Towards easy prototyping of pattern mining problems

Frédéric Flouvat[1], Fabien De Marchi[2], and Jean-Marc Petit[1]

[1] LIRIS, INSA-Lyon, F-69621, France
frederic.flouvat@liris.cnrs.fr
jean-marc.petit@liris.cnrs.fr
[2] LIRIS, Université Lyon I, F-69621, France
fabien.demarchi@liris.cnrs.fr

Abstract. In the last decade, plenty of algorithms, benchmarks, and experimental studies have been carried out for pattern mining problems. In this paper, we focus on the special class of pattern mining problems known to be "representable as sets". In this setting, 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 proposing a library of efficient data structures and algorithms for pattern mining. Thus, every problem fulfilling the theoritical requirements could be implemented with only minimal effort. According to our first results, the programs obtained using our library offer a very good tradeoff between performances and simplicity of their development.

Keywords : pattern mining, inductive databases, boolean lattices, C++ library.

1 Introduction

In the last decade, plenty of algorithms, benchmarks, and experimental studies have been carried out for pattern mining problems. Pattern mining problems encompass the discovery of patterns such as itemsets, sequences, trees, graphs, and many other structures. Varied approaches to these problems appear in numerous papers, e.g. [1–4].

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 [5–7], best known implementations use quite sophisticated low level code to optimize for instance basic IO routines, which raise interesting issues while being far away from trying to understand why and under what conditions one algorithm would outperform another.
Morevover, it was advocated that algorithms for frequent itemset 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" [8], as for instance, frequent itemset mining and variants [1, 9, 10], functional

or inclusion dependency inference [11–13] or learning monotone boolean function [14]. Roughly speaking, the class of problems we are interested in and referred to as "isomorphic to boolean lattices" in this paper, consists of pattern mining problems for which: 1) the predicate defining the interestingness criteria has to be monotone with respect to the partial order of patterns, 2) there exists a bijective function f between the set of patterns and some finite set, 3) the partial order among patterns is preserved, i.e. $X \preceq Y \Leftrightarrow f(X) \subseteq f(Y)$.

In this setting, the basic idea is to say that algorithms devised so far should be useful to answer these tasks and available open source implementations 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, even if the development of efficient data structures for managing huge sets of set is definitely useful, loading the database in main memory using sophisticated data structure specially devised for a monotone predicate turns out to give very efficient algorithms but deserve other data mining tasks. As a consequence, low level implementations hamper the rapid advances in the field.

In this paper, we focus on the special class of pattern mining problems known to be "representable as sets" [8]. Our objective is twofold:

- to make easier the development of data mining implementations by offering in a black box state of the art data structures and algorithms for free.
- to guide the developers in the key stages of the design of his/her implementation.

Thus, according to the problem characteristics, she/he will have to focus only on the core strategy to be used to implement a scalable implementation, without worrying about technical details useless for prototyping.

Paper contribution In this setting, 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 implementation of well known algorithms such as levelwise algorithm and dualization-based algorithm. By the way, every problem fulfilling these requirements could 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 library, called *iZi*[3], has been devised and applied to several problems such as itemset mining or constraint mining in relational database. According to our first results, the programs obtained using the library have very interesting performances in comparison with the simplicity of their development. Some methodological guidelines are also provided to guide the programmers both at the theoretical level and at the C++ code level. The library is freely available on the Web [15].

[3] derived from the phonetic of the word "easy": [ˈiːzɪ]

Paper organization In section 2, we introduce the theoretical framework used (subsection 2.1) and guidelines to use it (subsection 2.2). This section also discusses how state of the art solutions can be used in our generic context (subsection 2.3). The section 3 presents our contribution: the *iZi* library. The architecture of the library is given in subsection 3.1 and is followed by a quick start guide (subsection 3.2). Experimentations are described in section 4. Section 5 discusses the interest of our proposition w.r.t. to related works. The last section concludes and gives some perspectives to this work.

2 Framework

2.1 Theoritical components

We recall in this section the theoretical KDD framework defined in [8] for interesting patterns 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 φ is more general (resp. more specific) than θ, if $\varphi \preceq \theta$ (resp. $\theta \preceq \varphi$).

The predicate Q is said to be monotone (resp. anti-monotone) with respect to $\preceq$ if for all $\theta, \varphi \in \mathcal{L}$ 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 $Th(\mathcal{L}, d, Q)$ is upward (resp. downward) closed, and can be represented by any of the following set:

- its *positive border*, denoted by $\mathcal{B}d^+(\ Th(\mathcal{L}, d, Q)\ )$, made up of the most specialized TRUE patterns when $Th(\mathcal{L}, d, Q)$ is downward closed, and the most generalized TRUE patterns when $Th(\mathcal{L}, d, Q)$ is upward closed;
- its *negative border*, denoted by $\mathcal{B}d^-(\ Th(\mathcal{L}, d, Q)\ )$, made up of the most generalized FALSE patterns when $Th(\mathcal{L}, d, Q)$ is downward closed, and the most specialized FALSE patterns when $Th(\mathcal{L}, d, Q)$ is upward closed.

The union of this two borders is called the *border of* $Th(\mathcal{L}, d, Q)$, and is denoted by $\mathcal{B}d(\ Th(\mathcal{L}, d, Q)\ )$. The search space can be respresented by a lattice (see figure 1) composed of all the patterns of the language $\mathcal{L}$ and ordered w.r.t. the partial order $\preceq$. Then, the border caracterizes this search space. If the predicate is monotone (resp. anti-monotone), all the patterns "above" this border are true (resp. false), and all the patterns "below" are false (resp. true).

Note that when an interestingness measure is associated to patterns of $Th(\mathcal{L}, d, Q)$, the borders may not be sufficient for recovering all TRUE patterns together with their value. It is the case for the frequency measure defined for itemsets in a transaction database.

Last hypothesis of this framework, the problem must be representable as sets via an isomorphism, i.e. the search space can be represented by a boolean lattice.

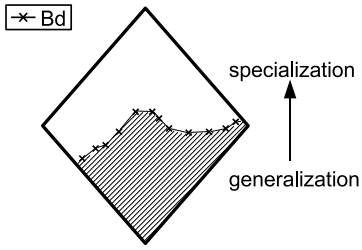

Fig. 1. Border and search space

Let $(\mathcal{L}, \preceq)$ be the ordered set of all the patterns defined by the langage $\mathcal{L}$. Let I be a finite set of items. The problem is said to be *representable as sets* if an isomorphism exists from $(\mathcal{L}, \preceq)$ to $(2^I, \subseteq)$, such that:

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

2.2 Guidelines

Once a problem fits in this framework, a program can be rapidly developped using our proposal. However, it is not always trivial to ensure this fitting; from our experience [16, 11, 17, 18], we give some methodological insights in this section, illustrated through an example which is key discovery in databases.

What kind of problems ? Problems have to be *enumeration problem under constraints*, i.e. of the form "enumerate all the patterns that verify a condition". When the condition must be verified in a data set, the term "enumerate" is commonly replaced by "extract". Frequently, the problem specification precises that patterns must be maximal or minimal: maybe only the positive of negative border is wondered.

Langage, predicate and monotony Once a problem is suspected to fit into the framework, its components must be properly defined to go further.

1. Precising the patterns: what is the search space ? it implies to define a langage or a formal characterization of what patterns are mining for.
2. Pointing out the predicate with a mathematical sentence as: A pattern X is true if $C(d, X)$, where C is a condition expressed over data.

These specifications are helpful to demonstrate the monotonicity of the predicate. The question is: if a pattern X is true (or false), does another more specific (i.e. less simple) pattern always true (resp.false) ? Note that if the problem is to search for maximal or minimal patterns verifying some properties, then the predicate monotonicity is almost sure. Given the sens of the order, it might be monotone or anti-monotone.

Isomorphism to boolean lattice Sometimes, the isomorphism is trivial, as it is the case for frequent itemsets problems: the set of candidates is naturally expressed as a powerset. In other cases, a more deep study must be performed. An important remark is that a necessary condition is that the number of candidates, when executing the program, must

As an example, we consider the problem of discovering the keys of a relation [19].

Example 1:
What kind of problems ?
Let us consider the key discovery problem in a relation, which can be enounced as follows: Let r be a relation over a schema R, extract the (minimal) keys satisfied in r.
Langage, predicate and monotony
The patterns are : $\{X \mid X \subseteq R\} = \mathcal{P}(R)$. X 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. It is clear that any superset of a superkey is also a superkey, it justifyes that only minimal keys are really interested. One can deduce that minimal keys constitute the positive border of superkeys, with natural set inclusion.
Isomorphism to boolean lattice
The transformation function here is the identity since patterns are sets.

2.3 Generic Algorithms and data structures

Even if this framework has been frequently used at a theoretical level, it has never been exploited at a technical point of view. The classical way used to solve pattern mining problems is to develop had-hoc solutions, with very specialized data structures and optimization technics. If such a solution leads to efficient programs in general, it requires a huge amount of work to obtain a sound and operational tool. Moreover, if problem specifications slightly differ over time, a consequent effort will be made to identify what parts of the program must 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 applied without any change for any problem, exploiting the isomorphic transformation.

Our solution reuse some previous works done for frequent itemset mining (FIM), which is a problem "directly" representable as sets, i.e. the isomorphic transformation is the identity.

Algorithms Currently, our library implements two algorithms from the multitude that has been proposed for the FIM problem. The first is the well known *Apriori* algorithm [20], which is still in course for discovering the whole theory for *not too large positive borders* [21]. The second one is *ABS* [22], well adapted to discover any positive border when the negative border is not too large. We argue these choices with the following consideration: they are really generic in our framework,

since they only rely on set operations and predicate evaluations. They do not exploit any specific property of the predicate, as it is the case for FP-growth like algorithms [23] or contributions using condensed representations [24].

Note that other solutions like a pure depth-first approach are also generic and potentially interested for some problems. We plan to offer such a strategy in future release of the library.

Data structures Since the generic part of our library only manipulates sets, we use a data structure based on prefix-tree specially devoted to this purpose [25, 26]. They have not only a power of compression by factorizing common prefix in a set collection, but are also 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.

3 A generic library for patterns discovery

Based on the theoretical framework presented in section 2, we propose a C++ library, called *iZi*, for this family of problems. The idea which guides this work is to offer a toolbox for a rapid development of an efficient and robust program, using a selected algorithm. Note that the development and structure of the proposed library result from our experience when implementing and adapting algorithms to solve these problems.

3.1 Architecture

The figure 2 represents the architecture and the "workflow" of our library.

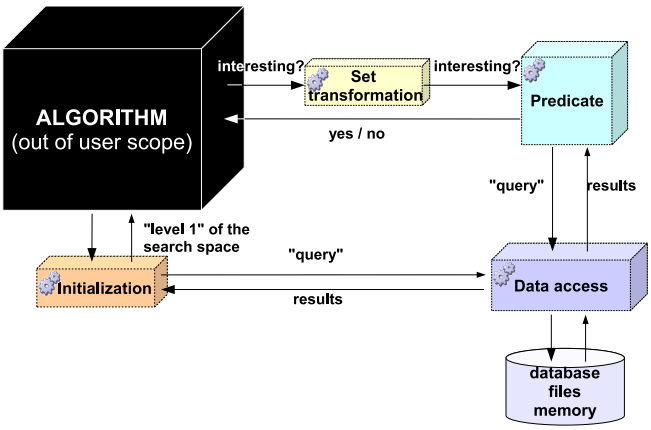

Fig. 2. *iZi* architecture

As shown by the figure 2, the library is composed of 5 types of components: *algorithm*, *set transformation* function, *predicate* function, *data access* component and *initialization* function.

The *algorithm* component represents generic algorithms implementations provided in the library and used to solve users pattern mining problems. As shown by the figure 2, algorithms are decoupled of the problems and are black box for the users. Each algorithm can be used directly to solve any problem fitting in the studied framework without modifications. This leads to the rapid construction of robust programs for the user without having to implement complex algorithms. The other components are specifics to a given problem. Consequently, to solve a new problem, users may have to implement some of these components. However, a user can also reuse existing components developed for other problems.

Another important aspect of our library is that data access is totally decoupled of all the other components (see figure 2). Currently, data access in most of the implementations is tightly coupled with the algorithm implementation and the problem which limits their use.

The functioning of the library is presented in the figure 2. First, the *algorithm* is initialized (*initialization* function) with the patterns corresponding to singletons in the set representation, eventually by accessing data (*data access* component). Then, during the execution of the algorithm, the predicate is used to test each pattern using the *data*. Recall that as defined in the framework, algorithms of the library only manipulate set elements. Consequently, before testing an element, the algorithm may have to use a *set transformation* function to transform each set generated into the corresponding pattern.

3.2 Getting started with *iZi*

To install *iZi*, the user only has to download the files from our website [15] and to include the header files in her/his source code. Then, she/he can directly use the library.

Before implementing any components, the user has to verify that her/his problem is not already provided by the library. If the problem is already implemented, the *Predicate* and *Initialization* components (figure 2) can be used without modifications, otherwise they must be redeveloped. However, if the problem is close to a problem provided with the library, the related components can be used as examples or derived to process the new problem. This alternative has in particular been tested when implementing frequent itemsets mining and frequent essential itemsets mining problems.

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.

The table 1 shows the differents problems, data types and sources provided with *iZi*. The library already contains 4 problems, and 4 data access types and sources. For itemset mining in transactional databases, the format considered is the FIMI file format which was defined for the FIMI workshops [5, 6], and is

Data type	Problem	File (format)	DBMS
tabular	inclusion dependencies	CVS FDEP	MySQL
tabular	keys	CVS FDEP	MySQL
binary	frequent itemsets	FIMI	
binary	frequent essential itemsets	FIMI	

Table 1. Problems, data types and formats provided with *iZi*

the most comonly used data format for this family of problems. For constraint mining in relational databases, components were also developed to access data in files (CSV format of *Excel* and *FDEP* format [27]) and in the *MySQL* DBMS.

Once the problem and data access components are defined, the analyst must choose a strategy to explore the search space. Currently, the library provides 4 search space strategies:

- two bottom-up algorithms (figure 3): an *Apriori*-like [20, 8] and *ABS*.
- two top-down algorithms (figure 4): an inverted version of *Apriori* and inverted version of *ABS*.

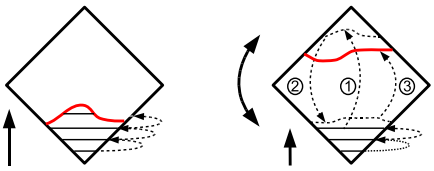

Fig. 3. The bottom-up algorithms: *Apriori* (left) and *ABS* (right)

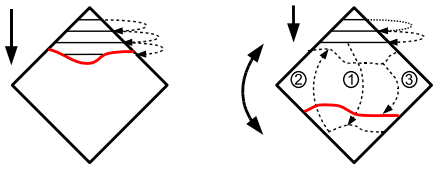

Fig. 4. The top-down algorithms: inverted *Apriori* (left) and inverted *ABS* (right)

First, the user has to identify algorithms "compatibles" with her/his predicate properties and the solution she/he wants to discover. Note that from a theoretical point of view, the positive (resp. negative) border of a predicate is the negative (resp. positive) border of the inverted predicate, i.e. $\mathcal{B}d^+(Th(\mathcal{L}, d, Q)) = \mathcal{B}d^-(Th(\mathcal{L}, d, \neg Q))$ and $\mathcal{B}d^-(Th(\mathcal{L}, d, Q)) = \mathcal{B}d^+(Th(\mathcal{L}, d, \neg Q))$. Consequently by inverting a monotone (resp. anti-monotone) predicate Q, bottom-up (resp. top-down) approaches could be used to mine the borders of this predicate. For this reason, the library provides a generic component to invert predicates.

The table 2 presents all these possibilities.

Secondly, the analyst may have to choose an algorithm among all the algorithms "compatibles" with her/his problem specificity. To do this choice, the analyst could use her/his own knowledge about the domain or the data.
This is typically the case when choosing between a bottom-up or a top-down

	monotone predicate Q	anti-monotone predicate Q	inverted monotone predicate $\neg Q$	inverted anti-monotone predicate $\neg Q$
Th	inverted *Apriori*	*Apriori*		
$\mathcal{B}d^+$	inverted *Apriori* inverted *ABS*	*Apriori* *ABS*	*Apriori* *ABS*	inverted *Apriori* inverted *ABS*
$\mathcal{B}d^-$	inverted *Apriori* inverted *ABS*	*ABS* *ABS*	*Apriori* *ABS*	inverted *ABS* inverted *ABS*

Table 2. Algorithms w.r.t. solution and predicate

strategy. As shown by the figure 5, the efficiency of each strategy is influenced by the "position" of the solutions in the search space.

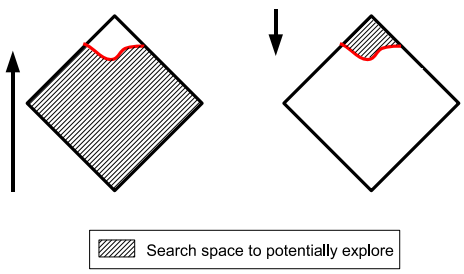

Fig. 5. Search space explored by bottom-up and top-down strategies w.r.t. solutions

The choice between an *Apriori*-like approach or an *ABS*-like approach could also be done based on expert knowledge. Actually, as shown in [21, 22], *Apriori* is more particularly efficient when interesting patterns are "low" in the search space, and *ABS* when pattens of the negative border are "far" from patterns of the positive border.

4 Experimentations

Implementation of a new problem It is a very hard task to evaluate the difficulty for a user to develop a program with our library.
From our past experience in the development of pattern mining algorithms [22, 18, 28], we noted that the adaptation of existing implementations was 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.
As shown in the previous section, four problems have been implemented in our library to test the difficulty of implementing a new predicate or changing the data access. As indication, the use of our library to implement a program for the key mining problem was done in less than one working day.

Performance In this section, we present some experimental results of our library for frequent itemset mining. This problem was more particularly choosen for our experiments because there is lot of ressources (algorithms implementations, datasets, benchmarks...) freely available on internet [29]. Whereas, 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, initially we plan to compare *iZi* with the algorithm proposed in [19] for key mining. Unfortunately, the implementation was not available and the authors could not provide us their datasets for confidentiality reasons.

The discovery of frequent itemsets is a problem where implementations are very optimized, specialized, and consequently very competitives. The most performant ones are often the results of many years of research and development. In this context, the objective of these experimentations is to show that a generic library such as *iZi* could have comparable performances with specialized implementations.

Experiments have been done on some FIMI databases [5, 6]. We compared our *Apriori* generic implementation together with two others devoted implementations: one by B. Goethals [30] and one by C. Borgelt [31]. The first one is a quite natural version, while the second one in, to our knowledge, is the best existing *Apriori* implementation, developed in *C* and strongly optimized.

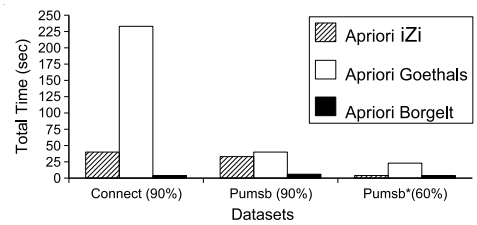

Fig. 6. Comparison of three *Apriori* implementations

The figure 6 shows execution times for some supports of databases *Connect*, *Pumsb* and *Pumsb*∗. One can observe that the performance of our generic version are between the two other implementations. These results are very encouraging, in regards of the simplicity to obtain an operational program.

5 Related works

Most of the algorithm implementations for pattern mining appear as independent and specialized programs, developed mainly to evaluate the effectiveness of the algorithm for a given problem. It is for example the case of all the implementations made available by FIMI workshops [5, 6] for frequent itemsets mining. Other works propose free data mining and machine learning packages.

As an example, the *Illimine* project [32] consider mining data cubes, association rules, sequential pattern, graphs and clustering. In this project, tools have been developed in order to facilitate the use of programs.

However, all these works are dedicated and optimized for the resolution of a specific problem. Thus, these programs cannot be directly used to solve other problems even if they are equivalent. Moreover, the source codes are not always available, and when they are, their modification is very difficult and sometimes impossible due to the complexity of the implementation.

To our knowledge, only one work intents to simplify the programmer task: the DMTL library (Data Mining Template Library) [33].

The DMTL library DMTL is a C++ library composed of algorithms and data structures for frequent pattern mining. This library allows the user to process many types of frequent patterns (sets, sequences, trees and graphs) using generic algorithms implementations. Actually, DMTL could be used to mine any types of patterns respresentable as graphs. Moreover, the data is decoupled of the algorithms, and could be stored in memory, files, Gigabase databases (an embedded object relational database), and PSTL components (a library of persistent containers). Nevertheless, surprisingly DMTL cannot process the FIMI format which is the most commonly used for frequent itemsets mining. This library currently contains two search space strategies: a levelwise *Apriori*-like [20, 8] and a depth-first approach. These algorithms could be used to mine all the frequent patterns of a given database.

However, to our point of view, the DMTL library has three main disavantages. First, this library cannot be used to solve other problems that frequent pattern mining, even other closely related problems such as mining frequent free itemsets. Secondly, only the enumeration of all the frequent patterns is possible. DMTL cannot discover the positive or negative borders of frequent patterns. Thirdly, only "classic" algorithms are implemented, and these algorithms test lot of patterns. None of the strategies use for example patterns of the positive border to prune the search space.
Note that our library intents to give a solution to these limits by allowing the resolution of any pattern mining problem fitting in the framework, the discovery of the theory and the borders, and the use of non classical and efficient algorithms such as *ABS*.

6 Discussion and perspectives

In this paper, we consider the discovery of interesting patterns from data, *when problems are representable as sets* [8]. This work concerns many data mining problems with a new point of view: the rapid development of an efficient and robust program, using a selected algorithm. From a theoretical framework, we propose a C++ library, called *iZi*, which aims at facilitate the development of a scaling solution, using generic algorithms and data structures. Algorithms details and optimizations are transparent for user, and only some specific aspects

of his/her problem must be coded. Moreover, data access is totally decoupled of the other components which enables to use the same code for many data sources wihtout modifications. The experimental results done are encouraging, in comparison with the rapidity and simplicity for devising an operational program.

There are numerous perspectives to this work. First, on a theoretical level, we plan to study some constraint relaxation for accepting more problems, like frequent sequence or tree mining. We also plan to develop a declarative version for such mining problem, with a query language and a cost model, allowing an optimizer to automatically chose the search strategy to liberate the user of this difficult task.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In Buneman, P., Jajodia, S., eds.: SIGMOD Conference, ACM Press (1993) 207–216
2. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**(2) (2000) 372–390
3. Termier, A., Rousset, M.C., Sebag, M.: Treefinder: a first step towards xml data mining. [34] 450–457
4. Asai, T., Arimura, H., Abe, K., Kawasoe, S., Arikawa, S.: Online algorithms for mining semi-structured data stream. [34] 27–34
5. Bayardo Jr., R.J., Zaki, M.J., eds.: FIMI '03, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA, November 19, 2003. In Bayardo Jr., R.J., Zaki, M.J., eds.: FIMI. Volume 90 of CEUR Workshop Proceedings., CEUR-WS.org (2003)
6. Bayardo Jr., R.J., Goethals, B., Zaki, M.J., eds.: FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004. In Bayardo Jr., R.J., Goethals, B., Zaki, M.J., eds.: FIMI. Volume 126 of CEUR Workshop Proceedings., CEUR-WS.org (2004)
7. Goethals, B., Nijssen, S., Zaki, M.J.: Open source data mining: Workshop report. In: OSDM'05, Proceeding of the ACM SIGKDD Open Source Data Mining Workshop, Chicago,USA. (2005)
8. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Min. Knowl. Discov. **1**(3) (1997) 241–258
9. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining frequent patterns with counting inference. SIGKDD Explorations **2**(2) (2000) 66–75
10. Boulicaut, J.F., Bykowski, A., Rigotti, C.: Free-sets: A condensed representation of boolean data for the approximation of frequency queries. Data Min. Knowl. Discov. **7**(1) (2003) 5–22
11. Lopes, S., Petit, J.M., Toumani, F.: Discovering interesting inclusion dependencies: application to logical database tuning. Inf. Syst. **27**(1) (2002) 1–19
12. De Marchi, F., Petit, J.M.: Zigzag: a new algorithm for mining large inclusion dependencies in database. In: ICDM, IEEE Computer Society (2003) 27–34
13. Koeller, A., Rundensteiner, E.A.: Heuristic strategies for inclusion dependency discovery. In Meersman, R., Tari, Z., eds.: CoopIS/DOA/ODBASE (2). Volume 3291 of Lecture Notes in Computer Science., Springer (2004) 891–908
14. Angluin, D.: Queries and concept learning. Machine Learning **2**(4) (1987) 319–342

15. Flouvat, F., De Marchi, F., Petit, J.M.: The iZi library website, http://liris.cnrs.fr/ffflouvat/librairie/index.html
16. Lopes, S., Petit, J.M., Lakhal, L.: Efficient discovery of functional dependencies and armstrong relations. In Zaniolo, C., Lockemann, P.C., Scholl, M.H., Grust, T., eds.: EDBT. Volume 1777 of Lecture Notes in Computer Science., Springer (2000) 350–364
17. Marchi, F.D., Lopes, S., Petit, J.M., Toumani, F.: Analysis of existing databases at the logical level: the dba companion project. SIGMOD Record **32**(1) (2003) 47–52
18. De Marchi, F., Flouvat, F., Petit, J.M.: Adaptive strategies for mining the positive border of interesting patterns: Application to inclusion dependencies in databases. In Boulicaut, J.F., De Raedt, L., Mannila, H., eds.: Constraint-Based Mining and Inductive Databases. Volume 3848 of Lecture Notes in Computer Science., Springer (2005) 81–101
19. Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: Gordian: Efficient and scalable discovery of composite keys. In Dayal, U., Whang, K.Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.K., eds.: VLDB, ACM (2006) 691–702
20. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In Bocca, J.B., Jarke, M., Zaniolo, C., eds.: VLDB, Morgan Kaufmann (1994) 487–499
21. Goethals, B., Zaki, M.J.: Advances in frequent itemset mining implementations: report on fimi'03. SIGKDD Explorations **6**(1) (2004) 109–117
22. Flouvat, F., De Marchi, F., Petit, J.M.: ABS: Adaptive Borders Search of frequent itemsets. [6]
23. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In Chen, W., Naughton, J.F., Bernstein, P.A., eds.: SIGMOD Conference, ACM (2000) 1–12
24. Uno, T., Asai, T., Uchida, Y., Arimura, H.: Lcm: An efficient algorithm for enumerating frequent closed item sets. [5]
25. Borgelt, C.: Efficient implementations of Apriori and Eclat. [5]
26. Bodon, F.: Surprising results of trie-based fim algorithms. [6]
27. Flach, P.A., Savnik, I.: Database dependency discovery: A machine learning approach. AI Commun. **12**(3) (1999) 139–160
28. Jaudoin, H., Flouvat, F.: Techniques de fouille de données pour la réécriture de requêtes en présence de contraintes de valeurs. In Ritschard, G., Djeraba, C., eds.: EGC. Volume RNTI-E-6 of Revue des Nouvelles Technologies de l'Information., Cépaduès-Éditions (2006) 77–88
29. Goethals, B.: Frequent itemset mining implementations repository, http://fimi.cs.helsinki.fi/
30. Goethals, B.: Apriori implementation. University of Antwerp. http://www.adrem.ua.ac.be/~goethals/
31. Borgelt, C.: Recursion pruning for the apriori algorithm. [6]
32. Han, J., Group, D.M.: IlliMine project. University of Illinois Urbana-Champaign Database and Information Systems Laboratory. http://illimine.cs.uiuc.edu/ (2005)
33. Hasan, M., Chaoji, V., Salem, S., Parimi, N., Zaki, M.: DMTL: A generic data mining template library. In: Workshop on Library-Centric Software Design (LCSD'05), with Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'05) conference, San Diego, California. (2005)
34. Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan, IEEE Computer Society (2002)