Partial orders and logical concept analysis to explore patterns extracted by data mining

Peggy Cellier, Sébastien Ferré, Mireille Ducassé, Thierry Charnois

Partial Orders and Logical Concept Analysis to Explore Patterns Extracted by Data Mining

Peggy Cellier[1], Sébastien Ferré[2], Mireille Ducassé[1], and Thierry Charnois[3]

[1] IRISA/INSA Rennes
[2] IRISA/University of Rennes
Campus Beaulieu, F-35043 Rennes Cedex, France,
firstname.lastname@irisa.fr
[3] GREYC/University of Caen
Campus Côte de Nacre, F-14032 Caen cedex, France,
thierry.charnois@unicaen.fr

Abstract. Data mining techniques are used in order to discover emerging knowledge (patterns) in databases. The problem of such techniques is that there are, in general, too many resulting patterns for a user to explore them all by hand. Some methods try to reduce the number of patterns without a priori pruning. The number of patterns remains, nevertheless, high. Other approaches, based on a total ranking, propose to show to the user the top-k patterns with respect to a measure. Those methods do not take into account the user's knowledge and the dependencies that exist between patterns. In this paper, we propose a new way for the user to explore extracted patterns. The method is based on navigation in a partial order over the set of all patterns in the Logical Concept Analysis framework. It accommodates several kinds of patterns and the dependencies between patterns are taken into account thanks to partial orders. It allows the user to use his/her background knowledge to navigate through the partial order, without a priori pruning. We illustrate how our method can be applied on two different tasks (software engineering and natural language processing) and two different kinds of patterns (association rules and sequential patterns).

Keywords: data mining, partial order, selection of patterns, logical concept analysis, formal concept analysis

1 Introduction

Knowledge Discovery in Databases (KDD) [FPSS96] can be seen as a process in three steps: preparation of data, data mining and exploitation of extracted patterns. In the second step of the KDD process, data mining techniques are used in order to discover emerging knowledge (patterns) in databases. That step highlights regularities and tendencies which can give important information to a user about the data. The problem for a practical use is that, in general, too many patterns are generated. It is not easy for a user to explore by hand a large amount of patterns. The problem is not specific to one kind of patterns. Indeed,

a huge amount of association rules [AIS93] but also sequential patterns [AS95] or graph patterns can be extracted with data mining techniques.

In order to address this problem, some methods try to reduce the number of patterns without a priori pruning, for example condensed representation [PBTL99,PC09] or constraints [PHL01]. The number of patterns remains, nevertheless, high. Other approaches, based on a total ranking [KB10], propose to show to a user the top-k patterns with respect to a specific measure. User's knowledge and dependencies between patterns are not taken into account by that kind of methods.

In this paper we propose an application of Logical Concept Analysis (LCA) [FR04] to build a generic framework to explore patterns extracted by data mining techniques. The framework is based on a data structure which organizes the set of patterns, and provides operations on that structure, namely navigation in the set of patterns, selection of patterns of interest and pruning off patterns without interest. The data structure is given in the form of *Hasse diagram* [DP90], exploiting the fact that patterns are naturally partially ordered. Indeed, some patterns are *sub-patterns* of others. The operations take advantage of the power of LCA. As LCA can be applied to any ordering, its navigation capabilities can be re-used as such. Furthermore, the operations of selection of patterns of interest and pruning off patterns without interest can be straightforwardly implemented on top of its updating capabilities. We illustrate how our framework can be instantiated on two different tasks (natural language processing (NLP) and fault localization) and two different kinds of patterns (sequential patterns and association rules). The tasks had ad hoc formalizations [CDFR08] [CC10] which are unified and generalized by the framework proposed in this paper.

The contribution of the paper is twofold. Firstly, as opposed to existing approaches, users can benefit from their background knowledge to navigate through the patterns until their goal(s) have been reached, without a priori pruning. Secondly, the framework is generic, there is no constraint on the kind of patterns and it can accommodate several kinds of tasks. The genericity is mainly due to the power of LCA. Indeed, LCA can be applied to any ordering of patterns. Whereas Formal Concept Analysis (FCA) [GW99] can also be used on partial ordering, it is tied to sets of attributes ordered by inclusion. Furthermore, in LCA partial orders can be combined with other logics allowing a rich description of patterns. For example, the extracted patterns often have some information about statistical measures such as a support value or a confidence value.

In the remaining of the paper, Section 2 presents the case studies. Section 3 gives background knowledge about LCA. Section 4 defines the proposed approach and Section 5 discusses related work.

2 Case Studies

In this section, we describe two different tasks. Those case studies are used in the paper to illustrate the theory (Section 4).

2.1 Natural Language Processing Task

The first application is a Natural Language Processing (NLP) task [CC10]. Some linguistic patterns are automatically extracted from a corpus. The linguistic patterns have to recognize *appositive qualifying phrases* in French texts. Some examples of appositive qualifying phrases are: *"En bon père de famille,"* ("As a good father,"), *", connu pour sa cruauté,"* (", known for his cruelty,"). Several parts of different sentences representing appositive qualifying phrases are collected to build a training corpus. In the training corpus, each appositive qualifying phrase is replaced by a sequence where each word is associated to part-of-speech information. For example, $\langle(en\ en\ PRP)\ (bon\ bon\ ADJ)\ (père\ père\ NOUN)\ (de\ de\ PRP)\ (famille\ famille\ NOUN)\rangle$ is a sequence[4].

From the training corpus, patterns are extracted. The patterns are *closed sequential patterns under constraints*. For instance, $\langle(champion\ NOUN)(PRP)\rangle$ is a sequential pattern that describes phrases starting by a noun whose lemma is "champion" and followed by a preposition (PRP). The *support* of a sequential pattern S in a corpus is the number of sequences of the corpus matching S. The *frequent* sequential patterns are the sequences with a support greater than a threshold. An extracted sequential pattern, S_1, is *closed* if there is no other extracted sequential pattern, S_2, such that S_1 is included in S_2 and $sup(S_1) = sup(S2)$. The advantage of closed sequential patterns is the reduction of redundancy. A preliminary automatic filtering is done with the application of two constraints in order to keep only closed sequential patterns that are relevant for the task: **no gap** in patterns (i.e. between itemsets of sequential patterns) and patterns represent **the beginning** of the appositive qualifying phrases. The number of patterns is high (1 789 patterns). The goal of the user is to identify interesting linguistic patterns among extracted sequential patterns.

2.2 Fault Localization Task

The second application is a fault localization task [CDFR08]. When the result of a program execution is not the same as the expected one, that execution is called a *failure*. Fault localization is a software engineering task that tries to find an explanation to the failures by examining information from the executions. To each execution is associated an *execution trace* that contains information about the execution: the executed lines and the verdict of the execution (*Pass* when the result of the execution is the same as the expected one, otherwise *Fail*).

From the execution traces of different executions of a given program, particular association rules are computed where the conclusion is set to *Fail*. For example, the rule "$r_2 = 78, ..., 81, 84, 87, 90 \rightarrow Fail$" means that "when the lines 78, ..., 81, 84, 87 and 90 are executed, most of the time it implies a failure". In order to measure the relevance of the rules, the *support* and the *lift* values are also computed. The support measures the number of execution traces that

[4] Each word is replaced by three elements : the word itself, its lemma and grammatical information. Sometimes the word and its lemma are identical. ADJ means "adjective" and PRP means "preposition".

execute the lines of the premise of the rule and fail. The lift value measures how the observation of the premise in an execution trace increases the probability that this execution fails. Some rules are identified as "failure rules". A rule r is a failure rule if there exist some failed executions that contain the whole premise of r in their trace but not the whole premise of rules more specific than r.

The number of extracted rules can be high. The goal of the user is to give at least one explanation for each failure.

3 Logical Concept Analysis (LCA)

Logical Concept Analysis (LCA) is defined in [FR04]. It is a general theory allowing extensions of Formal Concept Analysis (FCA) [GW99] to be easily specified in a formal way. In LCA the description of an object is a logical formula instead of a set of attributes as in FCA. Pattern structures [GK01] are an equivalent alternative to LCA, where "patterns" are used instead of formulas.

3.1 Logic and Partial Order

Definition 1 (logic). *A* logic *is a lattice* $\mathcal{L} = (L, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$ *where*

- L *is the language of formulas,*
- $\sqsubseteq$ *is the subsumption relation (the order on the formulas),*
- $\sqcap$ *and* $\sqcup$ *are, respectively, the lower bound and the upper bound,*
- $\top$ *and* $\bot$ *are, respectively, the top and the bottom of the lattice.*

Let f and g be two formulas, i.e. $f, g \in L$, if $f \sqsubseteq g$ and $g \sqsubseteq f$ then f and g are said *logically equivalent*. It is denoted by $f \equiv g$. Some logics are partially defined, *partial logic*, namely the lower and upper bounds are not always defined. The definition of a logic is left very abstract. This makes it possible to accommodate non-standard types of logics. For example, $\mathcal{L}_{base}$ is the logic that describes base domains, e.g. $support = 3 \sqsubseteq support \geq 2$. The subsumption relation also allows the terms of a taxonomy to be ordered (see line attributes in the fault localization illustration in Section 4.5).

We define a partial order, $\mathcal{P}$, as a couple $(P, \leq)$ where P is a set and $\leq$ is a binary relation on P that is reflexive, anti-symmetric and transitive [DP90]. In the LCA framework, we can define a logic associated to a partial order thanks to *logic functors* (see [FR04] for details). There are several logic functors. $\mathcal{F}_{POSET}(\mathcal{P})$ is the functor that builds a partial logic from a partial order, $\mathcal{P} = (P, \leq)$, such that $p_1 \sqsubseteq p_2$ if $p_1 \leq p_2$. $\mathcal{F}_{UNION}$, also denoted by $\cup$, is the functor that combines several logics into a logic, potentially partial. $\mathcal{F}_{LIS}$ is the functor that builds a well-defined logic from a partial logic by adding boolean connectors ("and", "or" and "not") and the closed world assumption. Indeed, the "and" and "or" connectors guarantee the lower and upper bound of the logic.

3.2 Logical Context

Definition 2 gives the definition of a logical context in the LCA framework. Definition 3 defines the logical versions of *extent* and *intent*. The extent of a logical formula f is the set of objects in $\mathcal{O}$ whose description is subsumed by f. The intent of a set of objects O is the most precise formula that subsumes all descriptions of objects in O. Definition 4 gives the definition of a *logical concept*.

Definition 2 (logical context). *A* logical context *is a triple* $(\mathcal{O}, \mathcal{L}, d)$ *where* $\mathcal{O}$ *is a set of objects,* $\mathcal{L}$ *is a logic and* d *is a mapping from* $\mathcal{O}$ *to* $\mathcal{L}$ *that describes each object by a formula.*

Definition 3 (extent, intent). *Let* $\mathcal{K} = (\mathcal{O}, \mathcal{L}, d)$ *be a logical context. The definition of the extent is:* $\forall f \in \mathcal{L}$ *, ext(f)* $= \{o \in \mathcal{O} \mid d(o) \sqsubseteq f\}$*. The definition of the intent is:* $\forall O \subseteq \mathcal{O}$*, int(O)* $= \bigsqcup_{o \in O} d(o)$*.*

Definition 4 (logical concept). *Let* $\mathcal{K} = (\mathcal{O}, \mathcal{L}, d)$ *be a logical context. A* logical concept *is a pair* $c = (O, f)$ *where* $O \subseteq \mathcal{O}$*, and* $f \in \mathcal{L}$*, such that* $int(O) \equiv f$ *and* $ext(f) = O$*. O is called the extent of the concept c, i.e.* ext_c*, and f is called its intent, i.e.* int_c*.*

The set of all logical concepts is ordered and forms a *lattice*: let c and c' be two concepts, $c \leq c'$ iff $ext_c \subseteq ext_{c'}$. Note that $c \leq c'$ iff $int_c \sqsubseteq int_{c'}$. Concept c is called a **sub-concept** of c'.

4 The Proposed LCA Framework to Navigate into the Set of Extracted Patterns

In this section, we present the general framework and show how it can be instantiated for the case studies presented in Section 2. Firstly, some pre-requisite to use the method are presented (Section 4.1). Secondly, the logical context is defined and an example with the NLP task is given (Section 4.2). Thirdly, the user actions are presented (Section 4.3) and a stopping criterion is defined (Section 4.4). Then a complete example is given with the fault localization task (Section 4.5). Finally, there is a discussion about the proposed approach (Section 4.6).

4.1 Preliminaries

In our method there are three important parameters: the patterns, the user and the goal of the user. The first hypothesis is that the patterns are already extracted. We do not make any assumption about the kind of patterns or about the extraction technique. The only one pre-requisite is the definition of a partial order over the set of patterns. Note that there is a natural order between patterns: the inclusion order. The second hypothesis is that the user is a domain expert. This means that he/she can judge the relevance of any individual pattern with sufficient information. The proposed method is designed to help a user

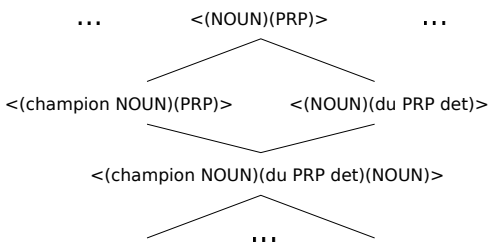

Fig. 1. Excerpt of the partial order on the patterns extracted from a corpus. The most general patterns are at the top.

to understand patterns using his/her background knowledge. If the user is not an expert, the method cannot provide much help. The last hypothesis is that the goal of the user is clearly expressed as a subset of patterns that have to be identified. This hypothesis is important to define a relevant stopping criterion (see Section 4.4).

4.2 A Logical Context to Explore Extracted Patterns

Partial Order. The patterns are naturally partially ordered. Indeed, some patterns are more general than others: *sub-patterns*. Note that constraints can provide other, possibly more relevant, partial orders.

For example, Definition 5 gives the partial order on the patterns, $\mathcal{P}_{seq}$, for the NLP task. As mentioned in Section 2.1, the patterns used for that task are closed sequential patterns under two constraints. Figure 1 shows a part of that partial order. We see that $\langle(champion\ NOUN)(PRP)\rangle$ is more specific than pattern $\langle(NOUN)(PRP)\rangle$. Indeed, all phrases that match $\langle(champion\ NOUN)(PRP)\rangle$ also match $\langle(NOUN)(PRP)\rangle$, but the converse is not true. The phrase "*Champion du monde*" ("Champion of the world") matches both patterns, but "*Gagnant du concours*" ("Winner of the contest") only matches $\langle(NOUN)(PRP)\rangle$. From that partial order, a logic is derived $\mathcal{L}_{seq} = \mathcal{F}_{POSET}(\mathcal{P}_{seq})$.

Definition 5 ($\mathcal{P}_{seq}$). *Let $\mathcal{P}_{seq}$ be a couple (P_{seq},$\leq_{seq}$) such that:*

- *P_{seq} is all extracted closed sequential patterns that check the constraints,*
- *Let $l = \langle I_1...I_n\rangle$ and $l' = \langle J_1...J_m\rangle$ be two patterns of P_{seq} then $l \leq_{seq} l'$ if $m \leq n$ and $\forall i \in 1..m\ J_i \subseteq I_i$.*

Pattern Context. From the extracted patterns and their associated partial order, a logical context is defined. That context is called *pattern context*. Definition 6 defines pattern contexts by instantiating Definition 2. In this context the objects are identifiers of the extracted patterns. Each pattern is described by the pattern itself. That part of the description is unique for each pattern and mandatory. In addition, the pattern description can contain additional optional information, for example statistical measures (e.g., support, lift). The concept

lattice of a *pattern context* represents the search space for exploring the information about patterns.

Definition 6 (pattern context). *Let $\mathcal{P} = (P, \leq)$ be a partial order over the pattern set. The associated* Pattern Context *is a triple $\mathcal{K}(\mathcal{P}) = (\mathcal{O}_p, \mathcal{L}_p, d_p)$ where*

- *$\mathcal{O}_p$ are the identifiers of the patterns of P,*
- *$\mathcal{L}_p = \mathcal{F}_{LIS}(\mathcal{F}_{POSET}(\mathcal{P}) \cup \mathcal{L}_{base})$,*
- *Let $p \in P$, the description of p is a conjunction (defined in $\mathcal{F}_{LIS}$) of p and optional additional information about p (defined in $\mathcal{L}_{base}$).*

Pattern ID	$\mathcal{P}_{seq}$	Add. information: support
p_1	$\langle(NOUN)(PRP)\rangle$	805
p_2	$\langle(champion\ NOUN)(PRP)\rangle$	106
p_3	$\langle(NOUN)(du\ PRP\ det)\rangle$	187
p_4	$\langle(champion\ NOUN)(du\ PRP\ det)(NOUN)\rangle$	94
...		

Table 1. Excerpt of pattern context for the natural language processing task.

Table 1 gives an example of *Pattern Context* for the NLP task. The objects are the identifier of frequent closed sequential patterns. Each line describes a pattern (in $\mathcal{P}_{seq}$) and the associated support value (in $\mathcal{L}_{base}$). The support values come from a previous data mining step.

4.3 User Actions: Navigation and Updating

The lattice defined in the previous section can be very large and cannot be displayed. We propose to navigate through the lattice thanks to a LCA tool such as Camelis[5] [Fer09] and Abilis[6] [AFR10]. They allow a user to navigate in logical contexts and to update them. They do not compute the whole lattice a priori but compute parts of the lattice on demand when relevant to the navigation. Figure 2 shows Camelis with the NLP context [7].

In LCA tools, the interface has three main parts. At the top, the *query view* displays the current query. In Figure 2 the query is "`support >= 2`", it means that only patterns whose support is greater than 2, are displayed. At the bottom left hand part, the *navigation tree* displays the features of the navigation (the patterns themselves and additional information). The number next to a feature is the number of patterns that have that feature in their description. For example, 198 patterns have the feature $\langle(NOUN)(PRP)\rangle$ in their description, namely 198 patterns are more specific than the pattern $\langle(NOUN)(PRP)\rangle$. We see the patterns from the excerpt of $\mathcal{P}_{seq}$ of Figure 1 (underlined patterns). On the right hand part, the *pattern view* displays all patterns whose description is subsumed by the query. With respect to the query view, only patterns having a support equal or greater than 2 are shown there.

[5] http://www.irisa.fr/LIS/ferre/camelis
[6] http://ledenez.insa-rennes.fr/abilis/
[7] Annotations in bold red have been added for this paper.

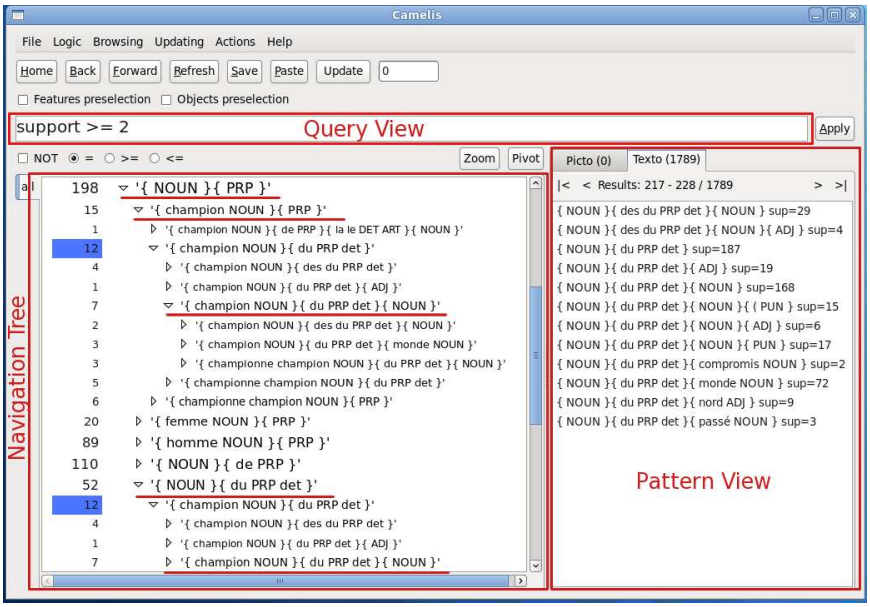

Fig. 2. Camelis with the NLP context.

Navigation. The user can navigate through different kinds of attributes (patterns and additional information). The flexibility in the navigation comes from the logics. Indeed, thanks to the combination of logics (logic functors), the user can create queries that mix elements of the partial order and additional information such as support values.

For the NLP task, the user explores the patterns from the most general ones, which are matched by a lot of phrases (e.g., $\langle(champion\ NOUN)(PRP)\rangle$ matched by 805 phrases), to the most specific patterns, that are matched by less phrases (e.g., $\langle(champion\ NOUN)(du\ PRP\ det)(NOUN)\rangle$[8] matched by 94 phrases). The partial order over the set of patterns is highlighted in the navigation tree. Note that behind the so called navigation tree, there is not a tree structure but a partial order ($\mathcal{P}_{seq}$). It explains the fact that the pattern $\langle(champion\ NOUN)(du\ PRP\ det)(NOUN)\rangle$ appears twice in the navigation tree. Indeed, it is the same pattern that has two parents.

Context Updating to Select and Prune Patterns. When exploring the patterns the user may add some information about the patterns by adding some features to their description. The two main advantages are that it permits to build a result set with selected patterns and to prune patterns without interest, i.e. patterns already selected or patterns not interesting for the purpose. If a pattern p is selected, all more specific patterns than p do not have to be explored,

[8] *det* means "determiner"

they are subsumed by p. In the same way, if a pattern p is away from the point, all more specific patterns than p are away from the point and do not have to be explored. Therefore, when a pattern is tagged, all more specific patterns are also tagged and the search space is pruned. To facilitate the navigation and to reduce the search space, we propose to create a taxonomy of tags. All tags required to update the descriptions of patterns are subsumed by a general tag: `Tags`. When navigating, the user adds to the query `not Tags` in order to eliminate patterns already tagged from the views.

For instance, for the NLP task there are two tags in `Tags`: `LinguisticPattern` and `NotLinguisticPattern`. Those tags are used in two cases. The first case is when the user finds a pattern, p, really interesting to recognize appositive qualifying phrases. The user selects p and the query becomes "p". In the pattern view, all patterns that are more specific than p are thus displayed. Then the user adds the tag `LinguisticPattern` to all patterns displayed. Thus, p and all more specific patterns than p are tagged as `LinguisticPattern`, i.e. they belong to the resulting set of linguistic patterns. In order to avoid to explore those patterns again, the user has just to add `not Tags` to the current query. The second case is when the user finds a pattern, p, clearly not relevant to recognize appositive qualifying phrases. The user selects p and the query becomes "p". In the pattern view, all patterns that are more specific that p are thus displayed. Then the user adds the tag `NotLinguisticPattern` to all patterns displayed. Thus, p and all more specific patterns are tagged as `NotLinguisticPattern`. As previously, thanks to the "`not Tags`" query, the patterns already labelled are pruned from the navigation space of the user.

4.4 Stopping Criterion

When the user can clearly define a goal as a set of patterns to label, P_{goal}, a stopping criterion is provided (Property 1). The user specifies his/her goal by adding a specific feature to the description of the goal patterns.

Property 1. Let P be a set of patterns. Let $P_{goal} \subseteq P$ be the user goal. The process stops when all elements of P_{goal} are labelled.

Let goal be the feature that enables to tag a pattern as being in the user goal. Assuming (hypothesis 2) that the user is competent, every time he identifies a pattern in the goal he tags it accordingly. When query "`not Tags and goal`" has no answer, the process ends. The advantage is that users constantly know without any effort how much of the information they still have to investigate; another advantage is that users do not need to explore the whole set of patterns even if the goal is the whole set of patterns.

4.5 The Fault Localization Example

As presented in Section 2.2, the goal of the fault localization is to understand why a program fails. In this section, we show how the proposed approach can be instantiated to that task.

Partial Order. The first step is the definition of the partial order over the association rules extracted from execution traces (Definition 7). All association rules have the same conclusion: $Fail$. The partial order is thus derived from the inclusion on the premise of the rules. A rule, r, is more specific than another one, r', if the premise of r' is included into the premise of r. For example, $r_1 = 81, 90, 93 \rightarrow Fail \leq_{ar} r_2 = 81, 93 \rightarrow Fail$.

Definition 7 ($\mathcal{P}_{ar}$). *Let $\mathcal{P}_{ar}$ =(P_{ar},$\leq_{ar}$) be a partial order where:*

- *P_{ar} is the set of extracted association rules;*
- *Let $r = L \rightarrow Fail$ and $r' = L' \rightarrow Fail$ be two association rules of P_{ar} then $r \leq_{ar} r'$ if $L' \subseteq L$.*

Pattern ID	Pattern	Add. Information													
		support	lift	`Failure`	line					inv_line					
					81	90	93	101	...	81	90	93	101	...
r_1	$81, 90, 93, ... \rightarrow Fail$	60	1.48		X	X	X		...		X			...
r_2	$81, 93, ... \rightarrow Fail$	112	1.42		X		X		...	X	X		X	...
r_3	$81, 93, 101, ... \rightarrow Fail$	52	1.36		X		X	X	...				X	...
r_4	$93, ... \rightarrow Fail$	112	1.35				X		...	X	X	X	X	...
...														

Table 2. Excerpt of pattern context for the fault localization task.

Pattern Context. Table 2 gives an excerpt of a pattern context for the fault localization task. The pattern context associated to the fault localization task, describes each rule, r, by: (1) r, the rule itself; (2) the support value and (3) the lift value that are computed during the data mining step; (4) the `Failure` attribute if r is a failure rule; (5) the lines that belong to the premise of the rule (`line`); (6) the lines that belong only to the premise of r or more specific rules than r (`inv_line`).

The elements of `line` form a taxonomy on the lines. That taxonomy is described by the partial order defined in Definition 8. At the top of the taxonomy, there are the lines that are specific to the most general rules. At the bottom, there are the lines that are specific to the most specific rules. The elements of `inv_line` form the inverted taxonomy of `line`, $\mathcal{P}_{inv\_line}$ =(P_{line},$\geq_{line}$). The logic of the pattern context can thus be summarized by:

$$\mathcal{L}_p = \mathcal{F}_{LIS}(\mathcal{F}_{POSET}(\mathcal{P}_{ar}) \cup \mathcal{F}_{POSET}(\mathcal{P}_{line}) \cup \mathcal{F}_{POSET}(\mathcal{P}_{inv\_line}) \cup \mathcal{L}_{base}).$$

Definition 8 ($\mathcal{P}_{line}$). *Let $\mathcal{P}_{line}$ be a couple (P_{line},$\leq_{line}$) such that:*

- *P_{line} is all lines of the program;*
- *Let l and l' be two lines of P_{line} then $l \leq_{line} l'$ if all association rules that contain l' also contain l.*

For instance, r_2 which is not a failure rule, represents the 112 failed executions that execute lines 81, 93, ... of the program. $r_1 \leq_{ar} r_2$ and $r_3 \leq_{ar} r_2$, r_2 has thus

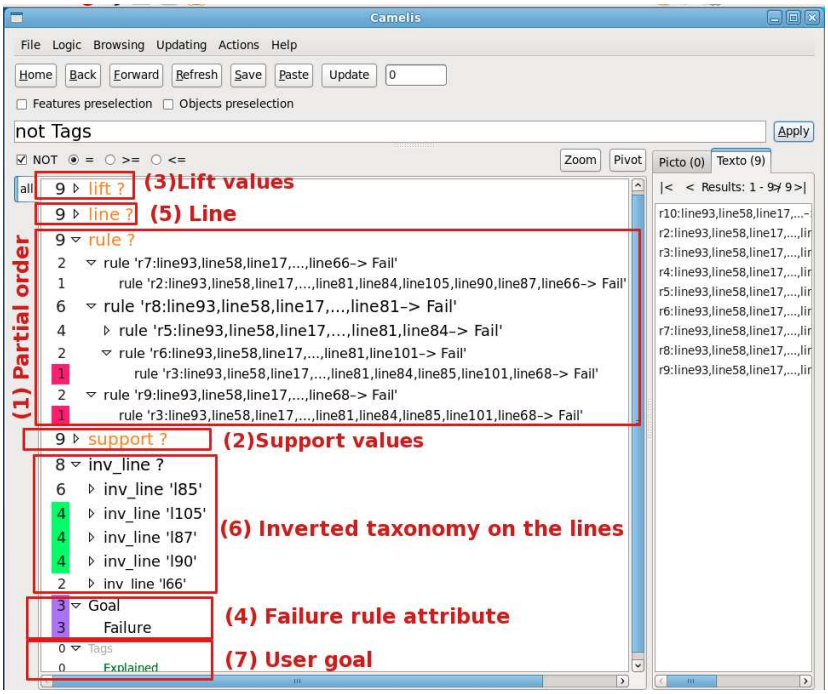

Fig. 3. Camelis with the fault localization context.

the specific lines of r_1 and r_3 in its inverted line description, i.e. 90 and 101, plus its specific line 81. Figure 3 shows Camelis with the fault localization context and highlights the different elements of rule descriptions.

Navigation. In order to understand the behaviour of the program when it fails, the user starts by checking the lines that are specific to failure rules. When this is not sufficient to understand why the program fails, the user checks specific lines of a more general rule and so on. It means that the user checks the lines of the patterns in the order of the inverted line taxonomy.

In order to navigate from the lines that specifically belong to the failure rules, the user can select `Failure` in the navigation tree. Then only failure rules appear in the pattern view. The user can then choose one rule among them and display the intent of the rule which is its description. The query is now the description of the selected rule. That description shows the most specific lines describing the rule. The user can select one of those lines as the starting point of the navigation and explore the lines through the inverted order (`inv_line`).

Updating. For the fault localization task, there is only one tag: `Explained` (7). The user explores the lines in the reverse order (`inv_line`). When the user sees a line, `inv_line`:l, which allows him/her to understand a failure, he/she

adds the attribute `Explained` to all rules that have `line:`l in their description. Consequently, those rules are no more interesting for the exploration, they are explained by l. The user can add `not Tags` in the query to avoid seeing the association rules that are already "explained".

User Goal and Stopping Criterion. For the fault localization task, the user goal is to label all *failure rules*. The exploration thus stops when there is one explanation by failure rule, namely when all failure rules are labelled by `Explained`.

4.6 Discussion

The proposed approach does not depend on the type of patterns. The only requirement is a partial order on the patterns. There is always a natural order between patterns: the inclusion order of their cover in the original database. But constraints may introduce other more relevant orders.

There are several advantages to use the LIS framework. The first advantage is when the number of patterns is high and thus the size of the lattice is also high. LIS tools do not compute the whole lattice a priori but parts of the lattice on demand when it is relevant for the navigation. The second advantage is that the patterns are organized as a lattice. It allows to highlight the dependencies between them whereas other existing approaches, such as total ranking, do not provide that information. In addition, thanks to the logics, the patterns can have a rich description (e.g., integer values, partial order, taxonomy). Finally, when exploring, the user can add information in order to prune the search space and to be helped in its exploration.

The definition of a user goal is not a requirement, indeed, the user may want to navigate through patterns. But when the user has a clear goal, it is important to specify it with care. Indeed, the user goal defines the stopping criterion.

5 Related Work

The method that we propose in this paper can be applied after methods that reduce the number of patterns (e.g., condensed representation [PBTL99]), when the set of patterns remains too large to be explored by hand. In addition, unlike top-k ranking methods, our method takes into account the user's knowledge and highlights the dependencies between patterns. There are some methods that reduce more drastically the number of patterns. Recursive mining [CSK+08] gives control over the number of patterns, but some information is lost, indeed the data are summarized. In order to reduce the number of patterns, one can compute the stable concepts [Kuz07]. That method is interesting when the goal is to compute a general overview of the data with respect to group behaviours, for example [JKN08]. If the user is interested in patterns that are less frequent, the number of patterns remains high. The approach proposed in this paper does not a priori filter out patterns, but gives a data structure (the lattice) to facilitate

the exploration of the patterns. In addition, thanks to the updating step and the stopping criterion, the number of patterns that are really explored by the user should be lower than the number of the whole set of patterns. In [CG05], Garriga proposes to summarize sequential patterns thanks to partial orders; in [MOG09], the authors propose to explore association rules thanks to *rule schemas*; in [MG10], they propose to improve the method by integrating the user knowledge with an ontology. Unlike those methods, our approach is generic and can be applied on sequential patterns but also on association rules. The only requirement is the definition of the partial order on patterns. Richards *et al.* [RM03] have proposed to display the specific rules (Ripple-Down rules) in a lattice thanks to formal concept analysis (FCA) [GW99]. They create an artificial hierarchy over the rules in order to display them in a lattice. Note that to scale up, they have to filter out some nodes of the hierarchy, whereas our method does not a priori prune patterns. Visual data mining [Kei02,SBM08] provides users with a visualization of the patterns in a graphical way. That method is useful when little is known about the data and the exploration goals are vague. The goal of that kind of methods is not the same as the goal of our method. Indeed, our method is used when a lot of patterns are generated and the user has a specific goal and wants accurate details.

6 Conclusion

In this paper we have presented a new way to interactively explore patterns extracted with data mining techniques. The method is based on navigation in a partial order over the set of all patterns. It accommodates several kinds of patterns and the dependencies between patterns are taken into account. It allows users to use their background knowledge to navigate through the partial order until their goal(s) have been reached, without a priori pruning. In addition, the updating features allows the user to be helped by the reduction of the search space while exploring. We have formally defined our approach in the LCA framework, and we have illustrated our method on two different tasks and two different kinds of patterns.

References

[AFR10] P. Allard, S. Ferré, and O. Ridoux. Discovering functional dependencies and association rules by navigating in a lattice of OLAP views. In *Concept Lattices and Their Applications*, pages 199–210. CEUR-WS, 2010.

[AIS93] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Int. Conf. on Management of Data*. ACM Press, 1993.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Int. Conf. on Data Engineering*. IEEE, 1995.

[CC10] P. Cellier and T. Charnois. Fouille de données séquentielle d'itemsets pour l'apprentissage de patrons linguistiques. In *Traitement Automatique des Langues Naturelles (short paper)*, 2010.

[CDFR08] P. Cellier, M. Ducassé, S. Ferré, and O. Ridoux. Formal concept analysis enhances fault localization in software. In *Int. Conf. on Formal Concept Analysis (ICFCA)*, volume 4933 of *LNCS*. Springer, 2008.

[CG05] G. Casas-Garriga. Summarizing sequential data with closed partial orders. In *SIAM International Data Mining Conference (SDM)*, 2005.

[CSK+08] B. Crémilleux, A. Soulet, J. Klema, C. Hébert, and O. Gandrillon. *Discovering Knowledge from Local Patterns in SAGE data*. IGI Publishing, 2008.

[DP90] B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order: second edition 2001*. Cambridge University Press, 1990.

[Fer09] S. Ferré. Camelis: a logical information system to organize and browse a collection of documents. *Int. J. General Systems*, 38(4), 2009.

[FPSS96] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: an overview. In *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence, 1996.

[FR04] S. Ferré and O. Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3):383–419, Elsevier, 2004.

[GK01] B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In *Proc. of the Int. Conf. on Conceptual Structures: Broadening the Base*, ICCS '01, pages 129–142. Springer-Verlag, 2001.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1999.

[JKN08] N. Jay, F. Kohler, and A. Napoli. Analysis of social communities with iceberg and stability-based concept lattices. In *Int. Conf. on Formal Concept Analysis (ICFCA)*, volume 4933 of *LNCS*. Springer, 2008.

[KB10] K. Kontonasios and T. De Bie. An information-theoretic approach to finding informative noisy tiles in binary databases. In *Proc. of the SIAM Int. Conf. on Data Mining*, pages 153–164, 2010.

[Kei02] D. A. Keim. Information visualization and visual data mining. *IEEE Trans. Vis. Comput. Graph.*, 8(1):1–8, 2002.

[Kuz07] S. O. Kuznetsov. On stability of a formal concept. In *Annals of Mathematics and Artificial Intelligence*. Springer Netherlands ACM, 2007.

[MG10] C. Marinica and F. Guillet. Knowledge-based interactive postmining of association rules using ontologies. *IEEE Trans. Knowl. Data Eng.*, 2010.

[MOG09] C. Marinica, A. Olaru, and F. Guillet. User-driven association rule mining using a local algorithm. In *Int. Conf. on Enterprise Information Systems (ICEIS) (2)*, pages 200–205, 2009.

[PBTL99] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Int. Conf. on Database Theory*, pages 398–416. Springer-Verlag, 1999.

[PC09] M. Plantevit and B. Crémilleux. Condensed representation of sequential patterns according to frequency-based measures. In *Int. Symp. on Advances in Intelligent Data Analysis*, LNCS(5772). Springer, 2009.

[PHL01] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Int. Conf. on Data Engineering*. IEE computer society, 2001.

[RM03] D. Richards and U. Malik. Mining propositional knowledge bases to discover multi-level rules. In *Mining Multimedia and Complex Data*, volume 2797 of *LNCS*, pages 199–216. Springer Berlin / Heidelberg, 2003.

[SBM08] S. J. Simoff, M. H. Böhlen, and A. Mazeika, editors. *Visual Data Mining*. Springer-Verlag, Berlin, Heidelberg, 2008.