These include DualMiner by [11] for item sets, ProFea [15] for analyzing the secondary structure of proteins, MineSeqLog [33] for mining logical (i.e. structured) sequences and Version Space Trees [16].
Nevertheless, there are several important limitations of the proposed design. It only addresses local pattern mining tasks[9], does not account for primitives that are neither anti-monotonic nor monotonic (such as e.g. accuracy), ignores probabilistic issues, etc. Despite these limitations the author hopes that the framework inspires some further developments in inductive databases.

[9] [25] distinguish local patterns from global models. Global models are models about a data set as a whole, whereas local patterns are statements about a (local) subset of the data.

Acknowledgements

This work was partly supported by the ESPRIT FET project cInQ. The author is grateful to Heikki Mannila for several suggestions for improvement concerning this paper, to Manfred Jaeger, Stefan Kramer, Sau Dan Lee, and Heikki Mannila for an exciting collaboration on inductive querying, and to the cInQ partners for inspiring discussions, in particular to Jean-Francois Boulicaut, Rosa Meo, Mika Klemettinen, and Pier Luca Lanzi.

8. REFERENCES

[1] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pp. 207-216, 1993.

[2] E. Baralis, G. Psaila. Incremental Refinement of Mining Queries. In Mukesh K. Mohania, A. Min Tjoa (Eds.) *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery* Proceedings. Lecture Notes in Computer Science, Vol.1676, Springer Verlag, pp. 173-182, 1999.

[3] R. Bayardo, (Ed.) Special Issue on Constraint-Based Data Mining, *SIGKDD Explorations*, 2002.

[4] R. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1998.

[5] Jean-Francois Boulicaut, Mika Klemettinen, Heikki Mannila: Querying Inductive Databases: A Case Study on the MINE RULE Operator. In *Proceedings of the 2nd European Conference on Principles and Practice of Knowledge Disocvery in Databases*, Lecture Notes in Computer Science, Vol. 1510, Springer Verlag, pp. 194-202, 1998.

[6] J-F. Boulicaut, M. Klemettinen, H. Mannila. Modeling KDD Processes within the Inductive Database Framework. In *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery*, Lecture Notes in Computer Science, Vol. 1676, Springer Verlag, pp. 293-302, 1999.

[7] Jean Francois Boulicaut, habilitation thesis, INSA-Lyon, France, 2001.

[8] J-F. Boulicaut, A. Bykowski, C. Rigotti. Free-sets: a condensed representation of boolean data for frequency query approximation. *Data Mining and Knowledge Discovery*, Vol. 7(1), 2003.

[9] Toon Calders and Jan Paredaens, Axiomatization of Frequent Sets. In J. van den Bussche, V. Vianu (Eds.), *Proceedings of the 8th International Conference on Database Theory*, Lecture Notes in Computer Science, Vol. 1973, Springer-Verlag, 2001.

[10] D. Braga, A. Campi, A., Ceri, S, Lanzi, P., Klemettinen. Mining Association Rules from XML Data. In *Proceedings of the 4th International Conference on Data Warehousing and Knowlege Discovery*, Lecture Notes in Computer Science, Vol. 2454, Springer-Verlag, 2002.

[11] C. Bucila, J. Gehrke, D. Kifer, W. White. DualMiner: A dual pruning algorithm for item sets with constraints. In *Proc. of the 8th ACM SIGKDD Conference on Knowledge Discovery in Databases*, ACM Press, 2002.

[12] L. Dehaspe, H. Toivonen. Discovery of Frequent Datalog Patterns, in *Data Mining and Knowledge Discovery*, Vol. 3, 1999.

[13] L. De Raedt, S. Kramer, The level wise version space algorithm and its application to molecular fragment finding, in *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 2001.

[14] L. De Raedt. Query evaluation and optimisation in inductive databases using version spaces. In *Proceedings of the DTDM Workshop*, EDBT 2002.

[15] L. De Raedt, S. Kramer. Inductive databases for bio- and chemo-informatics. In Frasconi, P. and Shamir, R. (Eds.) Artificial Intelligence and Heuristic Methods for Bioinformatics, IOS Press (Nato Science Series), 2002. In press.

[16] L. De Raedt, M. Jaeger, S.D. Lee, H. Mannila. A theory of inductive querying. In *Proceedings of the 2nd IEEE Conference on Data Mining*, Mabaeshi, Japan, 2002.

[17] A. Giacometti, D. Laurent and C. Diop. Condensed Representations for Sets of Mining Queries. In *Proc. 1st Workshop on Knowledge Discovery with Inductive Queries* Helsinki, ECML/PKDD 2002.

[18] F. Giannotti, G. Manco: Querying Inductive Databases via Logic-Based User-Defined Aggregates. In *Proceedings of the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases*, Lecture Notes in Artificial Intelligence, Vol. 1704, Springer-Verlag, 1999.

[19] B. Goethals, J. Van den Bussche. On supporting interactive association rule mining. In *Proceedings of the 2nd International Conference on Data Warehousing and Knowlege Discovery*, Lecture Notes in Computer Science, Vol. 1874, Springer Verlag, 2000.

[20] Bart Goethals, Jan Van den Bussche: A priori versus a posteriori filtering of association rules. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1999.

[21] D. Gunopulos, H. Mannila, S. Saluja: Discovering All Most Specific Sentences by Randomized Algorithms. In Foto N. Afrati, Phokion Kolaitis (Eds.): *Proceedings of , 6th International Conference on Database Theory*, Lecture Notes in Computer Science, Vol. 1186, Springer Verlag, 1997.

[22] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane, DMQL: A Data Mining Query Language for Relational Databases, in *Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1996.

[23] J. Han, L. V. S. Lakshmanan, and R. T. Ng, Constraint-Based, Multidimensional Data Mining, *Computer*, Vol. 32(8), pp. 46-50, 1999.

[24] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceeding of ACM SIGMOD Conference on Management of Data*, 2000.

[25] D. Hand, H. Mannila, P. Smyth. Principles of data mining. The MIT Press. 2001.

[26] H. Hirsh. Generalizing Version Spaces. *Machine Learning*, Vol. 17(1): 5-46 (1994).

[27] H. Hirsh. Theoretical underpinnings of versionspaes. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1991.

[28] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.

[29] Tomasz Imielinski and Aashu Virmani. MSQL: A Query Language for Database Mining, *Data Mining and Knowledge Discovery*, Vol. 2(4), pp. 373–408, 1999.

[30] Baptiste Jeudy and Jean-François Boulicaut. Optimization of association rule mining queries, *Intelligent Data Analysis*, Vol. 6 (5), 2002.

[31] S. Kramer, L. De Raedt, C. Helma. Molecular Feature Mining in HIV Data, in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, 2001.

[32] T. Lau, S. Wolfman, P. Domingos, D.S. Weld, Programming by demonstration using versionspace algebra, *Machine Learning*, to appear.

[33] S.D. Lee, L. De Raedt. Mining logical sequences in MineSeqLog. In Dzeroski, S., De Raedt, L., Wrobel, S. (Eds.) *Proceedings of the SIGKDD workshop on Multi-Relational Data Mining*, 2002.

[34] H. Mannila and H. Toivonen, Levelwise search and borders of theories in knowledge discovery, *Data Mining and Knowledge Discovery*, Vol. 1, 1997.

[35] R. Meo, G. Psaila and S. Ceri, An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, Vol. 2 (2), pp. 195-224, 1998.

[36] C. Mellish. The description identification algorithm. *Artificial Intelligence*, Vol. 52 (2), pp,. 151-168, 1990.

[37] T. Mitchell. Generalization as Search, *Artificial Intelligence*, Vol. 18 (2), pp. 203-226, 1980.

[38] R. T. Ng, L. V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of ACM SIGMOD Conference on Management of Data* , 1998.

[39] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25-46, Jan. 1999.

[40] J. Pei, J. Han. Constraint frequent pattern mining: a pattern-growth view. *SIGKDD Explorations*, 4(1), 2002.

[41] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In W. Chen, J.F. Naughton, and P.A. Bernstein, editors, *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Dallas, TX, 2000.

[42] A. K. H. Tung, J. Han, L. V. S. Lakshmanan, and R. T. Ng, Constraint-Based Clustering in Large Databases, in *Proceedings of the 8th International Conference on Database Theory*, Lecture Notes in Computer Science, Vol. 1973, 2001.