Rapid prototyping of pattern mining problems
isomorphic to boolean lattices

Frédéric Flouvat
Université de Lyon
LIRIS, UMR5205 CNRS
F-69621, France
Email: frederic.flouvat@liris.cnrs.fr

Fabien De Marchi
Université de Lyon
LIRIS, UMR5205 CNRS
F-69621, France
Email: fabien.demarchi@liris.cnrs.fr

Jean-Marc Petit
Université de Lyon
LIRIS, UMR5205 CNRS
F-69621, France
Email: jean-marc.petit@liris.cnrs.fr

Abstract—Interesting pattern mining is an important family of data mining problems with applications in many domains. In the last decade, many algorithms and benchmarks have been devised and many experimental studies have been performed for pattern mining problems. In this paper, we focus on the special class of pattern mining problems known to be “representable as sets”. This class of problem is important and have many applications such as in data mining, in databases, in artificial intelligence, or in software engineering. The main contribution of this paper is to take advantage of the common theoretical background of these problems from an implementation point of view by providing efficient data structures for boolean lattice representation and several implementations of well known algorithms. By the way, these problems can be implemented with only minimal effort, i.e. programmers do not have to be aware of low level code, customized data structures and algorithms being available for free. A toolkit, called iZi, has been devised and applied to several problems such as itemset mining, constraint mining in relational databases and query rewriting in data integration systems. According to our first results, the programs obtained using our toolkit offer a very good tradeoff between performances and development simplicity. Some methodological guidelines are also provided to guide the programmers both at the theoretical level and at the code level.

Index Terms—data mining, pattern, toolkit

I. INTRODUCTION

Interesting pattern mining is an important family of data mining problems with applications in many domains. Pattern mining problems encompass the discovery of patterns such as sets, sequences, trees, graphs, and many other structures. The search space being exponential, these problems are quite challenging. In the last decade, many algorithms and benchmarks have been devised and many experimental studies have been performed for pattern mining problems. Given the proliferation of pattern mining implementations, there is a pressing need to understand where the key issues are. As shown in FIMI and OSDM workshops in 2003, 2004 and 2005 [1], [2], [3], best known implementations use quite sophisticated low level code to optimize for instance basic IO routines, which raises interesting issues while being far away from trying to understand why and under what conditions one algorithm would outperform another one.

Moreover, it was advocated that algorithms for pattern mining can be used as a building block for other, more sophisticated data mining problems. This is especially true for pattern mining problems known to be “representable as set” [4], i.e. those problems whose solution space is isomorphic to a boolean lattice. This class of problem is important and have many applications such as in data mining (e.g. frequent itemset mining and variants [5], [6]), in databases (e.g. functional or inclusion dependency inference [7], [8]), in artificial intelligence (e.g. learning monotone boolean function [9]), or in software engineering (e.g. software bug mining [10]). Common characteristics of these problems are: 1) the predicate defining the interestingness criteria is monotone (or anti-monotone) with respect to a partial order over patterns, 2) there exists a bijective function \( f \) from the set of patterns to a boolean lattice and its inverse \( f^{-1} \) is computable, and 3) the partial order among patterns is preserved, i.e. \( X \preceq Y \iff f(X) \subseteq f(Y) \).

In this setting, a common idea is to say that algorithms devised so far should be useful to answer these tasks and available open source implementations are a great source of “know-how”. Unfortunately, it seems rather optimistic to envision the application of most of publicly available implementations of frequent itemset mining algorithms, even for closely related problems. Indeed, sophisticated data structures specially devised for monotone predicates turn out to give very efficient algorithms but limit their application to other data mining tasks. As a consequence, low level implementations hamper the rapid advances in the field. Nevertheless, one may remark that the development of efficient data structures for managing huge sets of sets is definitely useful for data mining.

Paper contribution: This paper takes advantage of the common theoretical background of problems isomorphic to boolean lattices. We provide efficient data structures for boolean lattice representation and several implementation of well known algorithms, such as a levelwise algorithm and a dualization-based algorithm. By the way, any problem can be implemented with only minimal effort, i.e. the programmers do not have to be aware of low level code, customized data structures and algorithms being available for free. A C++ library called iZi has been devised and applied to several problems such as itemset mining, constraint mining in relational databases and query rewriting in data integration.
systems. According to our first results, the programs obtained using the library have very interesting performances regarding simplicity of their development. Some methodological guidelines are also provided to guide the programmers both at the theoretical level and at the code level. The library is freely available on the Web.

**Paper organization:** Section II discusses and illustrates our motivations. Section III discusses the value of our proposition w.r.t. existing related works. In section IV, we introduce the underlined theoretical framework. The section V introduces the iZi library. This section presents some methodological guidelines, points out how state of the art solutions can be exploited in our generic context, and describes the architecture of the iZi library. Experimentations are described in section VI. The last section concludes and gives some perspectives of this work.

II. Motivation

In this section, we give problems apparently quite different but very similar from the theoretical perspective followed in this paper. This (not exhaustive) list aims at proving the potential interest of the iZi library.

**Itemset mining:** Frequent itemsets mining from a transaction database is probably the most famous pattern mining problem. Frequent itemsets are sets of elements that appear in a dataset with a frequency no less than a user-defined threshold. It is trivially representable as sets since the search space is a boolean lattice.

Moreover, the discovery of several condensed representations of frequent itemsets [6] has emerged and turns out to be also pattern mining problems. Their goal is twofold: improve if possible the efficiency of frequent itemsets mining, and compact their storage for future use. Mining these representations leads in general to the conjunction of (anti-) monotone predicates. We can cite as examples frequent free itemsets [11], essential itemsets [12], and more generally frequent patterns with monotone constraint [13].

**Constraint mining in relational databases:** In relational databases, the most important class of constraints are functional and inclusion dependencies, which generalize respectively keys and foreign keys. These constraints convey real-world data semantic and ensure the consistency of any database instance. When such constraints are not available, a natural approach is to recover them from the data themselves, leading to data mining problems [14], [15].

Functional dependency inference [16] and key inference have been shown to be isomorphic to a boolean lattice [4]. In [8], the discovery of inclusion dependencies between two relations also appears to be a pattern mining application isomorphic to a boolean lattice, by posing some restriction on the class of inclusion dependencies to be studied without any loss of knowledge.

Note that these problems have been developed with our library, the data being stored in either a DBMS or in flat files.

**Discovery of schema matching:** In [17] authors propose a process for discovering schema matching among sources, using web query interfaces. The mining part of this process is a levelwise algorithm extracting groups of terms and groups of terms which reach a given threshold. The measure is the min (max) of a positive (negative) correlation between pairs.

Clearly, this part of algorithm can be quickly developed using our library. Moreover, one may notice an interesting point not mentioned by the authors of [17]: any candidate which still survives after the second level is always true. As a consequence, while a pure apriori-like algorithm (as done in [17]) is likely to be intractable whenever large mappings do exist, our library offers the possibility to select the ABS algorithm (described in section V-B) which would terminate after only one iteration.

This application illustrates the interest of our library: for the same problem, several algorithmic strategies can be implemented without more effort than implementing the problem itself.

**Query rewriting in data integration systems:** In [18], authors study query rewriting in presence of value constraints in data integration systems; they identify a sub-problem as a pattern mining problem. The context is a LA V environment [19]. Given a set of views $V$ and a query $Q$, the purpose of query rewriting is to reformulate $Q$ into a query expression that uses only the views in $V$ and is maximally contained in $Q$ [20], [19]. The addition of value constraints enable fine-grained description of queries and views, and can be compared to enumerated data types in databases.

However, the search space being exponential, naïve approaches are intractable and clearly do not scale w.r.t. the number of views. To solve this problem, they suggest to use an approach in the spirit of the "bucket algorithm" [19] to prune the search space. This algorithm is a classical algorithm for query rewriting. Particularly, they focus their attention on the problem of creating bucket since it is one of the most costly step of the rewriting.

One of the salient feature of this work is to point out how this sub-problem can be formulated as a pattern mining problem. Thanks to our library, an external module has been developed and integrated into a query rewriting prototype, allowing the scalability with respect to the number of views.

From our point of view, this is a typical case where our library is very useful, providing a scalable component, almost for free, for the data-centric problems of a larger problem/application.

III. RELATED WORKS

One may notice that algorithm implementations for pattern mining problems are “home-made” programs, see for example implementations available in FIMI workshops [1], [2].
Packages and libraries have also been proposed, for instance Weka [21], Illimine [22], and DMTL [23]. Except DMTL, they do not focus on interesting pattern discovery problems and address several data mining tasks (classification, clustering,...). Moreover, their source codes are not always available.

DMTL (Data Mining Template Library) is a C++ library composed of algorithms and data structures optimized for frequent pattern mining. Different types of frequent patterns (sets, sequences, trees and graphs) using generic algorithms implementations are available. Actually, DMTL supports any types of patterns representable as graphs. Moreover, the data is decoupled of the algorithms, and can be stored in memory, files, Gigabase databases (an embedded object relational database), and PSTL components (a library of persistent containers). This library currently implements an exploration strategy: a depth-first approach (eclat-like [24]). Moreover, some support for breadth-first strategies is also provided. These algorithms could be used to mine all the frequent patterns of a given database.

To our knowledge, only the DMTL library has objectives close to iZi. Even if objectives are relatively similar w.r.t. code reusability and genericity, the motivations are quite different: while DMTL focuses on patterns genericity w.r.t. the frequency criteria only, iZi focuses on a different class of patterns but on a wider class of predicates. Moreover, iZi is based on a well established theoretical framework, whereas DMTL does not rely on such a theoretical background. However, DMTL encompasses problems that cannot be integrated into iZi, for instance frequent sequences or graphs mining since such problems are not isomorphic to a boolean lattice. The iZi library is complementary to DMTL since it offers the following new functionalities:

1) any monotone predicate can be integrated in iZi, while DMTL “only” offers support for the “frequent” predicate;
2) the structure of the patterns does not matter for iZi, while the patterns studied by DMTL must be representable as graphs (e.g., inclusion dependencies cannot be represented in DMTL);
3) while DMTL only gives all frequent patterns, iZi is able to supply different borders of “interesting” patterns (positive and negative borders). These borders are the solutions of many pattern mining problems. Moreover, end-users often do not care about all the patterns and prefer a smaller representation of the solution.

As indication, the table I illustrates some software metrics on DMTL and iZi.

### IV. Theoretical Framework

We recall in this section the theoretical KDD framework defined in [4] for interesting pattern discovery problems.

Given a database $d$, a finite language $\mathcal{L}$ for expressing patterns or defining subgroups of the data, and a predicate $Q$ for evaluating whether a pattern $\varphi \in \mathcal{L}$ is true or “interesting” in $d$, the discovery task is to find the theory of $d$ with respect to $\mathcal{L}$ and $Q$, i.e. the set $Th(\mathcal{L}, d, Q) = \{ \varphi \in \mathcal{L} \mid Q(d, \varphi) \text{ is true} \}$.

Let us suppose a specialization/generalization relation between patterns of $\mathcal{L}$. Such a relation is a partial order $\preceq$ on the patterns of $\mathcal{L}$. We say that $\varphi$ is more general (resp. more specific) than $\theta$, if $\varphi \preceq \theta$ (resp. $\theta \preceq \varphi$).

Let $(I, \preceq)$ be a partially ordered set of elements. A set $S \subseteq I$ is closed downwards (resp. closed upwards) if, for all $X \in S$, all subsets (resp. supersets) of $X$ are also in $S$.

The predicate $Q$ is said to be monotone (resp. anti-monotone) with respect to $\preceq$ if for all $\varphi, \theta \in \mathcal{L}$ such that $\varphi \preceq \theta$, if $Q(d, \varphi)$ is true (resp. false) then $Q(d, \theta)$ is true (resp. false). As a consequence, if the predicate is monotone (resp. anti-monotone), the set $Th(I, d, Q)$ is upward (resp. downward) closed, and can be represented by either of the following sets:

- its positive border, denoted by $Bd^+(Th(I, d, Q))$, made up of the MOST SPECIALIZED true patterns when $Th(I, d, Q)$ is downward closed, and the MOST SPECIALIZED false patterns when $Th(I, d, Q)$ is upward closed;
- its negative border, denoted by $Bd^-(Th(I, d, Q))$, made up of the MOST GENERALIZED false patterns when $Th(I, d, Q)$ is downward closed, and the MOST GENERALIZED true patterns when $Th(I, d, Q)$ is upward closed.

The union of these two borders is called the border of $Th(I, d, Q)$, and is denoted by $Bd(Th(I, d, Q))$.

The last hypothesis of this framework is that the problem must be representable as sets via an isomorphism, i.e. the search space can be represented by a boolean lattice (or subset lattice). Let $(\mathcal{L}, \preceq)$ be the ordered set of all the patterns defined by the language $\mathcal{L}$. Let $E$ be a finite set of elements. The problem is said to be representable as sets if a bijective function $f : (\mathcal{L}, \preceq) \rightarrow (2^E, \subseteq)$ exists and its inverse function $f^{-1}$ is computable, such that:

$$X \preceq Y \iff f(X) \subseteq f(Y)$$

In the sequel, a problem representable as sets will be referred to as “isomorphic to a boolean lattice”.

A salient feature of this latter restriction relies on the notion of dualization [4], [25], well known in combinatorics as minimal transversals of an hypergraph. Let $M$ be a set of patterns from $\mathcal{L}$. Consider the hypergraph $H_M$ containing as edges the sets $f(\varphi)$ for $\varphi \in M$, i.e. $H_M = \{ f(\varphi) | \varphi \in M \}$.
Let $T_r(\mathcal{H}_M)$ be all minimal transversals\(^1\) of $\mathcal{H}_M$. Given $S \subseteq 2^E$, note $\overline{S} = \{ E \setminus X \mid X \in S \}$ the complements of the sets of $S$ in $E$. The relationship between the positive and negative border is given by the following properties.

**Theorem 1:**

- $Bd^+(T_r(\mathcal{H}(L, d, Q))) = f^{-1}(Tr(\mathcal{H}(Bd^-(T_r(\mathcal{H}(L, d, Q)))))) [8]$
- $Bd^-(T_r(\mathcal{H}(L, d, Q))) = f^{-1}(Tr(\mathcal{H}(Bd^+(T_r(\mathcal{H}(L, d, Q)))))) [25]$

V. A GENERIC TOOLKIT FOR PATTERNS DISCOVERY

Based on the theoretical framework presented in section IV, we propose a C++ library, called *iZi*, for this family of problems. The basic idea is to offer a toolkit for a rapid development of efficient and robust programs. The development of this library takes advantage of the past experience to solve particular problems such as frequent itemsets mining, functional dependencies mining, inclusion dependencies mining, and query rewriting.

### A. Methodological guidelines

Once a problem fits into this framework, a program can be rapidly developed using our proposal. However, it is not always easy to ensure that a problem fits into this framework; based on works done by the community, we give in this section some methodological insights, illustrated through two running examples which are key and foreign key discovery in databases.

**What kind of problems?** Problems have to be enumeration problems under constraints, i.e. of the form “enumerate all the patterns that satisfy a condition”. When the condition must be verified in a data set, the word “enumerate” is commonly replaced by “extract”. Frequently, the problem specification requires that patterns must be maximal or minimal w.r.t. some natural order over patterns.

#### Example 1:

1) Let us consider the key discovery problem in a relation, which can be stated as follows:

**Key mining problem** (referred to as $KEY$): Let $r$ be a relation over a schema $R$, extract (minimal) keys satisfied in $r$. Let $KEY(r) = \{ X \subseteq R \mid X \text{ key in } r \}$

1A minimal transversal of an hypergraph $H$ is a set of elements $X$ such that (1) $X$ has an non empty intersection with every element of $H$ and (2) $X$ is minimal w.r.t. this property.

2) An inclusion dependency (IND) is an expression of the form $R[X] \subseteq S[Y]$, where $R$ and $S$ are relation schemas of a same database schema $D$. Such a constraint ensures that, for any relations $r$ and $s$ over $R$ and $S$, any $X$-value into $r$ is a $Y$-value into $s$. If $Y$ is a key in $S$, then $X$ is a foreign key in $R$. Inclusion dependency discovery is a way to discover foreign keys and other more general semantic constraints. It can be stated as follows:

**IND mining problem** (referred to as $IND$): Let $d$ be a database over a schema $D$, extract (maximal) inclusion dependencies satisfied in $d$. Let $IND(d) = \{ R[X] \subseteq S[Y] \mid R, S \in D, R[X] \subseteq S[Y] \text{ is satisfied in } d \}$

#### Language, predicate and monotonicity:

Once a problem is suspected to fit into the framework, some components must be properly defined to go further.

a. Defining the patterns: what is the search space? It implies to define a language or a formal characterization of patterns to be mined.

b. Pointing out the predicate with a mathematical sentence as: A pattern $X$ is interesting if $C(X, d)$ is true, where $C$ is a condition expressed over dataset $d$.

c. Defining a partial order among patterns. Frequently, when $X$ is more general than $X'$, i.e. $X' \subseteq X$, the syntax of $X$ is a restriction on the syntax of $X'$.

These specifications are necessary to prove predicate monotonicity. The question is: if a pattern $X$ is true (or false), does another more general (resp. specific) pattern always true (resp. false)? Note that if the problem is to search for maximal or minimal patterns satisfying some properties, then partial order and monotonicity should be straightforward to point out.

#### Example 2:

1) **Key mining problem**

   a. The pattern language is $L_{key} = \{ X \mid X \subseteq R \} = \mathcal{P}(R)$.

   b. The predicate $P_{key}(X, r)$ is true if $X$ is a superkey, i.e. if $|\pi_X(r)| = |r|$, where $\pi_X(r)$ is the projection onto $X$ over $r$.

   c. The partial order is $\subseteq$.

   It is clear that any superset of a superkey is also a superkey, i.e. only minimal keys are required.

   **Property 1:** The predicate $P_{key}(X, r)$ is monotone w.r.t. the superkey.

   **Theorem 2:** $KEY(r) = Bd^{-}(Th(L_{key}, r, P_{key}))$

2) **IND mining problem**

   a. The pattern language $L_{ind}$ is composed of all the IND expressions that can be expressed into a database schema.

   b. The predicate $P_{ind}(R[X] \subseteq S[Y], d)$ is true, if $\pi_X(r) \subseteq \pi_Y(s)$.

   c. From a well known inference rule for INDs [26], if an IND is satisfied, then any IND obtained
by applying the same projection on the left and right-hand sides is satisfied. As an example, if \( R[AB] \subseteq S[EF] \) is satisfied, then the following INDs (not exhaustive) are satisfied: \( R[A] \subseteq S[E], R[B] \subseteq S[F], R[C] \subseteq S[G], R[BC] \subseteq S[FG] \). Consequently, the partial order is defined by projections over INDs.

**Property 2:** [8] Considering the partial order defined by projections over INDs, the predicate \( P_{IND}(R[X] \subseteq S[Y], d) \) is anti-monotone.

The IND mining problem can be reformulated as follows [8]:

**Theorem 3:** \( IND(d) = Bd^+(Th(L_{ind}, d, P_{ind})) \)

These requirements are sufficient to exploit a pure levelwise algorithm. But a consequent work has to be performed, in general, to ensure a sound and scalable candidate generation at iteration \( i + 1 \) from interesting patterns at iteration \( i \).

**Isomorphism to boolean lattice:** For any problem representable as sets, all the work done for itemset mining can be exploited. Although such a representation is obvious in some cases, it requires more careful study in other cases.

**Example 3:**

1. Key mining problem - The search space is already a boolean lattice, i.e. the function \( f \) is the identity.
2. IND mining problem (see [8] for more details on this point) - The search space is not a boolean lattice at all. As an example, consider the two INDs \( R[X] \subseteq S[Y] \) and \( R[X'] \subseteq T[Z] \): they cannot be projections of a same IND, so they do not have upper bound. To solve this, we have to consider the subproblems \( IND(r, s) \) for each \( \{r,s\} \) in \( d \). However, the search spaces of these subproblems are still not boolean lattices. For example \( R[A] \subseteq S[E] \) and \( R[B] \subseteq S[F] \) have two possible least upper bound, which are \( R[AB] \subseteq S[EF] \) and \( R[BA] \subseteq S[FE] \). In order to fit each subproblem into a boolean lattice context, we define the function \( f \) which transform any IND into the set of all unary INDs (i.e. INDs between single attributes) obtained by projection. Thus, \( f(R[AB]) \subseteq S[EF] \rightarrow \{ R[A] \subseteq S[E]; R[B] \subseteq S[F] \} \).

- \( f \) is a one-to-one function, since \( f(R[AB]) \subseteq S[EF] \) is satisfied. Consequently, the partial order is defined by projections over INDs.

- \( f \) is not surjective, since e.g. \( f^{-1}(R[A] \subseteq S[E]; R[B] \subseteq T[G]) \) cannot be defined. To cope with this problem, one needs to mine INDs from pairs of relations one by one. Moreover, duplicate attributes must be allowed in IND definition as it is done in [27].

With the above restrictions, one can easily verify that \( f \) is an isomorphism between IND search space and the powerset of unary INDs.

The search space \( C \) of INDs over \( (R, S) \) is defined by:

\[
C(R, S) = \{ R[< A_1 ... A_n >] \subseteq S[< B_1 ... B_n >] \mid \forall 1 \leq i < j \leq n, (A_i < A_j) \lor (A_i = A_j \land B_i < B_j) \}
\]

where \( n = \min(|R|, |S|) \).

Let \( I_1 \) be the set of unary INDs over \( R \). The function \( f : C \rightarrow P(I_1) \) is defined by: \( f(i) = \{ j \in I_1 \mid j \leq i \} \).

The function \( f : C \rightarrow P(I_1) \) is bijective and its inverse function \( f^{-1} \) is computable. Moreover, given \( i \) and \( j \) two IND expressions of \( C \), \( i \leq j \Leftrightarrow f(i) \subseteq f(j) \).

Consequently, \( f \) is an isomorphism from \( (C, \leq) \) to \( (P(I_1), \subseteq) \), that is to say that the search space of INDs is representable as sets.

**Theorem 4:** [8] Let \( L_{ind} = C(R, S) \), the search space of \( IND(r, s) \) is isomorphic to a boolean lattice, and the function \( f : C \rightarrow P(I_1) \).

This example is a typical case: the problem becomes representable as sets by restricting the language to be used to define the search space (without any loss of knowledge thanks to patterns properties).

B. Generic algorithms and data structures

The classical way to solve pattern mining problems is to develop ad-hoc solutions from scratch, with specialized data structures and optimization techniques. If such a solution leads to efficient programs in general, it requires a huge amount of work to obtain a sound and operational program. Moreover, if problem specifications slightly differ over time, a consequent effort should be made to identify what parts of the program should be updated.

One of our goal is to factorize some technical solutions which can be common to any pattern mining problem representable as sets. We are interested in algorithms and data structures that apply directly on sets, since they can be used without any change for any problem, thanks to the isomorphism.

Our solution reuses previous works devoted to frequent itemset mining (FIM), which is the most studied pattern mining problem and “directly” representable as sets.

**Algorithms:** Currently, many algorithms from the multitude that has been proposed for the FIM problem could be implemented into iZI, from well knowns Apriori algorithm [28] or depth-first approaches, to more sophisticated dualization-based algorithms (Dualize and Advance [25] or ABS [29]).

However, some algorithms don’t fit into this framework because they are not based on a clear distinction between the
exploration strategy and the problem. For example, FP-growth like algorithms [30] cannot be used into this framework since their strategy is based on a data structure specially devised for FIM. In the same way, condensed representations based algorithms like LCM [31] cannot be applied to any pattern mining problem representable as sets.

The need to have multiple strategies integrated in our library is twofold. First, note that the type of solution discovered by each algorithm is specific. For example, the Apriori algorithm discovers (without any overhead) the theory and the two borders, whereas dualization-based algorithms “only” discovers the two borders. Since depending on the studied problem, we might be interested in either the theory, or the positive border, or the negative border, it is necessary to have multiple strategies to enable the discovery of the required solution. Secondly, as shown by the FIMI workshops, the algorithms performance depend on dataset/problem characteristics. For example, Apriori algorithm is more appropriate when the theory is composed of relatively small elements. Consequently, different algorithms must be integrated into the library to have the best performances according to problem properties. The strategies proposed in our library had been chosen w.r.t. these points, and their specificities are provided in its documentation.

Data structures: Since the generic part of our library only manipulates sets, we use a data structure based on prefix-tree (or trie) specially devoted to this purpose [32] (see figure 2 for an example).

They have not only a power of compression by factorizing common prefix in a set collection, but are also very efficient for candidate generation. Moreover, prefix-trees are well adapted for inclusion and intersection tests, which are basic operations when considering sets. Of course, as for algorithms, one can imagine to extend our library with alternative structures for sets, like bitmaps. The use of indexes is also an important issue but not considered yet.

Note that template trie container and iterator are provided with the library. Actually, two trie implementations are available with the library: one optimized for data compression and one optimized for data search. Their implementation have been mapped on the implementations of the standard STL (Standard Template Library) containers. This class also contains an implementation of an incremental algorithm, based on trie data structures, for the minimal transversals computation of an hypergraph.

C. Architecture

The figure 3 represents the architecture and the “workflow” of our library: The algorithm is initialized (initialization component) with patterns corresponding to singletons in the set representation, using the data (data access component). Then, during the execution of the algorithm, the predicate is used to test each pattern against the data. Before testing an element, the algorithm use the set transformation component to transform each set generated into the corresponding pattern.

This architecture is directly derived from the studied framework and has the main advantage of decoupling algorithms, patterns and data. Only the predicate, set transformation and initialization components are specifics to a given problem. Consequently, to solve a new problem, users may have to implement or reuse with light modifications some of these components.

The algorithm component represents generic algorithm implementations provided with the library and used to solve pattern mining problems. As shown in figure 3, algorithms are decoupled of the problems and are black box for users. Each algorithm can be used directly to solve any problem fitting in the framework without modifications. This leads to the rapid construction of robust programs without having to deal with low level details. Currently, the library offers a levelwise algorithm [4], a dualization-based algorithm (a generic version of $ABS$ [29]), and two other variants of these algorithms. These variants globally have the same strategy but explore the search space in a different way (top-down exploration instead of bottom-up, see figure 4 for the two Apriori variants) which is more appropriate for some predicates. Finally, depth-first strategies are also currently being integrated.
Another important aspect of our library is that data access is totally decoupled of all other components (see figure 3). Currently, data access in most of the implementations is tightly coupled with algorithm implementations and predicates. Consequently, algorithms and “problem” components can be used with different data formats without modifications.

Note that the patterns do not directly appear in the architecture, but are omnipresent in the predicate and set transformation components. Consequently, patterns are totally independent of the algorithms and the data.

The development of the library has been inspired by the C++ STL (Standard Template Library). For example, the predicate, the set transformation, the initialization and the algorithm classes are implemented as functors (i.e. function objects) based on template parameters (e.g. template containers or iterators).

The figure 6 presents, from an implementation point of view, an UML model of the library. In particular, this model specifies how patterns and sets interact with the other components: patterns are used in problem specific components and sets are used internally by the algorithms. This model also points out the possibility to do predicate composition which is the case in many applications (e.g. itemset mining using conjunction of monotone constraints). For data access, this model distinguishes two cases: input data and output data. Input data is used by the predicate to test patterns and is totally independent of the algorithms. Output data is used by the algorithm to output the solutions (theory and/or positive border and/or negative border).

Moreover, thanks to this model and to the object-oriented paradigm, user can also do algorithm and predicate variants/refinements, i.e. use inheritance to define new algorithms or predicate based on existing ones. The figure 7 presents an example of algorithm and predicate variants/refinements already implemented. In this figure, the frequent class represents the frequent itemset mining predicate, and the frequent essential class represents the predicate for a condensed representation of frequent itemsets. In the same way, the ABS class represents the ABS algorithm provided in iZi and the ABSReverse class represents a variant of this algorithm changing the exploration strategy.

In our context, another interesting property is method overloading which can be used to optimized some predicates sub-methods w.r.t. specific data structures. For example, the frequent itemset predicate uses a counting method to find whether an itemset is frequent or not, and this sub-method is crucial for the algorithm performances. Using method overloading, it
is for example possible to have a generic count method and one optimized for trie data structures. Thanks to this property, it is possible to have a good compromise between components genericity and algorithms performances.

Finally, to solve a new problem, users only have few implementations constraints. For example, their predicate and set transformation classes have to be functors with the same signature for the operator() method, and output data classes must only have a push_back() method. To facilitate these developments, abstract base classes are provided with the library as well as sample components.

VI. EXPERIMENTATIONS

From our past experience in the development of pattern mining algorithms, we note that the adaptation of existing implementations is extremely difficult. In some cases, it implies the redevelopment of most of the implementation and could take more time than developing a new program from scratch. Many problems have been implemented in our library along with several components. As indication, the use of our library to implement a program for the key mining problem has been done in less than one working day.

In the same way, the library provides data access components to the user for implementing her/his problem, thus allowing to directly access several data types and data sources.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Problem</th>
<th>Data format</th>
<th>DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabular</td>
<td>inclusion dependencies</td>
<td>CSV, FDEP</td>
<td>MySQL</td>
</tr>
<tr>
<td>tabular</td>
<td>keys</td>
<td>CSV, FDEP</td>
<td>MySQL</td>
</tr>
<tr>
<td>binary</td>
<td>frequent itemsets</td>
<td>FIMI</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>frequent essential itemsets</td>
<td>FIMI</td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>query rewriting</td>
<td>specific</td>
<td></td>
</tr>
</tbody>
</table>

TABLE II PROBLEMS, DATA TYPES AND FORMATS EXPERIMENTED WITH iZI

The table II shows the different problems, data types and sources experimented with iZi. Our library was tested against 5 problems, and dealt with 5 different formats. For itemset mining in transactional databases, the format considered is the FIMI file format which has been defined for the FIMI workshops [1], [2]. This data format is widely used for this family of problems. For constraint mining in relational databases, components have been also developed to access data in files (CSV format of Excel and FDEP format [33]) and in the MySQL DBMS. For query rewriting in integration systems, we have studied two combinatorial sub-problems (i.e. two different predicates). The data access and output components processed specific file formats.

**Performances** : Our motivation here is to show that our generic library has good performances w.r.t. specialized and optimized implementations.

We present some experimental results for frequent itemset mining, since it is the original application domain of the algorithms we used and the only common problem with DMTL. Moreover, many resources (algorithms implementations, datasets, benchmarks...) are available on Internet [34] for frequent itemset mining. For other problems such as key mining, even if algorithms implementations are sometimes available, it is difficult to have access to the datasets. As an example, we plan to compare iZi with the proposal in [35] for key mining. Unfortunately, neither their implementation, nor their datasets have been made available in time.

Implementations for frequent itemset mining are very optimized, specialized, and consequently very competitive. The most performant ones are often the results of many years of research and development. In this context, our experimentations aims at proving that our generic algorithms implementations behave well compared to specialized ones. Moreover, we compare iZi to the DTML library, which is also optimized for frequent pattern mining.

Experiments have been done on some FIMI datasets [1], [2]. We compared our Apriori generic implementation to two others devoted implementations: one by B. Goethals [36] and one by C. Borgelt [37]. The first one is a quite natural version, while the second one is, to our knowledge, the best existing Apriori implementation, developed in C and strongly optimized. Then, we compared “iZi Apriori and ABS” to the eclat implementation provided with DMTL.

The figure 8 shows execution times for datasets Connect (129 items and 67557 transactions), Pumsb (2113 items and 49046 transactions) and Pumsb* (2088 items and 49046 transactions). One can observe that our generic version has good performances with respect to other implementations. These results are very encouraging, in regards of the simplicity to obtain an operational program.
The figure 9 shows execution times for the same datasets. Even if DMTL is optimized and specialized for the frequent predicate, algorithm implementations of iZi have good performances w.r.t. eclat DMTL. The difference between the two libraries is mainly due to the algorithm used during the experimental. This could be easily confirmed by looking at the performances of Apriori, Eclat and ABS algorithms observed during FIMI benchmarks [34].

VII. DISCUSSION AND PERSPECTIVES

In this paper, we have considered a classical problem in data mining: the discovery of interesting patterns for problems known to be representable as sets, i.e. isomorphic to a boolean lattice. As far as we known, this is the first contribution trying to bridge the gap between fundamental studies in data mining around inductive databases [4], [38], [25] and practical aspects of pattern mining discovery. Our work concerns plenty of applications from different areas such as databases, data mining, or machine learning.

Many perspectives exist for this work. First, we may try to integrate the notion of closure which appears under different flavors in many problems. The basic research around concept lattices [39] could be a unifying framework. Secondly, we are interested in integrating the library as a plugin for a data mining software such as Weka [21]. Analysts could directly use the algorithms to solve already implemented problems or new problems by dynamically loading their own components.

Finally, a natural perspective of this work is to develop a declarative version for such mining problems using query optimization techniques developed in databases [40].

REFERENCES


