Workshop Notes

International Workshop

"What can FCA do for Artificial Intelligence?"

FCA4AI

International Joint Conference on Artificial Intelligence

IJCAI 2013

August 4, 2013

Beijing, China

Editors

Sergei O. Kuznetsov (NRU HSE Moscow)

Amedeo Napoli (LORIA Nancy)

Sebastian Rudolph (TU Dresden)

`http://fca4ai.hse.ru/2013/`

What FCA Can Do for Artificial Intelligence?
FCA4AI: An International Workshop

Preface

This is the second edition of the FCA4AI workshop, the first edition being associated to the ECAI 2012 Conference, held in Montpellier, in August 2012 (see http://www.fca4ai.hse.ru/). In particular, the first edition of the workshop showed that there are many AI researchers interested in FCA. Based on that, the three co-editors decided to organize a second edition of the FCA4AI workshop at the IJCAI 2013 Conference in Beijing.

Formal Concept Analysis (FCA) is a mathematically well-founded theory aimed at data analysis and classification. FCA allows one to build a concept lattice and a system of dependencies (implications) which can be used for many AI needs, e.g. knowledge processing involving learning, knowledge discovery, knowledge representation and reasoning, ontology engineering, as well as information retrieval and text processing. Thus, there exist many "natural links" between FCA and AI.

Recent years have been witnessing increased scientific activity around FCA, in particular a strand of work emerged that is aimed at extending the possibilities of FCA w.r.t. knowledge processing, such as work on pattern structures and relational context analysis. These extensions are aimed at allowing FCA to deal with more complex than just binary data, both from the data analysis and knowledge discovery points of view and from the knowledge representation point of view, including, e.g., ontology engineering.

All these works extend the capabilities of FCA and offer new possibilities for AI activities in the framework of FCA. Accordingly, in this workshop, we are interested in two main issues:

- How can FCA support AI activities such as knowledge processing (knowledge discovery, knowledge representation and reasoning), learning (clustering, pattern and data mining), natural language processing, information retrieval.
- How can FCA be extended in order to help AI researchers to solve new and complex problems in their domains.

The workshop is dedicated to discuss such issues. The papers submitted to the workshop were carefully peer-reviewed by two members of the program committee and 11 papers with the highest scores were selected. We thank all the PC members for their reviews and all the authors for their contributions. We also thank the organizing committee of ECAI-2012 and especially workshop chairs Jérôme Lang and Michèle Sebag for the support of the workshop.

The Workshop Chairs

Sergei O. Kuznetsov
National Research University Higher Schools of Economics, Moscow, Russia

Amedeo Napoli
LORIA (CNRS – INRIA – Université de Lorraine), Vandoeuvre les Nancy, France

Sebastian Rudolph
Technische Universität Dresden, Germany

Program Committee

Mathieu D'Aquin (Open University, UK)

Franz Baader (Technische Universität Dresden, Germany)

Karell Bertet (Université de La Rochelle, France, Germany)

Claudio Carpineto (Fondazione Ugo Bordoni, Roma, Italy)

Felix Distel (Technische Universität Dresden, Germany)

Peter Eklund (University of Wollongong, Australia)

Sébastien Ferré (IRISA Rennes, France)

Pascal Hitzler (Wright State University, Dayton, Ohio, USA)

Dmitry I. Ignatov (NRU Higher School of Economics, Moscow, Russia)

Mehdi Kaytoue (INSA - LIRIS Lyon, France)

Markus Krötzsch (University of Oxford, UK)

Sergei A. Obiedkov (NRU Higher School of Economics, Moscow, Russia)

Uta Priss (Ostfalia University of Applied Sciences, Wolfenbüttel, Germany)

Baris Sertkaya (SAP Dresden, Germany)

Henry Soldano (Université de Paris-Nord, France)

Table of Contents

2 *FCA and pattern structures for mining care trajectories*
Aleksey Buzmakov, Elias Egho, Nicolas Jay, Sergei O. Kuznetsov, Amedeo Napoli and Chedy Raïssi 7

3 *Using pattern structures to support information retrieval with Formal Concept Analysis*
Víctor Codocedo, Ioanna Lykourentzou, Hernan Astudillo and Amedeo Napoli 15

4 *FCA-Based Concept Detection in a RosettaNet PIP Ontology*
Jamel Eddine Jridi and Guy Lapalme 25

5 *Bases via Minimal Generator*
Pablo Cordero, Manuel Enciso, Angel Mora Bonilla and Manuel Ojeda-Aciego 32

6 *Debugging Program Code Using Implicative Dependencies*
Artem Revenko 37

7 *Practical Computing with Pattern Structures in FCART Environment*
Aleksey Buzmakov and Alexey Neznanov 49

8 *Towards Knowledge Structuring of Sensor Data Based on FCA and Ontology*
Peng Wang, Wenhuan Lu, Zhaopeng Meng, Jianguo Wei and Françoise Fogelman-Soulié 53

FCA and pattern structures for mining care trajectories

Aleksey Buzmakov[1,2], Elias Egho[1], Nicolas Jay[1], Sergei O. Kuznetsov[2],
Amedeo Napoli[1], and Chedy Raïssi[1]

[1] LORIA (CNRS – Inria NGE – U. de Lorraine), Vandœuvre-lès-Nancy, France
[2] National Research University Higher School of Economics, Moscow, Russia
{aleksey.buzmakov, chedy.raissi}@inria.fr,
{elias.egho, nicolas.jay, amedeo.napoli}@loria.fr, skuznetsov@hse.ru

Abstract. In this paper, we are interested in the analysis of sequential data and we propose an original framework based on Formal Concept Analysis (FCA). For that, we introduce sequential pattern structures, an original specification of pattern structures for dealing with sequential data. Pattern structures are used in FCA for dealing with complex data such as intervals or graphs. Here they are adapted to sequences. For that, we introduce a subsumption operation for sequence comparison, based on subsequence matching. Then, a projection, i.e. a kind of data reduction of sequential pattern structures, is suggested in order to increase the efficiency of the approach. Finally, we discuss an application to a dataset including patient trajectories (the motivation of this work), which is a sequential dataset and can be processed with the introduced framework. This research work provides a new and efficient extension of FCA to deal with complex (not binary) data, which can be an alternative to the analysis of sequential datasets.

Keywords: formal concept analysis, pattern structures, sequential pattern structures, sequences

Introduction

Sequence data is largely present and used in many applications. Consequently, mining sequential patterns from sequence data has become an important and crucial data mining task. In the last two decades, the main emphasis has been on developing efficient mining algorithms and effective pattern representations [1–5]. However, the problem with traditional sequential pattern mining algorithms (and generally with all pattern enumeration algorithms) is that they generate a large number of frequent sequences while few of them are truly relevant. Moreover, in some particular cases, only sequential patterns of a certain type are of interest and should be mined first. *Are we able to develop a framework for taking into account only patterns of required types?* In addition, another drawback for these pattern enumeration algorithms is that they depend on a user selected support threshold, which is usually hard to be properly set by non-experts. *How can one avoid the setting of support threshold while having optimal pattern analysis?*

The above questions can be answered by addressing the problem of analyzing sequential data with the formal concept analysis framework (FCA), an elegant mathematical approach to data analysis [6], and pattern structures, an extension of FCA that handles complex data [7]. We explain the usage of projections which are mathematical functions respecting certain properties and allow to reduce the computational costs by reducing the volume of resulting patterns. Such a reduction helps an expert to interpret the extracted sequential patterns and reduces the famous "pattern flooding".

In this paper, we develop a novel and efficient approach for working with sequential pattern structures in FCA. The rest of the paper is organized as follows. Section 1 introduces main definitions of FCA and pattern structures. The next section defines sequential pattern structures. Then, before concluding the results are presented and discussed in Section 3.

1 FCA and Pattern Structures

FCA [6] is a mathematical formalism having many applications in data analysis. Pattern structures is a generalization of FCA for dealing with complex structures, such as sequences or graphs [7]. As it is a generalization it is enough to introduce only pattern structures.

Definition 1. *A pattern structure is a triple $(G, (D, \sqcap), \delta)$, where G is a set of objects, $(D, \sqcap)$ is a complete meet-semilattice of descriptions and $\delta : G \to D$ maps an object to a description.*

The lattice operation in the semilattice ($\sqcap$) corresponds to the similarity between two descriptions d_1 and d_2, i.e. the description which is common between d_1 and d_2. Standard FCA can be presented in terms of pattern structures in the following way. The set of objects G remains, while the semilattice of descriptions is $(\wp(M), \cap)$, where $\wp(M)$ is a powerset of M, and, thus, a description is a set of attributes, and the similarity operation corresponding to the set intersection, i.e. the similarity is the set of shared attributes. If $x = \{a, b, c\}$ and $y = \{a, c, d\}$ then $x \sqcap y = x \cap y = \{a, c\}$. The mapping $\delta : G \to \wp(M)$ is given by, $\delta(g) = \{m \in M \mid (g, m) \in I\}$, returning the describing set of attributes.

The Galois connection for a pattern structure $(G, (D, \sqcap), \delta)$ between the set of objects and the semilattice of descriptions is defined as follows:

$$A^\diamond := \bigsqcap_{g \in A} \delta(g), \qquad \text{for } A \subseteq G$$

$$d^\diamond := \{g \in G \mid d \sqsubseteq \delta(g)\}, \qquad \text{for } d \in D$$

Given a set of objects A, $A^\diamond$ returns the description which is common to all objects in A. And given a description d, $d^\diamond$ is the set of all objects whose description subsumes d. The partial order on D ($\sqsubseteq$) is defined w.r.t. the similarity operation $\sqcap$: $c \sqsubseteq d \Leftrightarrow c \sqcap d = c$, and c is subsumed by d.

Definition 2. *A pattern concept of a pattern structure $(G, (D, \sqcap), \delta)$ is a pair (A, d) where $A \subseteq G$ and $d \in D$ such that $A^\diamond = d$ and $d^\diamond = A$, A is called the pattern extent and d is called the pattern intent.*

As in the standard case of FCA, a pattern concept corresponds to the maximal set of objects A whose description subsumes the description d, while there is no $e \in D$, subsuming d, i.e. $d \sqsubseteq e$, describing every object in A. The set of all concepts can be partially ordered w.r.t. partial order on extents (dually, intent patterns, i.e $\sqsubseteq$), within a concept lattice.

It is wroth mentioning, that the size of the concept lattice can be exponential w.r.t. to the number of objects, and, thus, we need a special ranking method to select the most interesting concepts for further analysis. Several such techniques are considered in [8], where it is shown that stability index [9] is more reliable in noisy data. Thus, we decided to use this index in the current work.

An example of pattern structures is given by Table 1a and described in the next sections, the corresponding lattice is depicted in Figure 1a.

2 Sequential Pattern Structures

2.1 An Example of Sequential Data

Patient	Trajectory
p^1	$\langle\{a\};\{c,d\};\{b,a\};\{d\}\rangle$
p^2	$\langle\{c,d\};\{b,d\};\{a,d\}\rangle$
p^3	$\langle\{c,d\};\{b\};\{a\};\{a,d\}\rangle$

(a) Sequential dataset.

	Subsequences		Subsequences
ss^1	$\langle\{c,d\};\{b\};\{d\}\rangle$	ss^5	$\langle\{a\};\{d\}\rangle$
ss^2	$\langle\{d\};\{a\}\rangle$	ss^6	$\langle\{a,d\}\rangle$
ss^3	$\langle\{c,d\};\{b\}\rangle$	ss^7	$\langle\{a\}\rangle$
ss^4	$\langle\{c,d\};\{b\};\{a\}\rangle$		

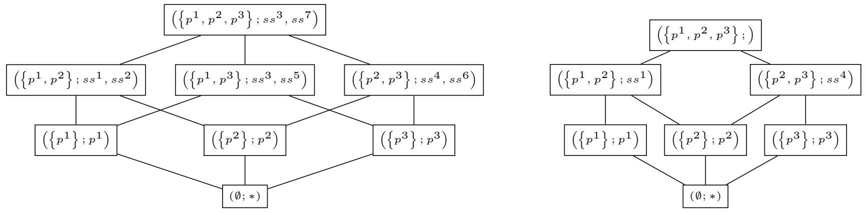
(b) Some common subsequences.

Table 1: A toy sequential dataset of patient medical trajectories.

A medical trajectory of a patient is a sequence of hospitalizations, where every hospitalization is described by a set of medical procedures the patient underwent. An example of medical trajectories for three patients is given in Table 1a. Patient p^1 had four hospitalizations and during the second hospitalization he underwent procedures c and d. Patients may have a different number of hospitalizations. Hereafter we use the following notation, different sequences are enumerated in superscript (p^1), while elements of a sequence are enumerated in the subscript ($p^1_2 = \{c,d\}$). One important task is to find the characteristic subsequences of hospitalizations for patients to optimize hospitalization processes. For example, we can find a strange sequence and, thus, motivate the deeper analysis of the problems behind or we can find typical sequences that allow us to estimate the treatment costs for a patient.

2.2 Partial Order on "Complex" Sequences

A sequence is constituted of elements from an alphabet. The classical subsequence matching task requires no special properties of the alphabet. Several generalization of the classical case were made by introducing subsequence relation based on itemset alphabet [10] or on multidimensional and multilevel alphabet [11]. Here, we generalize the previous cases, requiring for an alphabet



(a) Original Lattice.

(b) Projected Lattice.

Fig. 1: The concept lattices for the pattern structure given by Table 1a. Concept intents refers to sequences in Tables 1a and 1b.

to form a semilattice $(E, \sqcap_E)$[1]. This generalization allows one to process in a unified way all types of complex sequential data.

Definition 3. *A sequence $t = \langle t_1; ...; t_k \rangle$ is a subsequence of a sequence $s = \langle s_1; ...; s_n \rangle$, denoted $t \leq s$, iff $k \leq n$ and there exist $j_1, ..j_k$ such that $1 \leq j_1 < j_2 < ... < j_k \leq n$ and for all $i \in \{1, 2, ..., k\}$, $t_i \sqsubseteq_E s_{j_i}$.*

With complex sequences and such kind of subsequences the calculation procedures can be difficult, thus, to simplify the procedure, only "restricted" subsequences are considered, where only the order of consequent elements is taken into account, i.e. given a j_1 in Definition 3, $j_i = j_{i-1} + 1$ for all $i \in \{2, 3, ..., k\}$. Such a restriction makes sens for our data, because a hospitalization is a discrete event and it is likely that the next hospitalization has a relation with the previous one, for example, hospitalizations for treating aftereffects of chemotherapy. Below the word "subsequence" refers to a "restricted" subsequence.

Based on Definition 3 and on the alphabet $(\wp(P), \cap)$, the sequence ss^1 in Table 1b is a subsequence of p^1 because if we set $j_i = i + 1$ (Definition 3) then $ss^1_1 \sqsubseteq p^1_2$ ($\{c, d\} \subseteq \{c, d\}$), $ss^1_2 \sqsubseteq p^1_3$ ($\{b\} \subseteq \{b, a\}$) and $ss^1_3 \sqsubseteq p^1_4$ ($\{d\} \subseteq \{d\}$).

2.3 Meet-semilattice of Sequences

Using the previous definitions, we can precisely define the sequential pattern structure. For that, we make an analogy with the pattern structures for graphs [12] where the meet-semilattice operation $\sqcap$ respects subgraph isomorphism. Thus, we introduce a sequential meet-semilattice respecting subsequence relation. Given an alphabet lattice $(E, \sqcap_E)$, D consists of sets of sequences based on E, such that if $d \in D$ contains a sequence s then all subsequences of s should be included into d, $\forall s \in d, \nexists \tilde{s} \leq s : \tilde{s} \notin d$. Similarity operation is the set intersection for two sets of sequences. Given two patterns $d_1, d_2 \in D$, the set intersection operation ensures that if a sequence s belongs to $d_1 \sqcap d_2$ then any subsequence of s belongs to $d_1 \sqcap d_2$ and thus $(d_1 \sqcap d_2) \in D$. As the set intersection operation is idempotent, commutative and associative, $(D, \sqcap)$ is a valid semilattice.

[1] In this paper we consider two semilattices, the first one is on the characters of the alphabet, $(E, \sqcap_E)$, and the second one is introduced by pattern structures, $(D, \sqcap)$.

However, the set of all possible subsequences can be rather large. Thus, it is more efficient and representable to keep a pattern $d \in D$ as a set of all maximal sequences $\tilde{d}$, $\tilde{d} = \{s \in d \mid \nexists s^* \in d : s^* \geq s\}$. Below, every pattern is given only by the set of maximal sequences. For example, $\{p^2\} \sqcap \{p^3\} = \{ss^4, ss^6\}$ (see Tables 1a and 1b), i.e. $\{ss^4, ss^6\}$ is the set of maximal common subsequences between $\{p^2\}$ and $\{p^3\}$, correspondingly $\{ss^4, ss^6\} \sqcap \{p^1\} = \{ss^3, ss^7\}$. Moreover, representing a pattern by the set of maximal sequences allows for an efficient implementation of the intersection "$\sqcap$" (see Section 3.1 for more details).

The sequential pattern structure for the example given by Subsection 2.1 is $(G, (D, \sqcap), \delta)$, where $G = \{p^1, p^2, p^3\}$, $(D, \sqcap)$ is the semilattice of sequential descriptions, and δ is the mapping associating an object in G to a description in D shown in Table 1a. Figure 1a shows the resulting lattice of sequential pattern concepts for this particular pattern structure $(G, (D, \sqcap), \delta)$.

2.4 Projections of Sequential Pattern Structures

Pattern structures can be hard to process due to the usually large number of concepts in the concept lattice and the complexity of the involved descriptions and the similarity operation. Moreover, a given pattern structure can produce a lattice with a lot of patterns which are not interesting for an expert. *Can we save computational time by deleting some unnecessary patterns?* Projections of pattern structures "simplify" to some degree the computation and allow one to work with a reduced description. In fact, projections can be considered as constraints (or filters) on patterns respecting certain mathematical properties. These mathematical properties guarantee that the projection of a lattice is a lattice where projected concepts have certain correspondence to original ones. We introduce projections on sequential patterns, adapting them from [7].

A projection $\psi : D \to D$ is an operator, which is monotone ($x \sqsubseteq y \Rightarrow \psi(x) \sqsubseteq \psi(y)$), contractive ($\psi(x) \sqsubseteq x$) and idempotent ($\psi(\psi(x)) = \psi(x)$). A projection preserves the semilattice operation $\sqcap$ as follows. Under a projection ψ, the pattern structure $(G, (D, \sqcap), \delta)$ becomes the projected pattern structure $\psi((G, (D, \sqcap), \delta)) = (G, (D_\psi, \sqcap_\psi), \psi \circ \delta)$, where $D_\psi = \psi(D) = \{d \in D \mid \exists d^* \in D : \psi(d^*) = d\}$ and $\forall x, y \in D, x \sqcap_\psi y := \psi(x \sqcap y)$. The concepts of a projected pattern structure have a "similar" concept in the initial pattern structure [7].

One possible projection for sequential pattern structures comes from the following observation. In many cases it can be more interesting to analyze quite long common subsequences rather than small ones. For example, if we prefer common subsequences of length > 2, then between p^1 and p^2 in Table 1a there is only one maximal common subsequence, ss^1 in Table 1b, while ss^2 in Table 1b is too short to be considered as a common subsequence. Such kind of projections we call *Minimal Length Projection* (MLP) and depends on the parameter l, the minimal allowed length of the sequences in a pattern. The projected pattern concept lattice for MLP ≥ 3 is shown in Figure 1b. In the experimentation section we compare MLP projections with different value of the parameter. Projections are very useful, as they reduce the computational costs in a meaningful manner.

3 Sequential Pattern Structure Evaluation

3.1 Implementation

Nearly all state-of-the-art FCA algorithms can be adapted to process pattern structures. We adapted `AddIntent` algorithm [13], as the lattice structure is important for us to calculate stability (see the algorithm for calculating stability in [14]). To compute the semilattice operation ($\sqcap$, $\sqsubseteq$) between two sets of sequences $S = \{s^1, ...s^n\}$ and $T = \{t^1, ..., t^m\}$, $S \sqcap T$ is calculated according to Section 2.3, i.e. maximal sequences among all maximal common subsequences for any pair of s^i and t^j. To find all common subsequences of two sequences, the following observations is useful. If $ss = \langle ss_1; ...; ss_l\rangle$ is a subsequence of $s = \langle s_1; ...; s_n\rangle$ with $j^s_i = k^s + i$ (Definition 3: k^s is the index difference from which ss is a subsequence of s) and a subsequence of $t = \langle t_1; ...; t_m\rangle$ with $j^t_i = k^t + i$ (likewise), then for any index $i \in \{1, 2, ..., l\}$, $ss_i \sqsubseteq_E (s_{j^s_i} \sqcap t_{j^t_i})$. Thus, to find maximal common subsequences between s and t, we, first, align s and t in all possible ways, and then we compute the resulting intersections and keep only the maximal ones. Let us consider two possible alignments of s^1 and s^2:

$$\begin{array}{rllllll|rlllll}
s^1 = & \langle\{a\}; & \{c,d\}; & \{b,a\}; & \{d\} & & \rangle & s^1 = & \langle\{a\}; & \{c,d\}; & \{b,a\}; & \{d\} & \rangle \\
s^2 = & & & \langle\{c,d\}; & \{b,d\}; & \{a,d\} & \rangle & s^2 = & & \langle\{c,d\}; & \{b,d\}; & \{a,d\} & \rangle \\
ss^l = & & & \langle\ \ \emptyset\ \ ; & \{d\} & & \rangle & ss^r = & & \langle\{c,d\}; & \{b\}\ ; & \{d\} & \rangle
\end{array}$$

The left intersection ss^l is smaller than ss^r and, thus, is not kept.

3.2 Experiments and Discussion

The experiments are carried out on an "Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz" computer with 8Gb of memory under the Ubuntu 12.04 operating system. The algorithms are not parallelized and are coded in C++.

First, the public available dataset from UCI repository on anonymous web data is used as a benchmark data set for scalability tests. This dataset contains around 10^6 transactions, and each transaction is a sequence based on "simple" alphabet, i.e. with no order on the elements. The overall time changes from 37279 seconds for the sequences of length $MLP \geq 5$ upto 97042 seconds for the sequences of length $MLP \geq 3$. For more details see the web-page.[2]

Our use-case data set comes from a French healthcare system [15]. In the experiment, 1000 patients are analyzed. The dataset describes a patient as a sequence of hospitalizations without any timestamps. A hospitalization for a patient is a tuple with the hospital name, the cause for the hospitalization and the set of procedures the patient underwent. All the field of a hospitalization are joined into one single set, which describes one element of a sequence.

Table 2 shows the final times and the lattice sizes for different projections. The calculation time with projections changes from 3120 to 4510 seconds depending on the projection parameter. The lattice sizes for the different projections

[2] `http://www.loria.fr/~abuzmako/FCA4AI2013/experiment-uci.html`

Projection	No	$l = 2$	$l = 3$	$l = 4$	$l = 5$	$l = 6$
Time(s)	$> 1.4 * 10^5$	4510	3878	3722	3435	3120
Lattice	$> 2.8 * 10^6$	554332	271166	189353	137912	100492

Table 2: The processing time and the lattice size for the PMT dataset.

	Intent	Extent
c_1	$\langle\{D_{\text{Chemotherapy}}\} * 8\rangle$	284 (28%)
c_2	$\langle\{D_{\text{Device Adj.}}, P_{\text{Artery Catheter}}\}; \{D_{\text{Chemotherapy}}\} * 12\rangle$	74 (7%)
c_3	$\langle\{P_{\text{Chest Radiography}}\} * 2\rangle$	189 (19%)

Table 3: Some interesting concepts of the PMT database

changes from 10^5 to $5 \cdot 10^5$. Notice that the ratio for the lattice size is of 5 while the ratio for computation time is 1.5. Deletion of short sequences can dramatically change the lattice size, since the shorter a sequence is, the more probable it is a subsequence of a patient trajectory, but the computation of the semilattice operation is easily processed with shorter sequences. The calculation without projection takes a lot of time with relatively small lattice size (40 times more for the runtime and 6 times more for the lattice size w.r.t the projection $l = 2$). The reason for that is memory swapping to the hard disk. Thus, the best projection for that dataset is $l = 2$ as its computation time is reasonable and it preserves the most of the information among the other projections.

Table 3 shows some interesting concept intents and the sizes of the corresponding extents. The concept are selected among the most stable ones [9]. The concept c_1 corresponds to 28% of the patients having at least 8 consequent hospitalizations because of chemotherapy. Now we can estimate the minimal cost of the overall procedures for a patient (we know the price of chemotherapy and we know the expected number of procedures). Concept c_2 covers 7% of the patients, and its intent is more interesting: 12 hospitalizations for chemotherapy following the hospitalization for adjustment of a chemotherapy device and an artery catheter installation. Both concepts c_1 and c_2 can be found within any of the considered projections (with $l \in \{2, 3, .., 6\}$), while the concept c_3 can be found only within the most specific projection ($l = 2$). The concept c_3 covers 19% of the patients and describes patients with at least two consequent hospitalizations accompanied with a chest radiography. In this way, we have a kind of control of the trajectory quality, because we could have an under-examination of a patient during the first hospitalization.

Conclusion

In this paper, we present an approach for analyzing sequential datasets within the framework of pattern structures, an extension of FCA dealing with complex (non binary) data. Using pattern structures leads to the construction of a pattern concept lattice, which does not require the setting of a support threshold, as usually needed in sequential pattern mining. Another point worth to be noticed is that the use of projections gives a lot of flexibility especially for mining and interpreting special kinds of patterns. The framework is tested on a real-life dataset recording patient trajectories in a healthcare system. Interesting patterns are extracted and then interpreted, showing the feasibility and usefulness of the approach. For the future work, it is important to more deeply investigate projections, their potentialities w.r.t. the types of patterns.

Acknowledgements: this research received funding from the Basic Research Program at the National Research University Higher School of Economics (Russia) and from the BioIntelligence project (France).

References

1. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.: FreeSpan: frequent pattern-projected sequential pattern mining. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. (2000) 355–359
2. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In: 17th International Conference on Data Engineering. (2001) 215–226
3. Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Databases. In: In SDM. (2003) 166–177
4. Ding, B., Lo, D., Han, J., Khoo, S.C.: Efficient Mining of Closed Repetitive Gapped Subsequences from a Sequence Database. In: 2009 IEEE 25th International Conference on Data Engineering, IEEE (March 2009) 1024–1035
5. Raïssi, C., Calders, T., Poncelet, P.: Mining conjunctive sequential patterns. Data Min. Knowl. Discov. **17**(1) (2008) 77–93
6. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. 1st edn. Springer, Secaucus, NJ, USA (1997)
7. Ganter, B., Kuznetsov, S.O.: Pattern Structures and Their Projections. In Delugach, H., Stumme, G., eds.: Conceptual Structures: Broadening the Base SE - 10. Volume 2120 of LNCS. Springer Berlin Heidelberg (2001) 129–142
8. Klimushkin, M., Obiedkov, S.A., Roth, C.: Approaches to the Selection of Relevant Concepts in the Case of Noisy Data. In: Proceedings of the 8th international conference on Formal Concept Analysis. ICFCA'10, Springer (2010) 255–266
9. Kuznetsov, S.O.: On stability of a formal concept. Annals of Mathematics and Artificial Intelligence **49**(1-4) (2007) 101–115
10. Casas-Garriga, G.: Summarizing Sequential Data with Closed Partial Orders. In: SDM. (2005)
11. Plantevit, M., Laurent, A., Laurent, D., Teisseire, M., Choong, Y.W.: Mining multidimensional and multilevel sequential patterns. ACM Transactions on Knowledge Discovery from Data **4**(1) (January 2010) 1–37
12. Kuznetsov, S.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. Volume 1704 of LNCS. Springer (1999) 384–391
13. Merwe, D.V.D., Obiedkov, S., Kourie, D.: AddIntent: A new incremental algorithm for constructing concept lattices. In Goos, G., Hartmanis, J., Leeuwen, J., Eklund, P., eds.: Concept Lattices. Volume 2961. Springer (2004) 372–385
14. Roth, C., Obiedkov, S., Kourie, D.G.: On succinct representation of knowledge community taxonomies with formal concept analysis A Formal Concept Analysis Approach in Applied Epistemology. International Journal of Foundations of Computer Science **19**(02) (April 2008) 383–404
15. Fetter, R.B., Shin, Y., Freeman, J.L., Averill, R.F., Thompson, J.D.: Case mix definition by diagnosis-related groups. Med Care **18**(2) (February 1980) 1–53

Using pattern structures to support information retrieval with Formal Concept Analysis

Víctor Codocedo[1]*, Ioanna Lykourentzou[1,2]**, Hernán Astudillo[3], and Amedeo Napoli[1]

[1] LORIA - CNRS - INRIA - Univesité de Lorraine, BP 239, 54506 Vandœuvre-les-Nancy.
victor.codocedo@loria.fr, amedeo.napoli@loria.fr,
[2] Centre de Recherche Public Henri Tudor - 29, avenue John F. Kennedy L-1855 Luxembourg-Kirchberg, Luxembourg
ioanna.lykourentzou@tudor.lu
[3] Universidad Técnica Federico Santa María - Avenida España 1680 - Valparaíso, Chile
hernan@inf.utfsm.cl

Abstract. In this paper we introduce a novel approach to information retrieval (IR) based on Formal Concept Analysis (FCA). The use of concept lattices to support the task of document retrieval in IR has proven effective since they allow querying in the space of terms modelled by concept intents and navigation in the space of documents modelled by concept extents. However, current approaches use binary representations to illustrate the relations between documents and terms ("document D contains term T") and disregard useful information present in document corpora ("document D contains X references to term T"). We propose using pattern structures, an extension of FCA on multi-valued and numerical data, to address the above. Given a set of weighted document-term relations, a concept lattice based on pattern structures is built and explored to find documents satisfying a given user query. We present the meaning and capabilities of this approach, as well as results of its application over a classic IR document corpus.

Keywords: Formal Concept Analysis, Interval Pattern Mining, Information Retrieval

1 Introduction

Information retrieval (IR), is a problem of lasting interest for the research community. Among the tasks comprising the IR domain, document retrieval (i.e. the search and ranking of documents that are relevant to an original user query from a given document corpus) is one of the most popular in the field given its importance in everyday routines. In the wide spectrum of techniques applied to support document retrieval, formal concept analysis (FCA) has gained interest in the last years [3–6, 16] because of its robust framework and the qualities of a concept lattice.

Formal concept analysis (FCA) is a mathematical formalism used for data analysis and classification, which relies on the dualistic understanding of concepts as consisting

* Part of the Quaero Programme, funded by OSEO, French State agency for innovation.

** Supported by the National Research Fund, Luxembourg, and cofunded under the Marie Curie Actions of the European Commission (FP7-COFUND).

of an extent (the objects that belong to the concept) and an intent (the attributes that those objects share) organized in a lattice structure called a concept lattice [7]. We refer to FCA-based information retrieval as CL4IR which stands for“concept lattices for information retrieval”.

In a typical CL4IR approach, a binary table of documents and terms is created and then, using FCA algorithms, the respective concept lattice is created. This lattice contains several *formal concepts*, each defined by a set of documents (extent) and the set of terms that they share (intent). Thus, the lattice provides a multiple hierarchical classification of documents and terms which can be navigated and searched as an index, to retrieve concepts that are “close” or “similar” to the original query concept. In this way a CL4IR system exploits the connections of concepts within the lattice to find relevant documents for a given query and takes advantage of the lattice structure to enrich the answer in different ways: by navigating the lattice to look for approximate answers through generalizations and specifications of the original query concept’s intent [2, 14], by enriching the term vocabulary with a thesaurus [5] or by directly integrating external knowledge sources [16]. Nevertheless, current CL4IR systems are restricted by the binary nature of their data (a document can either contain a given term or not). Consequently, they can work only with Boolean-like queries, which is an important limitation w.r.t. other IR approaches such as vector-space ranking methods [13] that allow partial-matching documents to be considered as possible answers.

In this article we present a novel CL4IR approach, which deals with numerical datasets, i.e. document-term relations, where a document is annotated by a term with a certain weight. This approach provides CL4IR systems with an extended *query space*, on which vector-space ranking methods can be adapted and applied. In parallel, this approach retains the main advantage of using lattices as the document search index, which is the provision and exploration potential of the *complete query space*. Our approach is based on the pattern structures framework, an extension of FCA to deal with complex data [7]. Given a numerical table representing weighted associations between documents and terms, we apply pattern structures to build the extended query space, while we also introduce steps for reducing and simplifying the document search within the constructed query space. We illustrate our approach through running example on a classical IR dataset and by comparing our results, in terms of precision and recall, to those reported in the literature. Furthermore we provide a discussion on the meaning and capabilities of the proposed approach. The remainder of the paper is organized as follows. Section 2 provides an introduction to the use of formal concept analysis for document retrieval. Section 3 describes the pattern structure framework and details the proposed CL4IR approach which can be applied on numerical datasets. Section 4 presents the experiments. Finally, Section 5 concludes the paper.

2 Concept Lattices for Information Retrieval

The setting of a typical concept lattice for information retrieval (CL4IR) application is given by a formal context $\mathcal{K} = (D, T, I)$ made of a set of documents D, set of terms T and an incidence relation $I = \{(d_i, t_j)\}$ indicating that document d_i contains term t_j. Table 1 illustrates a document-term formal context created from a corpus of 9 documents and 12 terms.

	human	interface	computer	user	system	response	time	EPS	survey	tree	graph	minor
d_1	x	x	x									
d_2			x	x	x	x	x		x			
d_3		x		x	x			x				
d_4	x				x			x				
d_5				x		x	x					
d_6										x		
d_7										x	x	
d_8										x	x	x
d_9									x		x	x
q *										x	x	

* Grey row represents the *query.*

Table 1: A term-document formal context including the query q.

Given a user query $q = \{t_1, t_2 ... t_{|q|}\}$, the document retrieval task consists in returning a set of documents ordered by "relevance" w.r.t. the query q. In CL4IR systems a query can be represented as a virtual document containing the set of terms $\{t_1, t_2 ... t_{|q|}\}$. Then, the query is inserted in the formal context as another object and the incidence relation set I is updated to include the relations of the virtual query-document and its terms. The formal context becomes $\mathcal{K}_q = (D + \{q\}, T, I + \{(q, t_i)_{i..|q|}\})$.

The standard procedure to find "relevant" documents within the concept lattice consists in identifying the *query concept* (which is defined as the *object concept* of the virtual object q and denoted by $\gamma(q) = ((q')', q'))$ and concepts related to the *query concept* (for example, its superconcepts) which can provide further results. We refer to the later concepts as "answer concepts". For the formal context in Table 1, consider the query q with terms "graph" and "tree" (grey row). Figure 1 shows the concept lattice derived from this formal context (including the query). The *query concept* corresponds to concept 17 and contains in its extent documents d_7 and d_8 which satisfy the query and can be retrieved to the user. The superconcepts of the *query concept* (concepts 7 and 8) contain documents d_6 and d_9 which can also be retrieved. Different relevance measures can be used to rank the retrieved documents. For example, the topological distance within the lattice between the *query concept* and the "answer concepts" (i.e. concepts partially satisfying the query) can be calculated and in this case documents $d7$ and $d8$ are at distance 0 (more relevant), while d_6 and d_9 are at distance 1 (less relevant). Other such measures include semantic distance, extent intersection, and Jaccard similarity [2, 15, 5].

More generally, the concept lattice defines a *query space* where each formal concept C can be considered as a conjunctive Boolean query (i.e. a query where the constraint

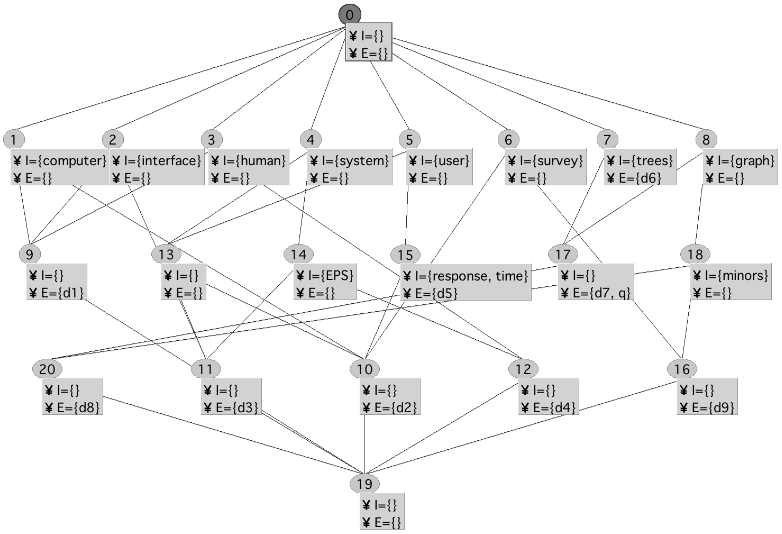


Fig. 1: Concept lattice in reduced notation derived from a document-term formal context including the query.

is given by the conjunction of the attributes in the intent of C) and a combination of formal concepts provides disjunction and negation (e.g. The union of concepts 7 and 8 in Figure 1 satisfies the disjunctive query "graph" or "tree"). Unfortunately, the binary case is the "ideal world". In most real-world datasets the relation between a document and a term is built w.r.t. a measure such as frequency, distance or weight involving a range of numerical values [13].

A document corpus can be defined as a term-document matrix $A = [a_{ij}]$, where terms $t_i \in T$ are in rows, documents $d_j \in D$ in columns and each cell a_{ij} of the matrix represents the "value" of the term t_i in the document d_j, given by a function $val(d_j, t_i)$ (weight, frequency, etc.). In order to work with this kind of datasets, a CL4IR system can resort to interordinal scaling [8] by simply assigning an incidence relation when a term in a document has a value within a given range, i.e. $I = \{(d,t) | val(d,t) > 0\}$). However, interordinal scaling could greatly increase the complexity for IR tasks [12] as it induces redundancy as shown in [11]. To the best of the authors' knowledge, a CL4IR system directly dealing with a weighted term-document dataset is not yet reported in the FCA nor in IR literature. In the following, we present a method and an implementation of a CL4IR approach dealing with numerical datasets.

3 CL4IR with many-valued datasets

3.1 Pattern structure framework

Here, we introduce the pattern structure framework firstly described in [7].

A pattern structure $\mathcal{K} = (G, (P, \sqcap), \delta)$ is a generalization of a formal context. In $\mathcal{K}$, G is a set of objects, $(P, \sqcap)$ is a semi-lattice of object descriptions and $\delta : G \rightarrow P$ is a mapping associating a description to an object. The description of an object $g \in G$ is a vector of intervals $v = \langle [l_i, r_i] \rangle_{i \in \{1..|M|\}}$, where $v \in P$, $l_i, r_i \in \mathbb{R}$ and $l_i \leq r_i$.

In $(P, \sqcap)$ the *similarity* operator $\sqcap$ applied to $v_a = \langle [l_i^1, r_i^1] \rangle$ and $v_b = \langle [l_i^2, r_i^2] \rangle$ yields the convex hull $v_a \sqcap v_b = \langle [min(l_i^1, l_i^2), max(r_i^1, r_i^2)] \rangle$ where $i \in \{1..|M|\}$. The associated subsumption relation is defined as $v_a \sqcap v_b = v_a \iff v_a \sqsubseteq v_b$.

A Galois connection between $\wp(G)$ (powerset of G) and $(P, \sqcap)$ is defined as follows:

$$X^{\square} = \textstyle\bigsqcap_{g \in X} \delta(g) \; ; \; v^{\square} = \{g \in G | v \sqsubseteq \delta(g)\}$$

where $X^{\square}$ represents the common description to all objects in X while $v^{\square}$ represents the set of objects respecting the description v. A pair (X, v) such as $X^{\square} = v$ and $v^{\square} = X$ is called a *interval pattern concept (ip-concept)* with extent X and pattern intent v. Ip-concepts can be ordered in an interval pattern concept lattice (ip-concept lattice). Algorithms for computing ip-concepts from an interval pattern structure are proposed in [11, 7].

3.2 CL4IR based on pattern structures

A document corpus or a term-document matrix, as described at the end of Section 2, can naturally be represented as a many-valued context [8] $\mathcal{K} = (D, T, W, I)$, where $W = \{val(d_j, t_i)\}_{\forall d_j \in D, t_i \in T}$ and $I = (d_j, t_i, w_k); f(d_j, t_i) = w_k, w_k \in W$. Table 2 shows an example containing 9 documents (white rows) and 12 terms. The value in a cell represents the "relative frequency" of a term in a document, i.e. the ratio between the amount of times a term appears in a document and the total amount of terms occurrences in the document. Like in the binary case, a query $q = \{t_1, t_2, ..t_{|q|}\}$ is considered as a virtual document and included in the many-valued context which becomes $\mathcal{K}^q = (D + \{q\}, T, W, I + \{val(q, t_i)\}_{\forall t_i \in q})$. The cells of the query contain also a "relative frequency" value. The query $q = \{$*"graph", "tree"*$\}$ is illustrated in the grey row in Table 2 (e.g. $val(q, graph) = 1/2 = 0.5$).

To deal with $\mathcal{K}^q$, we define the pattern structure as $(D + \{q\}, (P, \sqcap), \delta)$ where interval patterns in P contain the interval-vector representation of documents in $|T|$ dimensions (one for each term). The mapping $\delta(d) = \langle [val(d, t_i), val(d, t_i)]_{i \in [1..|T|]} \rangle$ assigns an interval pattern representation to a document (or the virtual query-document) consisting of a zero-length interval for each term existing in T at the value of the term in the document (e.g. in Table 2, $\delta(d_1) = \langle [0.33, 0.33][0.33, 0.33][0.33, 0.33][0, 0] \dots [0, 0] \rangle$, where $[0, 0]$ is represented by $[-]$). The similarity operator $\sqcap$ applied to two interval patterns returns the convex hull between their document representations. From the pattern structure we construct the ip-concept lattice representing the *query space* which will be used to retrieve documents in a similar way as binary approaches.

The *query concept* is still considered as the *object concept* of q. However the semantic of the *query space* changes. While in the binary case the *query space* represents a pool of Boolean query possibilities, here the *query space* can be considered as a vector space where the query is grouped with documents having similar representations. For example, consider the first three columns in Table 3 where each row represents an ip-concept. Concept 1 is the *query concept* which in its extent includes documents d_7

and its interval pattern (intent) only includes zero-length intervals in all 12 dimensions, making the description of the query identical to the description of d_7. Concept 2 is a superconcept of 1, whose extent contains d_7 and d_8. This time, there are only 9 zero-length intervals in all 12 dimensions. Concept 2 is less similar to the query than concept 1 w.r.t 3 dimensions. Following with concept 3, we can see that the later is less similar to the query than concept 1 w.r.t. 4 dimensions. We get in this way a “natural” ranking of the concepts.

In order to rank ip-concepts we rely on the notion of *maximal distance* within an interval pattern. For illustrating this notion, we will use the geometrical interpretation of patterns already introduced in [11]. Let us consider the 2-dimensional case with two ip-concepts in Figure 2, namely $Z_1 = (\{q, A, B, C\}, \langle[2, 7][2, 7]\rangle)$ (clear rectangle) and $Z_2 = (\{q, A, B, C, D\}, \langle[1, 7][1, 7]\rangle)$ (dark rectangle). For ranking an ip-concept Z_i w.r.t. the query, we will consider the “maximal distance” possible between any two objects in the extent of Z_i, which in the case of Z_1 is between objects q and C and for Z_2 is between q and D. Thus, this distance is actually the Euclidean distance between the edges of the interval vector. Table 3 presents the retrieved ip-concepts for the query in Table 2 ranked by *maximal distance* in column 4.

	human	interface	computer	user	system	response	time	EPS	survey	tree	graph	minor
d_1	0.33	0.33	0.33	0	0	0	0	0	0	0	0	0
d_2	0	0	0.16	0.16	0.16	0.16	0.16	0	0.16	0	0	0
d_3	0	0.25	0	0.25	0.25	0	0	0.25	0	0	0	0
d_4	0.25	0	0	0	0.5	0	0	0.25	0	0	0	0
d_5	0	0	0	0.33	0	0.33	0.33	0	0	0	0	0
d_6	0	0	0	0	0	0	0	0	0	1	0	0
d_7	0	0	0	0	0	0	0	0	0	0.5	0.5	0
d_8	0	0	0	0	0	0	0	0	0	0.33	0.33	0.33
d_9	0	0	0	0	0	0	0	0	0.33	0	0.33	0.33
q [a]	0	0	0	0	0	0	0	0	0	0.5	0.5	0

Table 2: Many-valued document term context (including query).

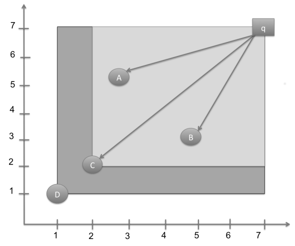


Fig. 2: Two interval patterns in the *query space*.

[a] Grey row represents the *query concept.*

Id	Extent	Pattern intent	max_dist
1	q, d_7 *	$\langle[-][-][-][-][-][-][-][-][-][0.5, 0.5][0.5, 0.5][-]\rangle$	0
2	q, d_7, d_8	$\langle[-][-][-][-][-][-][-][-][-][0.33, 0.5][0.33, 0.5][0, 0.33]\rangle$	0.408
3	q, d_7, d_8, d_9	$\langle[-][-][-][-][-][-][-][-][0, 0.33][0, 0.5][0.33, 0.5][0, 0.33]\rangle$	0.704
4	q, d_6, d_7	$\langle[-][-][-][-][-][-][-][-][-][0.5, 1][0, 0.5][-]\rangle$	0.707
5	q, d_2, d_7	$\langle[-][-][0, 0.16][0, 0.16][0, 0.16][0, 0.16][0, 0.16][-][0, 0.16][0, 0.5][0, 0.5][-]\rangle$	0.808
6	q, d_3, d_7	$\langle[-][0, 0.25][-][0, 0.25][0, 0.25][-][-][0, 0.25][-][0, 0.5][0, 0.5][-]\rangle$	0.866
7	q, d_1, d_7	$\langle[0, 0.33][0, 0.33][0, 0.33][-][-][-][-][-][-][0, 0.5][0, 0.5][-]\rangle$	0.909
8	q, d_5, d_7	$\langle[-][-][-][0, 0.33][-][0, 0.33][0, 0.33][-][-][0, 0.5][0, 0.5][-]\rangle$	0.909
9	q, d_4, d_7	$\langle[0, 0.25][-][-][-][0, 0.5][-][-][0, 0.25][-][0, 0.5][0, 0.5][-]\rangle$	0.935

* Grey row represents the *query concept.*

Table 3: Extents and Intents of concepts in Figure 2 presenting the cosine similarity between its edges ($[-]$ represents the zero-length interval $[0, 0]$).

3.3 Dealing with real-world datasets

Calculating a concept lattice is an expensive task which can yield a large amount of concepts making it prohibitive for large document corpora. The scenario is worst for pattern structures since for every concept the size of the intent is set to the whole set of attributes adding even more complexity. Calculating the whole *query space* of a term-document matrix is not advisable, since for a given query only a small part of the whole space is required. In order to avoid a sizeable *query space* in each step of the retrieval process, progressive actions to filter data are performed. In the following, we describe the retrieval process and each action.

1. Constructing the pattern structure: The process starts with the input of a query $q = \{t_1, t_2, ..., t_{|q|}\}$ and ends after the pattern structure containing the virtual query-document is created. We include in the set of documents only those which contain at least a given number of the terms provided in the query, which can be performed at a negligible cost by firstly storing documents and terms in a relational database. The set of terms only include those provided in the query. The minimum number of terms for a document is left as a parameter of the process.

2. Constructing the ip-concept lattice: This step receives the pattern structure in order to create an ip-concept lattice. A standard FCA algorithm, namely Ganter's algorithm [8] is used for this purpose. However the algorithm has been adapted for the present task.

Many ip-concepts found in the interval pattern lattice are not useful for document retrieval purposes. For example, the framework creates ip-concepts with documents which do not share terms (e.g. consider the interval $\langle[0,1][0,1][0,1]\rangle$ created from the documents sharing no terms with orthogonal representations $v_1 = \langle[0,0][1,1][1,1]\rangle$ and $v_2 = \langle[1,1][0,0][0,0]\rangle$). We denominate these concepts *non-informational*.

In order to reduce the amount of *non-informational* concepts, we modified the $\sqcap$ operator in the set of ordered patterns $(P, \sqcap)$ such as $[l,r] \sqcap [0,0] = [*]$ and $[l,r] \sqcap [*] = [*]; \forall l, r \in \mathbb{R}$. The interval $[*]$ has been used before to indicate absence of similarity [10]. Let $Z_i = (X_i, v_i)$ be an ip-concept, then $\rho(Z_i)$ represents the number of intervals different from $[*]$ in v_i. We call $\rho(Z_i)$ the dimensionality of Z_i. For a second ip-concept $Z_j = (X_j, v_j)$ is easy to show that $(Z_i \leq Z_j \iff v_j \sqsubseteq v_i) \implies \rho(Z_i) \leq \rho(Z_j)$. We use a threshold of minimal dimensionality (min_dim) to reduce the amount of ip-concepts calculated. Consider this analogous to the use of a minimal support in the construction of an iceberg lattice [18].

4 Experiments and Discussion

To test the validity of the proposed approach, we applied it on a popular IR dataset which is openly available. We refer to this implementation as "ip-CL4IR". The CISI dataset[4] consists of 1460 documents and 35 queries, each one containing a set of valid answers. Documents contain text in natural language and queries are given as a set of terms connected by Boolean operators. In our experiments, we converted documents to collections of weighted terms and stored them in a relational database. The weighting

[4] http://ftp.cs.cornell.edu/pub/smart/cisi/

measure used was term frequency-inverse document frequency (*tf.idf*) [13]. Boolean operators in the query were ignored since they do not provide meaning in the vector-space model (except in the extended Boolean model case [17] not considered in this work). The virtual query-document was constructed using the inverse document frequencies calculated from the dataset for each of its terms.

After receiving a query, ip-CL4IR consults the database and extracts all documents that contain at least 2 terms of the query (as described in Section 3.3, this value is a parameter of ip-CL4IR). The ip-concept lattice is computed using a minimal dimensionality of $min\_dim = 2$, to keep consistency w.r.t. the restriction given for the creation of the pattern structure. The *query concept* is searched in the lattice and its superconcepts are retrieved and ranked using the Euclidean distance between the boundaries of their interval patterns. Cosine distance (instead of Euclidean distance) was also calculated showing better results. Table 4 shows the results for 11-point precision of fixed recall and 6 measures of precision for the top 5,10 and 20 ranked documents retrieved. Results on an implementation based on concept lattice-based ranking (CLR) [2] using the same dataset and a simple binnarization of the relation document-term is reported along with our results for comparison purposes.

	ip-CL4IR	CLR	EM
11-point IAP[a]	**0.232**	0.191	0.174
MAP[b]	**0.202**	0.163	0.145
Precision@5	0.257	0.206	**0.285**
Precision@10	0.251	0.174	**0.257**
Precision@20	**0.245**	0.174	0.207
Recall@5	0.032	0.049	**0.057**
Recall@10	0.060	0.073	**0.079**
Recall@20	**0.146**	0.112	0.123

Table 4: CISI dataset. Results for 35 queries.

[a] Interpolated average precision
[b] Mean average precision

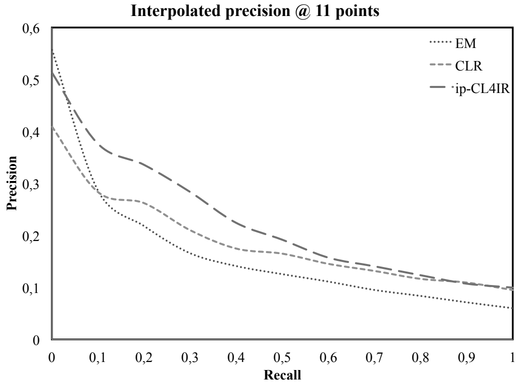


Fig. 3: Interpolated precision in 11 points of recall

Table 4 reports the results in 8 measures for ip-CL4IR, a reported CL4IR system called concept lattice-based ranking (CLR) [2] and a naive approach called exact matching (EM) where documents are ranked according to how many terms have in common w.r.t. the query. The second row contains the values of the interpolated average precision (IAP) over 11-points of recall illustrated on Figure 3. Interpolated precision in a given recall point r_i in Figure 3 indicates the best precision value in the interval $[r_i, r_{i+1}[$. From Figure 3, the interpolated precision in the recall point 0 for ip-CL4IR is the best precision obtained in the recall interval $[0, 1[$ equal to 0.51. The third row contains the values of the mean average precision (MAP) calculated over the precision values for each valid document found in the ranked documents retrieved by a system for each query. For example, given query if the first valid document is found in the third position of the ranking it has a precision value of 0.3. If the second is found in the fifth position its precision is 0.2 and the MAP is 0.25. IAP and MAP are standard information retrieval measures [13] to evaluate ranked results from a retrieval system. The remaining rows present values of precision and recall in the first 5 (@*5*), 10 (@*10*) and 20 (@*20*) ranked documents from each system. Boldface entries indicate the best values for the three systems.

Values in Table 4 show a better performance of ip-CL4IR on 4 of the 8 measures while EM is better in the remaining 4, namely precision and recall in the first 5 and 10 ranked documents. This indicates that EM is actually better to recognize documents very close to the query, but for documents with less elements in common with the query, EM is not very precise. This can be better appreciated in Figure 3 where the interpolated precision values of ip-CL4IR quickly overcome those of EM which is only better in 1 of the 11 recall points. This fact is also supported by the significant difference in the values of IAP and MAP between ip-CL4IR and EM. For the 35 queries in the dataset, our approach took 42.23 seconds (1.2 seconds per query) to execute while for CLR took 1550.333 (44.29 seconds per query) showing an impressive enhancement in the computational time required to retrieve documents, a key issue in document retrieval. Both these times include lattice construction. Using better measures which consider the correlation among terms, or including external knowledge sources like term taxonomies may improve greatly the quality of the answers provided by our approach. These issues are currently planned as future work. These experiments were performed in an Intel Xeon machine running at 2.27 GHz with 62 GB of RAM memory.

There are many perspectives for our approach, however the most important is the full exploitation of the ip-lattice structure to improve the quality in the answers. While our principal goal in this article is to describe a general process to directly support numeric term-document datasets in a concept lattice-based information retrieval system, we argue that different IR tasks (some already supported on CL4IR systems) can be also supported on ip-CL4IR for example, document clustering [1], user feedback inclusion [6] and recommendation [9].

5 Conclusions

In this article we introduce a CL4IR approach which is able to deal directly with numerical datasets through the use of the pattern structure framework (ip-CL4IR). We provide a method and a process to construct an interval pattern concept lattice (ip-concept lattice) which can be used as a document index. We present the idea of an ip-concept lattice as a *query space* which can be navigated in order to find relevant documents. We also provide means to rank these documents using vector-based distances. The feasibility of our approach is validated through its application on a popular IR dataset for which we present precision and recall values contrasted to those reported in the literature showing a better performance in the overall list of ranked documents and an impressive enhancement in the time needed to answer a single query.

The perspectives for our approach are numerous, ranging from the improvement of its answers, its application on different real-world datasets, but most importantly, the full exploitation of the lattice structure to support different IR tasks.

References

1. C. Carpineto, S. Osiński, G. Romano, and D. Weiss. A survey of Web clustering engines. *ACM Computing Surveys*, 41(3):1–38, July 2009.
2. C. Carpineto and G. Romano. Order theoretical ranking. *Journal of the American Society for Information Science*, 51(7):587–601, 2000.
3. C. Carpineto and G. Romano. Exploiting the potential of concept lattices for information retrieval with CREDO. *Journal of Universal Computer Science*, 10:985 – 1013, 2004.
4. C. Carpineto and G. Romano. Using Concept Lattices for Text Retrieval and Mining. *Formal Concept Analysis*, pages 161–179, Jan. 2005.
5. V. Codocedo, I. Lykourentzou, and A. Napoli. Semantic querying of data guided by Formal Concept Analysis. In *Formal Concept Analysis for Artificial Intelligence Workshop at ECAI 2012*, 2012.
6. S. Ferré. Camelis: a logical information system to organise and browse a collection of documents. *International Journal of General Systems*, 38(4):379–403, 2009.
7. B. Ganter and S. O. Kuznetsov. Pattern Structures and their projections. *Conceptual Structures: Broadening the Base*, 2001.
8. B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Dec. 1999.
9. D. I. Ignatov and S. O. Kuznetsov. Concept-based Recommendations for Internet Advertisement. *CoRR*, abs/0906.4, 2009.
10. M. Kaytoue, Z. Assaghir, A. Napoli, and S. O. Kuznetsov. Embedding tolerance relations in formal concept analysis. In *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10*, page 1689, New York, New York, USA, Oct. 2010. ACM Press.
11. M. Kaytoue, S. O. Kuznetsov, and A. Napoli. Revisiting numerical pattern mining with formal concept analysis. *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two*, pages 1342–1347, Nov. 2011.
12. S. O. Kuznetsov. Pattern Structures for Analyzing Complex Data. In *Proceedings of the 12th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, volume 5908 of *Lecture Notes in Computer Science*, pages 33–44. Springer Berlin Heidelberg, Dec. 2009.
13. C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press (Online edition), 1 edition, 2009.
14. N. Messai, M.-D. Devignes, A. Napoli, and M. Smaïl-Tabbone. Querying a bioinformatic data sources registry with concept lattices. In *Proceedings of the 13th international conference on Conceptual Structures: common Semantics for Sharing Knowledge*, volume 3596 of *Lecture Notes in Computer Science*, July 2005.
15. N. Messai, M.-D. Devignes, A. Napoli, and M. Smaïl-Tabbone. Using Domain Knowledge to Guide Lattice-based Complex Data Exploration. In *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 847–852, 2010.
16. U. Priss. Lattice-based Information Retrieval. *Knowledge Organization*, 27:132 – 142, 2000.
17. G. Salton, E. A. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, Nov. 1983.
18. G. Stumme, R. Taouil, Y. Bastide, and L. Lakhal. Conceptual clustering with iceberg concept lattices. In *Proc. GI-Fachgruppentreffen Maschinelles Lernen (FGML'01)*, Universität Dortmund 763, October 2001.

FCA-Based Concept Detection in a RosettaNet PIP Ontology

Jamel Eddine Jridi and Guy Lapalme

DIRO, Université de Montréal, Canada,
{jridijam, lapalme}@iro.umontreal.ca

Abstract. This paper presents an FCA-based methodology for concept detection in a flat ontology. We apply this approach to an automatically generated ontology for a RosettaNet Partner Interface Process (PIP) which does not take advantage of some important OWL semantic relations like `subClassOf`. The goal of our approach is to regroup ontology classes sharing a set of properties (Data and Object Property) in order to improve the quality, readability and the inheritance richness of a flat ontology.

Keywords: Formal Concept Analysis, Ontology, RosettaNet PIP Ontology, Inheritance Richness.

1 Introduction

Ontologies are widely used in knowledge management, information integration, natural language processing, information retrieval, business-to-business (B2B), e-commerce, etc [4].

Research is now envisioning the adoption of semantic web technologies in the business domain. Lytras et al. [8] analyse the practical requirements in terms of interoperability or knowledge representation. The use of ontologies is not only for communication between different applications but also to provide reasoning support to infer, integrate information and to extract meaning.

After an ontology is constructed, it is important to assess the quality of an ontology to detect design defects and to automatically recognize parts that cause problems and might need more work. In this paper, we try to improve the inheritance richness of an OWL ontology based on the balance between depth and height of the inheritance tree which, according to Tatir et al. [13], play a role in a quality assessment. In Software Engineering, Sheldon et al. [10] claim that when the hierarchy in the inheritance tree is shallow, better will be the maintainability, readability and understanding.

In this paper, we propose a method to verify, regroup concepts sharing properties to improve the quality and readability of an automatically generated ontology. We use Formal Concept Analysis in the context of concept detection in OWL ontologies using OWL artifacts (*Class*, *DataProperty* and *ObjectProperty*).

The remainder of this paper is structured as follows. Section 2 states the problem. In Section 3, we describe the principles of the Formal Concept Analysis algorithm underlying our approach and the adaptation of its principles for the detection of concepts in a RosettaNet Ontology. Section 4 presents and discusses the validation results. A summary of the related work in ontology-based attempts of B2B standards and the use of Formal Concept Analysis in Ontological Engineering is given in Section 5. We conclude and suggest future research directions in Section 6.

2 Background and Problem Statement

In this section, we describe the problem of concepts detection and the importance of the inheritance tree to represent a domain knowledge. We start by defining important notions. Then, we detail the specific problems that are addressed by our approach.

2.1 Basic notions

Ontology in Semantic Web technology provides a shared and common vocabulary for a domain of interest and a specification of the meaning of its terms [3]. It allows users to organize information into a hierarchy of concepts, to describe relationships between them, and to make semantics machine processable, not just readable by a human.

RosettaNet B2B Standard is a consortium which provides a global forum for suppliers, customers, and competitors to do business and collaboration in an efficient and profitable manner. To manage business activities, RosettaNet formalizes Partner Interface Processes (PIP) with either Data Type Definition (DTD) format or XML Schema, and define business processes between trading partners. PIPs are organized into eight groups of core business processes called clusters, themselves further grouped into segments. Each segment includes several PIPs. In section 3, we will use the PIP3A4 as running example of our methodology. But we managed to apply the same technique on all published PIPs.

The RosettaNet architecture contains a Business Dictionary to define the properties for basic business activities and Technical Dictionaries to provide properties for products [14]. The RosettaNet Implementation Framework (RNIF) describes the packaging, routing, and transport of all PIP messages and business signals.

2.2 Problem Statement

There are few works to measure quality of an ontology using metrics. But, to our knowledge, no work has yet proposed methods to verify, maintain concept hierarchy representation and improve the quality and readability of an ontology from inheritance point of view using OWL artifacts (*Class*, *DataProperty* and *ObjectProperty*).

Tatir et al. [13] and Sicilia et al. [11] propose an *Inheritance Richness* metric defined as the average number of subclasses per class and represent the distribution of information across different levels of the ontology inheritance tree. Values close to zero indicate horizontal ontologies representing perhaps more general knowledge while large values represent vertical ontologies describing detailed knowledge of a domain.

In our case study, we have chosen an ontology describing a detailed domain knowledge. Some approaches are proposed to ontologize some of the famous B2B standards like RosettaNet[1] and ebXML[2].

In this paper, we apply our methodology on a RosettaNet Ontology described in our paper published in the 15th International Conference on Enterprise Information Systems [5]. RosettaNet focuses on a supply chain domain and defines processes and business documents in detail. This automatically generated ontology, described in Section 4, has some drawbacks because it does not take advantage of some important OWL semantic relations like `subClassOf`. It is quite flat and does not describe in details the supply chain knowledge provided by RosettaNet, although it deals with a specific domain having several concepts with a common semantics. Although approaches have been previously proposed to ontologize a few B2B standards like RosettaNet and ebXML [2,6,7], we argue in [5] that our methodology is the first to deal with the complete set of RosettaNet PIPs.

During the transformation process, it is difficult to automatically detect elements with a common semantics which are described in separate OWL files. But, we noticed that many of those share some properties and lexemes in their compounded names. For this reason, we use FCA as a classification approach to detect concepts by regrouping classes in an ontology to improve its quality and readability.

3 FCA-Based Concept Detection Approach

The goal of our approach is to regroup ontology classes sharing a set of properties (Data and Object Property) and then maintain the hierarchy representations.

The core of our system has three main parts: Ontology Processing, FCA System and Regrouping concepts. The Ontology Processing step builds a cross-table from ontology artifacts (Class, Data Property and Object Property) without dealing with the property type or occurrence restrictions. This table describes relationships between objects (ontology classes are represented by rows) and attributes (Data and Object properties correspond to columns).

Taking the example in Table 1, $o_{1..k}$ represents ontology classes with k is the number of classes in the ontology. $p_{1..m}$ represents the set of Data and Object

[1] http://www.rosettanet.org/

[2] http://www.ebxml.org/

Properties with m being the total number of data and object properties. Element $<i,j>$ of the table is marked with "x" if the domain of property p_j is class o_i. We use FCA to create a hierarchy of concepts displayed as an inheritance tree. According to [10], maintainability, readability and understanding of a hierarchy are better when it is shallow. For this reason, we consider only two levels in the hierarchy of the concept lattice.

Table 1. Example of cross-table.

R	p_1	p_2	p_3	....	p_m
o_1	x	x	x		x
o_2		x	x		x
...					
o_k	x		x		x

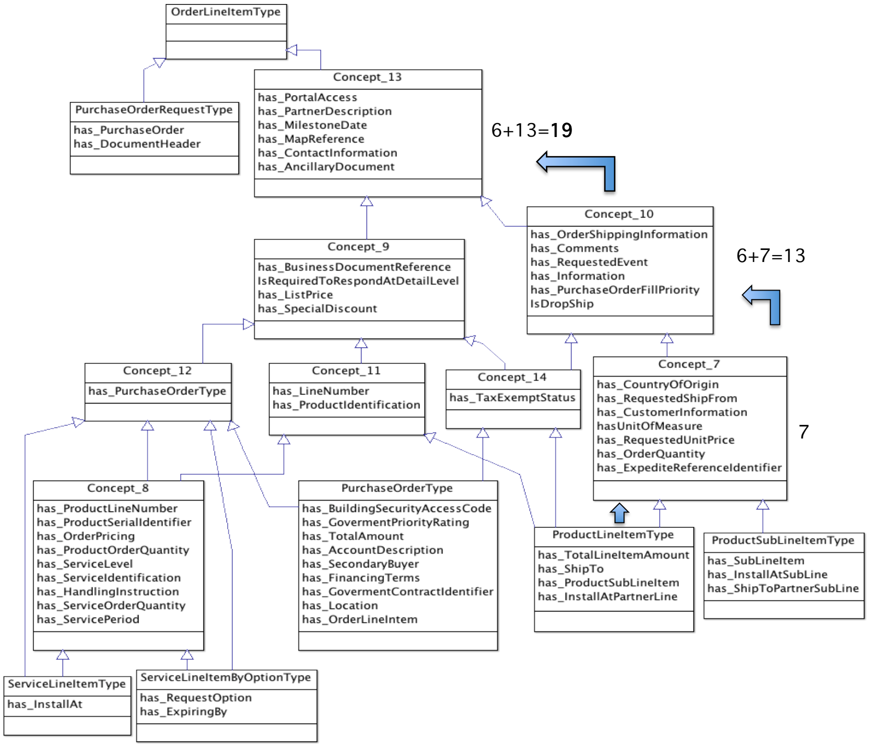


Fig. 1. An example of a `concept lattice` describing `RosettaNet PIP3A4 Purchase Order Request`. The process to build this `concept lattice` starts by extracting ontology artifacts (Class, Data Property and Object Property) from the OWL File of PIP3A4 which are added to a cross-table as CSV file. This file serves as input to the FCA system to build this concept lattice.

The concept lattice in Figure 1, which is the output of our FCA application, describes the RosettaNet PIP3A4 (Purchase Order Request). Intermediate concepts (of the form Concept_NN) were generated by the algorithm and represent the shared set of properties. The others represent the original ontology classes.

Each class will be associated with the concept having the largest number of shared properties. As shown in Figure 1, we note that the Concept_7 combines the two classes ProductSubLineItemType and ProductLineItemType with 19 shared properties; Concept_8 joins ServiceLineItemByOptionType and ServiceLineItemType because of 22 shared properties.

The combined classes share some semantics consistent with the concept definitions provided by RosettaNet. They represent one concept (Concept_7 and Concept_8) that has been generated by our approach. As the regrouped classes share tokens in their compound names, we will use these to rename the FCA generated names. So, Concept_7 becomes ProductLineItem and Concept_8, ServiceLineItem.

Using the generated group of classes, we update the original ontology. To do this, we used the OWL API[3] for manipulating, developing and maintaining the ontology.

4 Experimentation

The goal of our study is to evaluate the efficiency of our approach for concept detection in an OWL ontology. In this section, we describe our experimental setup and results.

In order to test our methodology, we use the RosettaNet Ontology, proposed in our ICEIS 2013 paper [5]. It is the result of mapping the full set of RosettaNet Partner Interface Process (PIP) descriptions, currently defined with DTD or XML Schemas format, to an ontological representation using an OWL/XML rendering. Among the 132 PIPs, 112 PIPs are available for download from the PIP Directory in RosettaNet website from which we generated 138 OWL documents, valid according to the XML Schema for OWL/XML serialization from syntactical point of view. These OWL documents were also checked for consistency with an OWL reasonner.

In Table 3, we evaluate our RosettaNet Ontology using state of the art ontology metrics defined in Table 2. Only basic metrics related to the main elements of ontologies have been used [11]. Although, we use two metrics that indicate the relationship and inheritance richness of an ontology schema. The **Relationship Richness metric** (*rr*) reflects the diversity and placement of relations in the ontology [13]. It is defined as the ratio of the number of properties (*nop*) divided by the sum of the number of subclasses (*nosc*) plus the number of properties. Also, the **Inheritance Richness metric** (*ir*) represents how knowledge is grouped across different levels of the ontology inheritance tree [13]. It is the average number of subclasses per class.

[3] http://owlapi.sourceforge.net/

Table 2. Ontology metrics.

Metrics		**Definition**
Number of	classes (*noc*)	number of classes (|C|) in the ontology.
	data properties (*nodp*)	number of data properties.
	object properties (*noop*)	number of object properties.
	properties (*nop*)	sum of *nodp* and *noop* metrics.
	subclasses (*nosc*)	number of subclasses (|sC|) in the ontology.
	root classes (*norc*)	number of root classes (without superclasses). The range of this metric is between 1 and |C|.
	leaf classes (*nolc*)	classes without subclasses. The range of this metric is between 1 and |C|.

Table 3. Empirical analysis of our RosettaNet Ontology using some Ontology Metrics: Before and After applying our FCA-based Concept Detection Approach.

Metrics	All PIPs (Before)	All PIPs (After)
noc	1252	1252
nodp	3045	3045
noop	2607	2607
nop	5652	5652
nosc	0	384
norc	1252	1050
nolc	1252	868
rr	1	0.93
ir	0	0.31

We notice in Table 3 that the values of *ir* and *nosc* metrics are zero and the values of *norc* and *nolc* are equal to the number of classes |C| in the ontology. These metric values indicate that our original RosettaNet PIP Ontology as generated from the DTD and XML Schemas is a flat or horizontal ontology representing a general knowledge despite the fact that RosettaNet represents a specific domain of interest with many elements sharing a common semantics.

The application of our FCA-based Concept Detection approach extracts 182 groups of concepts comprising 384 classes from the 1252 in the ontology, bringing the inheritance metric *ir* from `0` to `0.31`. On average, each concept contains 2 classes.

We extracted 90 groups of concepts because several concepts are shared between PIP files e.g. the concept, combining `RegionalBusinessTaxIdentifier` and `NationalBusinessTaxIdentifier` classes, is detected in 3 PIPs (`PIP3A5`, `PIP3A11` and `PIP3B6`).

We also performed a manual validation of all detected concepts. We noticed that among the 90 concepts detected, 79 concepts have effectively a common semantics. So, we have a detection precision of 87%. From Table 3, we can see that the value of *nosc* and *ir* metrics increase, *norc* and *nolc* decrease after applying our FCA-based approach.

5 Related Work

Ontologies and Formal Concept Analysis (FCA) aim at modeling concepts [1]. For this reason, we use FCA to regroup concepts in a flat ontology representing general knowledge to improve its inheritance richness. To our knowledge, no previous work has addressed the problem of flat ontology using FCA techniques.

Some FCA-based proposals in Ontology Engineering differ from our own strategy with respect to the nature of the problem. Cimiano et al. [1] proposed a benchmark to discuss how FCA can be used to support Ontology Engineering and how ontology can be exploited in FCA applications. The FCA can support the building of the ontology and the constructed ontology can be analyzed and navigated using FCA techniques [1].

Stumme [12] presents an Ontology Merging approach based on FCA, named `FCA-MERGE`, for organizing business knowledge. It takes as input one or more source ontologies and returns a merged ontology between the given source ontologies.

Obitko et al. [9] propose an approach to improve ontology design by discovering the need for new objects (or classes) and relations (properties). They argue for the necessity of a better description of concepts and relations than just ordering them in taxonomy. In our case, we consider instead regrouping concepts from existing ontology classes for improving the taxonomy representation in a flat ontology.

6 Conclusion

We have presented an approach for concepts detection in a flat ontology using a Formal Concept Analysis and applied this methodology on a RosettaNet PIP Ontology which was automatically generated ontology.

Our goal is to improve the readability, maintainability and inheritance richness of an ontology. We used FCA to detect groups of concepts sharing properties and having a common semantics. Through the use of ontology metrics, we have shown that our FCA-based concept detection methodology can improve inheritance richness.

As the results using RosettaNet are promising, we suggest as future work to test the efficiency of our concept detection approach to other ontologies dealing with other domains.

Acknowledgements

We would like to thank RosettaNet Group for allowing us to download the RosettaNet Partner Interface Processes (PIPs) from the RosettaNet website. This work has been partially funded by Tunisian Government and NSERC.

References

1. Cimiano, P., Hotho, A., Stumme, G., Tane, J.: Conceptual knowledge processing with formal concept analysis and ontologies. In: Concept Lattices, pp. 189–207. Springer (2004)
2. Dogac, A., Kabak, Y., Laleci, G.B.: Enriching ebXML registries with OWL ontologies for efficient service discovery. Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE) (2004)
3. Euzenat, J., Shvaiko, P.: Ontology Matching, vol. 18. Springer Heidelberg (2007)
4. Gómez-Pérez, A., Corcho, O., Fernandez-Lopez, M.: Ontological Engineering. Springer-Verlag, London, Berlin (2002)
5. Jridi, J.E., Lapalme, G.: Adapting RosettaNet B2B standard to Semantic Web Technologies. vol. 2, pp. 484–491. 15th International Conference on Enterprise Information Systems, Angers, France (July 2013)
6. Kotinurmi, P., Haller, A., Oren, E.: Ontologically Enhanced RosettaNet B2B Integration. Semantic Web for Business: Cases and Applications (2008)
7. Kotinurmi, P., Haller, A., Oren, E.: Global Business: Concepts, Methodologies, Tools and Application, vol. 4, chap. Ontologically enhanced RosettaNet B2B Integration, p. 27. USA (2011)
8. Lytras, M., García, R.: Semantic Web applications: a framework for industry and business exploitation–what is needed for the adoption of the semantic web from the market and industry. International Journal of Knowledge and Learning 4(1), 93–108 (2008)
9. Obitko, M., Snasel, V., Smid, J., Snasel, V.: Ontology design with formal concept analysis. Concept Lattices and their Applications, Ostrava: Czech Republic pp. 111–119 (2004)
10. Sheldon, F.T., Jerath, K., Chung, H.: Metrics for maintainability of class inheritance hierarchies. Journal of Software Maintenance and Evolution: Research and Practice 14(3), 147–160 (2002)
11. Sicilia, M., Rodríguez, D., García-Barriocanal, E., Sanchez-Alonso, S.: Empirical findings on ontology metrics. Expert Systems with Applications 39(8), 6706–6711 (2012)
12. Stumme, G.: Using ontologies and formal concept analysis for organizing business knowledge. In: In Proc. Referenzmodellierung 2001. Citeseer (2001)
13. Tartir, S., Arpinar, I.B., Moore, M., Sheth, A.P., Aleman-Meza, B.: OntoQA: Metric-based ontology quality analysis. In: IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources. vol. 9 (2005)
14. Wang, J., Song, Y.: Architectures supporting RosettaNet. In: Software Engineering Research, Management and Applications, 2006. Fourth International Conference on. pp. 31–39. IEEE (2006)

Bases via Minimal Generators[*]

Pablo Cordero, Manuel Enciso, Angel Mora, Manuel Ojeda-Aciego
Universidad de Málaga, Spain
pcordero@uma.es, enciso@lcc.uma.es, amora@ctima.uma.es, aciego@uma.es

July 24, 2013

Abstract

The concept lattice corresponding to a context may be alternatively specified by means of attribute implications. One outstanding problem in formal concept analysis and other areas is the study of the equivalences between a given set of implications and its corresponding basis (notice that there exists a wide range of approaches to basis in the literature). In this work we introduce a method to provide a Duquenne-Guigues basis corresponding to the minimal generators and their closed sets from a context.

1 Introduction

The main goal of Formal Concept Analysis (FCA) is to identify the relationships between sets of objects and sets of attributes using information from a cross table. The derivation operators establish a Galois connection between the power sets of objects and attributes which generates a complete lattice, the so-called concept lattice.

One obvious goal in this framework is to remove redundancy and obtain a minimal basis. The most widely approach comes from the notion of *Duquenne-Guigues Basis* [4] also called *stem base*. This basis is minimal with respect to the number of implications, i.e. if some implication is removed from the basis, there exist valid and non-redundant implications which are valid in the dataset and cannot be inferred from the new reduced basis using Armstrong's Axioms.

In [2], the authors presented an algorithm to obtain *all* the minimal generators and their corresponding closures. From that information it is possible to build a set of implications which mimics exactly the underlying concept lattice, by using a minimal generator as antecedent and its corresponding closure as consequent. Obviously, this set can be somehow minimized, obtaining what is called a *basis*; in this paper, we will focus on the Duquennes-Guigues basis.

Specifically , a method is introduced to calculate a Duquenne-Guigues basis from all closed sets and their minimal generator.

[*] Partially supported by grants P09-FQM-5233 of the Junta de Andalucía, and TIN09-14562-C05-01, TIN2011-28084, and TIN12-39353-C04-01 of the Science and Innovation Ministry of Spain, co-funded by the European Regional Development Fund (ERDF).

2 Background

We assume as known the basic concepts of Formal Concept Analysis (FCA). [3, 11]

2.1 Simplification logic and closures

We summarize the axiomatic system of Simplification Logic for Functional Dependencies $\mathbf{SL}_{FD}$ equivalent to the well-know Armstrong's Axioms. It avoids the use of transitivity and is guided by the idea of simplifying the set of implications by efficiently removing redundant attributes inside the implications [1]. We define $\mathbf{SL}_{FD}$ as the pair $(\mathcal{L}_{FD}, \mathcal{S}_{FD})$ where the axiomatic system $\mathcal{S}_{FD}$ has the following axiom scheme and inference rules. The third rule is named *Simplification* rule and it is the core of $\mathbf{SL}_{FD}$:

$$\texttt{[Ref]} \quad \frac{A \supseteq B}{A{\to}B}$$

$$\texttt{[Frag]} \quad \frac{A{\to}B \cup C}{A{\to}B} \qquad \texttt{[Comp]} \quad \frac{A{\to}B,\ C{\to}D}{A \cup C{\to}B \cup D} \qquad \texttt{[Simp]} \quad \frac{A{\to}B,\ C{\to}D}{A \cup (C \smallsetminus B){\to}D}$$

3 Obtaining basis from minimal generators

As stated in the introduction, the goal of this position paper can be considered one step beyond the work presented in [2], where we illustrated the use of the Simplification paradigm to guide the search of all minimal generator sets.

Our main goal is studied here: a method to get a Duquenne-Guigues basis given the set of all the minimal generators and its corresponding closed sets.

Based on the properties of minimal generators - closed sets and in $\mathbf{SL}_{FD}$, we propose an operator that characterizes when it is possible to remove redundant attributes in a set of implications. The exhaustive application of this result produces a reduced implication set and, in some cases, with an empty right-hand side. These implications are removed from the output set, returning a Duquenne-Guigues basis (see Theorem 3.4).

Thus, summarizing our proposal, from a set of *(minimal generators, closed set)* the method returns a Duquenne-Guigues basis.

Theorem 3.1 *Let $\langle A \cup B, A\rangle, \langle C \cup D, C\rangle$ be two pairs obtained using* `MinGen` *algorithm [2]* [1] *where A, C are minimal generators and $A \cup B, C \cup D$ are closed sets. In this situation, the following implications are valid: $\{A{\to}B, C{\to}D\}$*

If $A \subseteq C$, then the following equivalence holds:

$$\{A{\to}B, C{\to}D\} \equiv \{A{\to}B, (C \cup B){\to}(D \smallsetminus B)\} \tag{3.1}$$

Notice that $(C \cup B \cup D \smallsetminus B)$ is a closed set.

The above equivalence (3.1), infers the definition of an operator that reduces the set of implications if we apply it exhaustively.

[1] Closed sets - minimal generators can be calculated using others methods well known.

Definition 3.2 *Let $\langle A \cup B, A\rangle, \langle C \cup D, C\rangle$ be two pairs obtained using* `MinGen` *algorithm and let $\Gamma = \{A \rightarrow B, C \rightarrow D\}$ be the corresponding equivalent set of implications. We define the following operator:*

$$\Upsilon(A \rightarrow B, C \rightarrow D) = \{A \rightarrow B, C \cup B \rightarrow D \smallsetminus B\}$$

This operator is applied only when $A \subseteq C$. We traverse the set of implications and for any two implications we check whether $A \subseteq C$ or $C \subseteq A$, applying the operator if it is the case. We have developed in Prolog this operator.

In the following definition, we present the way in which the Υ operator will be applied to a set of implications. The way in which we check both inclusions of the left hand sides of the implications, reduces the traversing of the Γ set because we compare each implication only with all later implications in the Γ set.

Definition 3.3 *Let $\Gamma = \{A_1 \rightarrow B_1, A_2 \rightarrow B_2 \ldots A_n \rightarrow B_n\}$ be a set of implications. We define the application of the operator Υ to a set of implications as its exhaustive application as follows: $\Upsilon(\Gamma) = \{\Upsilon(A_i, A_j),\quad$ i=1... n-1, j=i+1... n$\}$*

Theorem 3.4 *Let $\Phi = (\langle C_1, mg(C_1)\rangle, \langle C_2, mg(C_2)\rangle, \ldots)$ be a set where C_i is a closed set of attributes and $mg(C_i) = \{D \colon D$ is a mingen and $D^+ = C_i\}$. And let $\Gamma = \{A_1 \rightarrow B_1, \ldots\}$ the set of implications deduced from Φ. The operator $\Upsilon(\Gamma)$ renders the Duquenne-Guigues basis equivalent to Γ.*

The proof aries from the transformation made with the Υ operator, which completes the left-hand side to be a pseudo-intent and remove implications from the original set to get minimal cardinality.

Example 3.1 *Let $\Gamma = \{b \rightarrow acef, ad \rightarrow ef, abd \rightarrow cef\}$ a set of implications equivalent to the following set of closed sets and their minimal generators $= \{\langle abcdef, \{abd\}\rangle, \langle adef, \{ad\}\rangle, \langle abcef, \{b\}\rangle, \langle \varnothing, \{\varnothing\}\rangle\}$. Applying exhaustively the Υ operator to any pair of implications of Γ we get the following set of implications, which conforms with a Duquenne-Guigues basis.*

$$\Gamma' = \{b \rightarrow acef, ad \rightarrow e\}$$

4 Conclusions and future works

In this paper we present an operator which allows the transformation of a set of implications builds over the set of all minimal generators into a Duquenne-Guigues basis. The first step is to use `MinGen` Algorithm introduced in [2] to compute all minimal generators corresponding to an arbitrary set of implications. A deep study about the soundness, completeness, and complexity of the algorithms proposed are the plan of work that we sketch in this proposal paper. In the future, our interest is to achieve a basis not only with minimal cardinality in the number of implications but also with minimal size in the attributes inside of the implications. The operator defined in this work is the first step in this direction.

References

[1] P. Cordero, M. Enciso, A. Mora, I.P, de Guzmán: SLFD logic: Elimination of data redundancy in knowledge representation, LNCS 2527: 141–150, 2002.

[2] P. Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego: Computing minimal generators from implications: a logic-guided approach, Concept Lattice and Aplications - CLA 2012: 187-198, 2012.

[3] B. Ganter, Two basic algorithms in concept analysis. Technische Hochschule, Darmstadt, 1984.

[4] J.L. Guigues and V. Duquenne, Familles minimales d'implications informatives résultant d'un tableau de données binaires. Math. Sci. Humaines, 95, 5–18, 1986.

[5] M. Hermann and B. Sertkaya, On the Complexity of Computing Generators of Closed Sets. ICFCA 2008: 158–168

[6] R. Hill, Computational Intelligence and Emerging Data Technologies. 2nd Intl Conf on Intelligent Networking and Collaborative Systems (INCOS'10), pg 449–454, 2010.

[7] A. Mora, P. Cordero, M. Enciso, I. Fortes, Closure via functional dependence simplification, IJCM, 89(4): 510–526, 2012.

[8] L. Szathmary and A. Napoli and S. O. Kuznetsov, ZART: A Multifunctional Itemset Mining Algorithm , Proc. of the 6th Intl. Conf. on Concept Lattices and Their Applications (CLA '08): 47–58, 2008.

[9] L. Szathmary and P. Valtchev and A. Napoli and R. Godin, An Efficient Hybrid Algorithm for Mining Frequent Closures and Generators, Concept Lattices and Their Applications (CLA '07): 26–37, 2007.

[10] K. Nehmé, P. Valtchev, M. H. Rouane, R. Godin, On Computing the Minimal Generator Family for Concept Lattices and Icebergs, LNCS 3403: 192–207, 2005.

[11] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival (ed.), Ordered sets, pp. 445-470, 1982.

Debugging Program Code Using Implicative Dependencies

Artem Revenko[1][2]

[1] Technische Universität Dresden
Zellescher Weg 12-14, 01069 Dresden, Germany
[2] National Research University Higher School of Economics
Pokrovskiy bd. 11, 109028 Moscow, Russia
artem_viktorovich.revenko@mailbox.tu-dresden.de

Abstract. Based on the technique for finding errors in new object intents a method of debugging source code is presented. This method is capable of finding strict implicative dependencies between lines of source code covered in successful and failed runs. The output is a logical expression. Using the new method it is possible to come closer to debugging programs on a logical level not checking executions line by line. An example of applying the new method is presented. Possibilities of further development are discussed.

Keywords: formal context analysis, implication, debugging

1 Introduction

Automatic debugging is not a new topic in science and is investigated by many computer scientist. For example, in [16] authors investigate a novel method of "relative debugging" which consists in comparing particular values of data structures. In [1] and [9] authors survey different approaches to automatic debugging. In the well known work [19] the Delta Debugger tool is presented; authors introduce an approach to isolation of failure-inducing inputs. However, when it comes to finding actual causes of the failure it is still not possible to automatically explain the failure logically. Usually near-probabilistic criteria like chi-square are used [4]. Somehow it does not correspond to the correctness of the program as a program bug is either present or not.

In this work we use recent advance in Formal Concept Analysis in an attempt to find logical dependencies between fails and successful runs of a program. For example, it could be that as long as a particular part of "if" statement is not covered during the program run (i.e. a particular conditional clause is not satisfied) the program runs successfully; this would mean that a bug lies probably in this particular part of the program.

Implications which can be derived from data tables (formal context) represent strong logical dependencies between attributes. We use this advantage of implications to introduce a way of debugging program code following the logic of a program.

Several studies were performed to discover the possibilities of using Formal Concept Analysis in software development. For example, in [17] and [10] authors use Formal Concept Analysis for building class hierarchies. In [13] FCA is used to determine dependencies on program trace. Authors reveal causal dependencies and even are able to find ”likely invariants“ of program in special cases. A very interesting work on fault localization is presented in [3]. However, to our best knowledge there are no works about applying Formal Concept Analysis to program debugging.

In our previous work [14] we have introduced two approaches to revealing errors in new object intents. In this paper we recall them; one is based on computing the implication system of the context and another one is based on computing the closures of the subsets of the new object intent. Since computing closures may be performed much faster we improve and generalize this approach and finally obtain a procedure for finding all possible errors of the considered types.

After that we present a method of debugging based on the discussed above technique of finding errors in data. We provide an example and discuss the possibilities of further development.

2 Main Definitions

In what follows we keep to standard definitions of FCA [8]. Let G and M be sets and let $I \subseteq G \times M$ be a binary relation between G and M. Triple $\mathbb{K} := (G, M, I)$ is called a *(formal) context*.

The set G is called a set of *objects*. The set M is called a set of *attributes*.

Consider mappings $\varphi\colon 2^G \to 2^M$ and $\psi\colon 2^M \to 2^G$: $\varphi(X) := \{m \in M \mid gIm \text{ for all } g \in X\}$, $\psi(A) := \{g \in G \mid gIm \text{ for all } m \in A\}$. Mappings φ and ψ define a *Galois connection* between $(2^G, \subseteq)$ and $(2^M, \subseteq)$, i.e. $\varphi(X) \subseteq A \Leftrightarrow \psi(A) \subseteq X$. Hence, for any $X_1, X_2 \subseteq G$, $A_1, A_2 \subseteq M$ one has

1. $X_1 \subseteq X_2 \Rightarrow \varphi(X_2) \subseteq \varphi(X_1)$
2. $A_1 \subseteq A_2 \Rightarrow \psi(A_2) \subseteq \psi(A_1)$
3. $X_1 \subseteq \psi\varphi(X_1)$ and $A_1 \subseteq \varphi\psi(A_1)$

Usually, instead of φ and ψ a single notation $(\cdot)'$ is used. $(\cdot)'$ is usually called a *derivation operator*. For $X \subseteq G$ the set X' is called the *intent* of X. Similarly, for $A \subseteq M$ the set A' is called the *extent* of A.

Let $Z \subseteq M$ or $Z \subseteq G$. $(Z)''$ is called the *closure* of Z in $\mathbb{K}$. Applying Properties 1 and 2 consequently one gets the *monotonicity* property: for any $Z_1, Z_2 \subseteq G$ or $Z_1, Z_2 \subseteq M$ one has $Z_1 \subseteq Z_2 \Rightarrow Z_1'' \subseteq Z_2''$.

Let $m \in M, X \subseteq G$, then $\overline{m}$ is called a *negated* attribute. $\overline{m} \in X'$ whenever no $x \in X$ satisfies xIm. Let $A \subseteq M$; $\overline{A} \subseteq X'$ iff all $m \in A$ satisfy $\overline{m} \in X'$.

An *implication* of $\mathbb{K} := (G, M, I)$ is defined as a pair (A, B), written $A \to B$, where $A, B \subseteq M$. A is called the *premise*, B is called the *conclusion* of the implication $A \to B$. The implication $A \to B$ is *respected by a set of attributes* N if $A \nsubseteq N$ or $B \subseteq N$. The implication $A \to B$ holds (is valid) in $\mathbb{K}$ if it is respected by all g', $g \in G$, i.e. every object, that has all the attributes from A, also has all the attributes from B. Implications satisfy *Armstrong rules*:

$$\frac{}{A \to A}\ ,\quad \frac{A \to B}{A \cup C \to B}\ ,\quad \frac{A \to B, B \cup C \to D}{A \cup C \to D}$$

A *support* of an implication in context $\mathbb{K}$ is the set of all objects of $\mathbb{K}$, whose intents contain the premise and the conclusion of the implication. A *unit implications* is defined as an implication with only one attribute in the conclusion, i.e. $A \to b$, where $A \subseteq M,\ b \in M$. Every implication $A \to B$ can be regarded as the set of unit implications $\{A \to b \mid b \in B\}$. One can always observe only unit implications without loss of generality.

An *implication basis* of a context $\mathbb{K}$ is defined as a set $\mathfrak{L}$ of implications of $\mathbb{K}$, from which any valid implication for $\mathbb{K}$ can be deduced by the Armstrong rules and none of the proper subsets of $\mathfrak{L}$ has this property.

A minimal implication basis is an implication basis minimal in the number of implications. A minimal implication basis was defined in [11] and is known as the *canonical implication basis*. In [6] the premises of implications from the canonical base were characterized in terms of pseudo-intents. A subset of attributes $P \subseteq M$ is called a *pseudo-intent*, if $P \neq P''$ and for every pseudo-intent Q such that $Q \subset P$, one has $Q'' \subset P$. The canonical implication basis looks as follows: $\{P \to (P'' \setminus P) \mid P$ - pseudo-intent$\}$.

We say that an object g is *reducible* in a context $\mathbb{K} := (G, M, I)$ iff $\exists X \subseteq G :$ $g' = \bigcap\limits_{j \in X} j'$.

All sets and contexts we consider in this paper are assumed to be finite.

3 Finding Errors

In this section we use the idea of *data domain dependency*. Usually objects and attributes of a context represent entities. Dependencies may hold on attributes of such entities. However, such dependencies may not be implications of a context as a result of an error in object intents. Thereby, data domain dependencies are such rules that hold on data represented by objects in a context, but may erroneously be not valid implications of a context.

Every object in a context is described by its intent. In the data domain there may exist dependencies between attributes. In this work we consider only dependencies that do not have negations of attributes in premises. As mentioned above there is no need to specially observe non-unit implications. In this work we try to find the algorithm to reveal the following two most simple and common types of dependencies ($A \subseteq M,\ b, c \in M$):

1. If there is A in an object intent, there is also b, which is represented by the implication $A \to b$
2. If there is A in an object intent, there is no b, which can be symbolically represented as $A \to \overline{b}$

If we have no errors in a context, all the dependencies of Type 1 are deducible from implication basis. However, if we have not yet added enough objects in the context, we may get false consequence. Nevertheless, it is guaranteed that none of valid dependencies is lost, and, as we add objects without errors we reduce the number of false consequences from the implication basis.

The situation is different if we add an erroneous object. It may violate a dependency valid in the data domain. In this case, until we find and correct the error, we are not able to deduce all dependencies valid in the data domain from the implication basis, no matter how many correct objects we add afterwards.

We aim to restore valid dependencies and therefore correct errors.

Below we assume that we are given a context (possibly empty) with correct data and a number of new object intents that may contain errors. This data is taken from some data domain and we may ask an expert whose answers are always correct. However, we should ask as few questions as possible.

We quickly recall two different approaches to finding errors introduced in our previous works. The first one is based on inspecting the canonical basis of a context. When adding a new object to the context one may find all implications from the canonical basis of the context such that the implications are not respected by the intent of the new object. These implications are then output as questions to an expert in form of unit implications. If at least one of these implications is accepted, the object intent is erroneous. Since the canonical basis is the most compact (in the number of implications) representation of all valid implications of a context, it is guaranteed that the minimal number of questions is asked and no valid dependencies of Type 1 are left out.

Although this approach allows one to reveal all dependencies of Type 1, there are several issues. The problem of producing the canonical basis with known algorithms is intractable. Recent theoretical results suggest that the canonical base can hardly be computed even with polynomial delay ([5], [2], [12]). One can use other bases (for example, see progress in computing proper premises [15]), but the algorithms known so far are still too costly and non-minimal bases do not guarantee that the expert is asked the minimal sufficient number of questions.

However, since we are only interested in implications corresponding to an object, it may be not necessary to compute a whole implication basis. Here is the second approach. Let $A \subseteq M$ be the intent of the new object not yet added to the context. $m \in A''$ iff $\forall g \in G : A \subseteq g' \Rightarrow m \in g'$, in other words, A'' contains the attributes common to all object intents containing A. The set of unit implications $\{A \to b \mid b \in A'' \setminus A\}$ can then be shown to the expert. If all implications are rejected, no attributes are forgotten in the new object intent. Otherwise, the object is erroneous. This approach allows one to find errors of Type 1.

However, the following case is possible. Let $A \subseteq M$ be the intent of the new object such that $\nexists g \in G : A \subseteq g'$. In this case $A'' = M$ and the implication $A \to A'' \setminus A$ has empty support. This may indicate an error of Type 2, because the object intent contains a combination of attributes impossible in the data domain, but the object may be correct as well. An expert could be asked if the combination of attributes in the object intent is consistent in the data domain. For such a question the information already input in the context is not used. More than that, this question is not sufficient to reveal an error of Type 1.

Proposition 1. *Let $\mathbb{K} = (G, M, I), A \subseteq M$. The set*

$$\mathcal{I}_A = \{B \to d \mid B \in \mathcal{MC}_A, d \in B'' \setminus A \cup \overline{A \setminus B}\},$$

where $\mathcal{MC}_A = \{B \in \mathcal{C}_A | \nexists C \in \mathcal{C}_A : B \subset C\}$ and $\mathcal{C}_A = \{A \cap g' \mid g \in G\}$, is the set of all unit implications (or their non-trivial consequences with some attributes added in the premise) of Types 1 and 2 such that implications are valid in $\mathbb{K}$, not respected by A, and have not empty support.

Proposition 1 allows one to find an algorithm for computing the set of questions to an expert revealing possible errors of Types 1 and 2. The pseudocode is pretty straightforward and is not shown here for the sake of compactness.

Since computing the closure of a subset of attributes takes $\mathrm{O}(|G| \times |M|)$ time in the worst case, and we need to compute respective closures for every object in the context, the time complexity of the whole algorithm is $\mathrm{O}(|G|^2 \times |M|)$.

We may now conclude that we are able to find possibly broken dependencies of two most common types in new objects. However, this does not always indicate broken real dependency, as we not always have enough information already input in our context. That is why we may only develop a hypothesis and ask an expert if it holds.

For more details, example, and the proof of Proposition 1, please, refer to [14].

4 Debugging

4.1 Context Preparation

Normally debugging starts with a failure report. Such a report contains the input on which the program failed. By this we mean that our program was not able to output the expected result or did not finish at all. This implicitly defines "goal" function which is capable of determining either a program run was successful or not. We could imagine a case where we do not have any successful inputs, i.e. those inputs which were processed successfully by the program. However, it does not seem reasonable. In a such a case the best option seems to rewrite the code or look for obvious mistakes. Modern techniques of software development suggest running tests even before writing code itself; unless the tests are passed code is not considered finished. Therefore, successful inputs are at least those contained in the test suites.

As discussed in the beginning of this paper the problem of finding appropriate inputs was considered by different authors. This problem is indeed of essential importance for debugging. However, we do not aim at solving it. Instead we assume that inputs are already found (using user reports, random generator, or something else), processed (it is better if inputs are minimized, however, not necessary), and are at hands. We focus on processing the program runs on given inputs.

Our method consists in the following. We construct two contexts: first with successful runs as objects, second with failed runs. In both cases attributes are the lines of the code (conveniently presented via line numbers). We put a cross if during the processing of the input the program has covered the corresponding line. So in both cases we record the information about covered lines during the processing of the inputs. After the contexts are ready we treat all the objects from the context with failed runs as new objects and try to find errors as described in the previous sections. Expected output is implication of the form $A \rightarrow B$. The interpretation is as follows: in successful runs whenever lines A are covered, lines B are covered as well. However, in the inspected failed run lines A *were* covered and lines B *were not* covered. Debugging consists now in finding the reason why lines B were not covered in the processing of the failed run.

This is not absolutely automatic debugging, however, we receive some more clues and may find a bug without checking the written code line by line. More than that, this method is logically strict, it does not deal with any kind of probability. This corresponds to the real situation: the bug *is* or *is not* there, not with any probability.

4.2 Example

Consider the following function written in Python (example taken from [18]):

Listing 1.1: remove_html_markup [18]

```
1   def remove_html_markup(s):
2       tag   = False
3       quote = False
4       out   = ""
5       for c in s:
6           if (c == '<' and
7               not quote):
8               tag = True
9           elif (c == '>' and
10                not quote):
11              tag = False
12          elif (c == '"' or
13                c == "'" and
14                tag):
15              quote = not quote
16          elif not tag:
17              out = out + c
18      return out
```

The goal of the function, as follows from its name, is to remove html markup from the input, no matter if it occurs inside or outside quotes. Therefore, we may formulate our goal as: no `<` in output. Such a formulation does not allow us to catch all the bugs (as seen from the contexts below the input `"foo"` has enabled the program to reach the line 15 which should not have happened, but it is considered as a successful run), but it suffices for our purposes.

The function works as follows. After initialisation we have four "if" cases. The first and the second one checks if we have encountered a tag symbol outside of quotes. If so, the value of "tag" is changed. The third one checks if we have encountered a quote symbol inside tag. This is important for not closing a tag if the closing symbol happens to be in one of the parameters (see inputs). If so, the value of "quote" is changed. The last "if" adds the current character to the output if we are outside the tag.

We consider the following set of inputs: `foo`, `<b>foo</b>`, `"<b>foo</b>"`, `"<b>a</b>"`, `"<b></b>"`, `"<>"`, `"foo"`, `'foo'`, `<em>foo</em>`, `""`, `<"">`, `<p>`, `<a href=">">foo</a>`

We run the function on every input and check if the output contains the symbol "<". If it does not, the run was successful. We also record the lines coverage during every run, gather this information, and construct two context:

Context with successful runs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`foo`		×	×	×	×	×			×			×	×			×	×	×
`<b>foo</b>`		×	×	×	×	×	×	×	×	×	×	×	×			×	×	×
`"foo"`		×	×	×	×	×			×			×	×		×	×	×	×
`'foo'`		×	×	×	×	×			×			×	×	×		×	×	×
`<em>foo</em>`		×	×	×	×	×	×	×	×	×	×	×	×			×	×	×
`<a href=">">foo</a>`		×	×	×	×	×	×	×	×	×	×	×	×		×	×	×	×
`""`		×	×	×	×	×			×			×			×			×
`<"">`		×	×	×	×	×	×	×	×	×	×	×			×			×
`<p>`		×	×	×	×	×	×	×	×	×	×	×	×			×		×

Context with failed runs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`"<b>foo</b>"`		×	×	×	×	×	×		×	×		×	×		×	×	×	×
`"<b>a</b>"`		×	×	×	×	×	×		×	×		×	×		×	×	×	×
`"<b></b>"`		×	×	×	×	×	×		×	×		×	×		×	×	×	×
`"<>"`		×	×	×	×	×	×		×	×		×	×		×	×	×	×

Fig. 1: Contexts with failed and successful runs

It is easy to notice that in the processing of every failed input the same lines are covered. Therefore, the only difference between different objects in the context with failed runs is the names of the objects.

Inspecting any of the failed inputs in the context with successful runs using the described above technique for finding errors yields the following implication:

$$7, 13, 15 \to 8, 11$$

What is essentially said is the following: in a *successful* run if the lines 7, 13, and 15 were covered then the lines 8 and 11 *were also* covered; in every *failed* run the lines 7, 13, and 15 were covered, but the lines 8 and 11 *were not* covered. We now expect the mentioned above lines and their impact to the program run.

If the line 7 is covered then the condition in the line 6 was met. Therefore, the line 7 is covered whenever there is the symbol “<” in the input.

If the line 13 is covered then the condition in the line 12 was not met and the conditions of the first two “if” clauses were not met. This means that for some symbol from the input we should not change the value of “tag” and the symbol is not “`"`”.

If the line 15 was covered then the condition of the third “if” clause was met and the conditions of the first two “if” clauses were not met. Therefore, some symbol in the input was either “`"`” or “`’`” and the “tag” was set to `True`.

If the line 8 was not covered then the value of “tag” was never set to `True` (because we do not have this variable set to `True` elsewhere in our program and originally it is initialized to `False`). There is no need to go further to the line 11 in our investigation, because we have already found a contradiction. The value of “tag” should have been set to `True` to reach the line 15, but it was never set to `True`. From this we can deduce that possibly the condition of the third “if” clause erroneously evaluates to `True` without checking that “tag” equals `True`.

The key to this puzzle is the following. In Python as well as in many other languages logical operation “and” has a higher priority than “or”, so condition of the third “if” `(c == ’"’ or c == "’" and tag)` is implicitly transformed in `(c == ’"’ or (c == "’" and tag))`. In other words on lines 12 and 13 brackets are forgotten. After debugging the condition should look as follows: `((c == ’"’ or c == "’") and tag)` and the program runs correctly.

4.3 Further Development

The sequence in which the lines of code were covered contains even more information about the execution of a program. This information may reveal even more dependencies in the working flow of a program. It may also happen that the only difference between successful and failed runs is in the sequence in which the lines are covered, whereas the set of covered lines remains the same. In this manner we also take into account the sequential aspect of the loop that is not taken into account in the previous example.

It is not difficult to extend the introduced method with this new feature. For this purpose we change the attributes of our context. Now an attribute contains two numbers: the first number corresponds to the preceding covered line and the second number corresponds to the succeeding covered line. The information from the original modification of the method may still be captured with attributes that have the same line number two times. However, we may be not interested in the absolute precedence relation between the lines, because this information is excessive and difficult to interpret while the size of the set of attributes increases dramatically. That is why we also introduce a new parameter containing information about the delay of interest. Below we are only interested in the precedence relation within this delay, i.e. not more than the dealy number of lines should be covered between the two specified lines in order to add them to the relation I.

The number of attributes for this case, assuming that any line may precede and succeed any other line including itself, is $|M|^d$, where d is the delay.

Unfortunately, the result obtained using this modification may be difficult to interpret even if the delay is small. It makes sense to consider this modification only if the standard modification does not yield any results.

For the `remove_html_markup` function and $d = 1$ we obtain the following results:

1. $(10, 10), (12, 15) \to (8, 8), (11, 11), (10, 11), (11, 5), (9, 10), (7, 8),$
   $\overline{(17, 17)}, \overline{(9, 12)}, \overline{(15, 5)}, \overline{(13, 13)}, \overline{(13, 16)}, \overline{(16, 16)}, \overline{(16, 17)};$
2. $(17, 17), (15, 5) \to \overline{(10, 10)}, \overline{(7, 7)};$
3. $(13, 13), (15, 15), (7, 7) \to \overline{(9, 12)}, \overline{(16, 17)}, \overline{(12, 15)}, \overline{(15, 5)},$
   $(8, 8), (11, 11), (10, 11), (11, 5), (9, 10), (7, 8), (12, 13).$

In the result we still have the same obtained dependency as we had before, namely Implication 3. However, we have two more obtained results that reveal more structural features of the bug. An interpretation of this result is left to the reader.

For example, we consider Implication 1. Actually already in the premise of this implication we can recognize a clue to finding the bug. Indeed, having a pair 12 and 15 with delay 1 means the following: after the line 12 the execution jumped to the line 15. After checking only condition in the line 12 (which happened to be true) execution jumped to the consequence. As already described, the interpreter has understood the condition as a disjunction and has only evaluated the first expression (which would be enough in case of disjunction).

5 Conclusion

Based on the procedure for finding errors in new object intents a method of debugging source code was proposed. This method finds strict dependencies between source code coverage in successful and failed runs. The output of the debugging method is a logical expression which allows one to find bugs following the implicit logic of the program. Further modification of the method is possible (described above), however, it leads to results that are difficult to interpret.

Acknowledgements The author was supported by German Academic Exchange Service (DAAD).

Author thanks Sergei Kuznetsov and other participants of the project Mathematical Models, Algorithms, and Software Tools for Intelligent Analysis of Structural and Textual Data supported by the Basic Research Program of the National Research University Higher School of Economics for discussion and useful remarks.

References

1. Hiralal Agrawal. Towards automatic debugging of computer programs. Technical report, ph.d. thesis, Purdue University, 1991.
2. Mikhail A. Babin and Sergei O. Kuznetsov. Computing premises of a minimal cover of functional dependencies is intractable. *Discrete Applied Mathematics*, 161(6):742–749, 2013.
3. Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Formal concept analysis enhances fault localization in software. In Raoul Medina and Sergei A. Obiedkov, editors, *ICFCA*, volume 4933 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2008.
4. Holger Cleve and Andreas Zeller. Locating causes of program failures. In Gruia-Catalin Roman, William G. Griswold, and Bashar Nuseibeh, editors, *ICSE*, pages 342–351. ACM, 2005.
5. Felix Distel and Barış Sertkaya. On the complexity of enumerating pseudo-intents. *Discrete Applied Mathematics*, 159(6):450–466, 2011.
6. Bernhard Ganter. Two basic algorithms in concept analysis. *Preprint-Nr. 831*, 1984.
7. Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors. *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*. Springer, 2005.
8. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations.* Springer, 1999.
9. Michael Gerndt. Towards automatic performance debugging tools. In *AADEBUG*, 2000.
10. Robert Godin and Petko Valtchev. Formal concept analysis-based class hierarchy design in object-oriented software development. In Ganter et al. [7], pages 304–323.
11. J.-L. Guigues and V. Duquenne. Familles minimales d'implications informatives r*é*sultant d'un tableau de donn*é*es binaires. *Math. Sci. Hum*, 24(95):5–18, 1986.
12. Sergei O. Kuznetsov and Sergei A. Obiedkov. Some decision and counting problems of the duquenne-guigues basis of implications. *Discrete Applied Mathematics*, 156(11):1994–2003, 2008.
13. John L. Pfaltz. Using concept lattices to uncover causal dependencies in software. In *Proc. Int. Conf. on Formal Concept Analysis, Springer LNAI 3874*, pages 233–247, 2006.
14. Artem Revenko and Sergei O. Kuznetsov. Finding errors in new object intents. In *CLA 2012*, pages 151–162, 2012.
15. Uwe Ryssel, Felix Distel, and Daniel Borchmann. Fast computation of proper premises. In Amedeo Napoli and Vilem Vychodil, editors, *International Conference on Concept Lattices and Their Applications*, pages 101–113. INRIA Nancy – Grand Est and LORIA, 2011.
16. Aaron James Searle. *Automatic relative debugging.* 2006.

17. Gregor Snelting and Frank Tip. Reengineering class hierarchies using concept analysis. *SIGSOFT Softw. Eng. Notes*, 23(6):99–110, November 1998.
18. Andreas Zeller. Software debugging course. https://www.udacity.com/course/cs259.
19. Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Trans. Software Eng.*, 28(2):183–200, 2002.

Practical Computing with Pattern Structures in FCART Environment

Aleksey Buzmakov[1,2] and Alexey Neznanov[2]

[1] LORIA (CNRS – Inria NGE – U. de Lorraine), Vandœuvre-lès-Nancy, France
[2] National Research University "Higher School of Economics", Moscow, Russia
aleksey.buzmakov@inria.fr, aneznanov@hse.ru

Abstract. A new general and efficient architecture for working with pattern structures, an extension of FCA for dealing with "complex" descriptions, is introduced and implemented in a subsystem of Formal Concept Analysis Research Toolbox (FCART). The architecture is universal in terms of possible dataset structures and formats, techniques of pattern structure manipulation.

Keywords: Formal Concept Analysis, Pattern Structures, Software

Introduction

FCART[1] is a specialized software for data analysis by means of Formal Concept Analysis (FCA) and related methods aiming at processing an arbitrary dataset [1]. FCA processes a binary context to a concept lattice, which can be very useful for "gold mining" – obtaining a new knowledge. However, datasets are unlikely kept in the binary way where an object is described as a set of binary attributes it possesses. To deal with this problem different kinds of scalings can be applied to a dataset, converting it to a binary context. In some cases it can be slow or meaningless. Pattern structures (PSs) is an extension of FCA dealing with "complex" data [2]. However, just a couple of applications of PSs are available for the community and, moreover, neither of them are able to work with an arbitrary PS. Thus, we introduce a generalized approach to PSs within FCART.

The paper is organized as follows. Section 1 defines FCA and PSs. The next section describes the overall PS processing within FCART, divided into logical submodules of the approach. Finally, the paper is concluded before program interfaces of different modules are given.

1 FCA and Pattern Structures

Formal concept analysis (FCA) [3] is a mathematical formalism having many applications in data analysis. It process a binary context (a triple (G, M, I)

[1] `http://ami.hse.ru/issa/Proj_FCART`

where G is a set of objects, M is a set of attributes and $I \subseteq G \times M$ is a relation between them) into a concept lattice. Pattern structures (PSs) is a generalization of FCA for dealing with complex structures, such as sequences or graphs [4]. As it is a generalization it is enough to introduce only PSs.

Definition 1. *A PS is a triple $(G, (D, \sqcap), \delta)$, where G is a set of objects, $(D, \sqcap)$ is a complete meet-semilattice of descriptions and $\delta : G \to D$ maps an object to the description.*

The lattice operation in the semilattice ($\sqcap$) corresponds to the similarity between two descriptions d_1 and d_2, i.e. the description which is common between d_1 and d_2. Standard FCA can be presented in terms of PSs in the following way. The set of objects G remains, while the semilattice of descriptions is $(\wp(M), \cap)$, where $\wp(M)$ is a powerset of M, and, thus, a description is a set of attributes. The similarity operation corresponds to the set intersection, i.e. the similarity is the set of common attributes. If $x = \{a, b, c\}$ and $y = \{a, c, d\}$ then $x \sqcap y = x \cap y = \{a, c\}$. The mapping $\delta : G \to \wp(M)$ is given by, $\delta(g) = \{m \in M \mid (g, m) \in I\}$.

The Galois connection for a PS $(G, (D, \sqcap), \delta)$ between the set of objects and the semilattice of descriptions is defined as follows:

$$\begin{aligned} A^{\diamond} &:= \bigsqcap_{g \in A} \delta(g), && \text{for } A \subseteq G \\ d^{\diamond} &:= \{g \in G \mid d \sqsubseteq \delta(g)\}, && \text{for } d \in D, \end{aligned}$$

where the partial order (or the subsumption order) on D is defined w.r.t. the similarity operation $\sqcap$: $c \sqsubseteq d \Leftrightarrow c \sqcap d = c$, and c is subsumed by d.

Definition 2. *A pattern concept of a PS $(G, (D, \sqcap), \delta)$ is a pair (A, d) where $A \subseteq G$ and $d \in D$ such that $A^{\diamond} = d$ and $d^{\diamond} = A$, A is called a concept extent and d is called a concept intent.*

As in the standard case of FCA, a pattern concept corresponds to the maximal set of objects A whose description subsumes the description d, while there is no $e \in D$, subsuming d, i.e. $d \sqsubseteq e$, describing every object in A. The set of all concepts can be partially ordered w.r.t. partial order on the extents (dually, the intents by $\sqsubseteq$), within a concept lattice.

Example 1. PSs are successfully used for interval data [5]. For example, in gene expression data every gene is described by its expression value in different situations. The meet-semilattice $(D_{ips}, \sqcap_{ips})$ includes vectors of intervals. An example of an interval PS is given by δ-function in Table 1. The description of g_1 is $g_1^{\diamond} = \langle [1, 3]; [3, 5]; [2, 4] \rangle$. The description materializes the fact that the gene expression in situations m_1, m_2, m_3 are within the corresponding intervals. The similarity operation ($\sqcap_{ips}$) between two interval descriptions $g_1^{\diamond}$ and $g_2^{\diamond}$ is the component-wise convex hull of intervals. Thus, $g_1^{\diamond} \sqcap g_2^{\diamond} = \langle [1, 7]; [3, 6]; [2, 5] \rangle$. The interval pattern concept lattice resulting from this PS is shown in Figure 1 (* is a special description subsuming anything).

Example 2. Given a dataset with objects described by elements of poset P, e.g. sequences (w.r.t sequence-subsequence relation) or graphs (w.r.t. subgraph isomorphism relation), a corresponding PS can be defined in the following way. The semilattice $(D, \sqcap)$ based on poset P is a subset of the powerset of P, $D \subseteq \wp(P)$, such that if $d \in D$ contains an element $p \in P$ then all its "subelements" x should be included into d, $\forall p \in d, \nexists x \leq p : x \notin d$, and the semilattice operation is the set intersection for two sets of elements. Given two patterns $d_1, d_2 \in D$, the set intersection operation ensures that if an element p belongs to $d_1 \sqcap d_2$ then any subsequence of p belongs to $d_1 \sqcap d_2$ and, thus, $(d_1 \sqcap d_2) \in D$.

However, the set of all possible "subelements" for a given pattern can be rather large. Thus, it is more efficient and representable to keep a pattern $d \in D$ as a set of all maximal elements $\tilde{d}$, $\tilde{d} = \{p \in d \mid \nexists x \in d : x \geq p\}$. Note that representing a pattern by the set of all maximal elements allows for an efficient implementation of the intersection "$\sqcap$" of two patterns.

	m_1	m_2	m_3
g_1	$[1, 3]$	$[3, 5]$	$[2, 4]$
g_2	$[5, 7]$	$[4, 6]$	$[2, 5]$
g_3	$[1, 9]$	$[2, 7]$	$[6, 6]$

Table 1: An Interval PS.

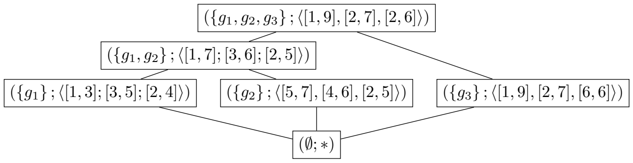

$(\{g_1, g_2, g_3\}; \langle[1, 9], [2, 7], [2, 6]\rangle)$
$(\{g_1, g_2\}; \langle[1, 7]; [3, 6]; [2, 5]\rangle)$
$(\{g_1\}; \langle[1, 3]; [3, 5]; [2, 4]\rangle)$
$(\{g_2\}; \langle[5, 7], [4, 6], [2, 5]\rangle)$
$(\{g_3\}; \langle[1, 9], [2, 7], [6, 6]\rangle)$
$(\emptyset; *)$

Fig. 1: The concept lattice for the PS in Table 1.

PSs can be hard to process due to the usually large number of concepts in the concept lattice and the complexity of the similarity operation (think for instance of the graph isomorphism problem). Moreover, a pattern lattice can contain a lot of irrelevant patterns for an expert. Projections of PSs "simplify" to some degree the computation and allow one to work with a reduced description. In fact, projections can be considered as constraints (or filters) on patterns respecting certain mathematical properties, ensuring that the concepts in the projected lattice have correspondence to the original ones [4].

A projection $\psi : D \to D$ is an operator, which is monotone ($x \sqsubseteq y \Rightarrow \psi(x) \sqsubseteq \psi(y)$), contractive ($\psi(x) \sqsubseteq x$) and idempotent ($\psi(\psi(x)) = \psi(x)$). A projection preserves the semilattice operation $\sqcap$ as follows. Under a projection ψ, a PS $(G, (D, \sqcap), \delta)$ becomes the projected PS $\psi((G, (D, \sqcap), \delta)) = (G, (D, \sqcap), \psi \circ \delta)$. The concepts of a projected pattern structure have a "similar" concept in the initial pattern structure [4].

2 Pattern Structures Techniques

As a PS is an abstract mathematical object, any software aiming at the PS realization should either prepare several different PSs, such as PSs based on intervals or graphs, or give to a user an opportunity to add arbitrary PSs to the software. Our goal is to process any PSs and in this case one should decide how an arbitrary semilattice can be introduced by a user. It is not possible in some cases to enumerate all elements of a semilattice. For example, the semilattice of

```
Function CloseByOne($Ext$, $Int$)
    Data: $(G, (D, \sqcap), \delta)$, extent $Ext$ and intent $Int$ of a concept.
    Result: All canonical ancestor concepts of the concept $(Ext, Int)$.
    foreach $S \subseteq G$, $S \succ Ext$ do
        $NewInt \longleftarrow \prod_{g \in S} \delta(g)$ ;                /* $\sqcap$ - intersection */
        $NewExt \longleftarrow \{g \in G \mid NewInt \sqsubseteq \delta(g)\}$;   /* $\sqsubseteq$ - subsumption */
        if IsCanonicExtension($Ext$, $NewExt$) then
            SaveConcept(($NewExt, NewInt$));
            CloseByOne($NewExt$,$NewInt$);
CloseByOne($\emptyset$, $\top$);                          /* Find all concepts... */
```

Algorithm 1: The modified version of CbO for PS processing.

graphs is infinite and even if one would like to select a finite subset of it, the subset should be significantly large in order to be useful in real-life applications. Another option is the constructive way for defining a semilattice, i.e. one should be able to keep any element of the given semilattice, to compute the semilattice operation between two elements of the semilattice and to check equality of two elements. Although the subsumption relation on a semilattice can be checked as $c \sqsubseteq d \Leftrightarrow c \sqcap d = c$, in many cases it can be more efficient to check the subsumption relation directly. Later we discuss how semilattices are processed more carefully.

But how can we build a concept lattice from a given PS? Many state-of-the-art algorithms can be slightly modified in such a way that avoid enumeration of attributes, i.e. performing only the set intersection operation and checking the subset relation without naming the attributes. This modification allows to further substitute the set intersection by the corresponding semilattice operations and to compute the concept lattice from a PS. Algorithm 1 shows the listing of the modified `CbO` [6] algorithm. Moreover, modified algorithms can easily process standard FCA by introducing the described above powerset semilattice. Since PSs can be processed with a number of different algorithms, we should allow to a user to introduce any algorithms he wants.

The following parts, called plugins, are introduced in FCART:

- A constructive semilattice description;
- Extent and Intent storages, managing extents and intents of a lattice;
- A concept lattice builder working with any available semilattices.

Now we can build a concept lattice from any PSs, but we still do not know how to process the different element nature of a semilattice, i.g. how to load or save it. These problems are discussed in the following subsection as well as the processing of projections of PSs.

2.1 Input and Output Data Formats

For the purposes of keeping and exchanging of patterns format JSON is chosen because it allows to serialize nearly any kind of data, is standardized [1], has low

[1] http://www.json.org/

```
{
    “Count”: 3,
    “Inds”:[2, 5, 8]
}
```

(a) Indices array.

```
{
    “Count”: 3,
    “Inds”::[2.3, 5.5, 8.1]
}
```

(b) Real numbers array.

```
{[
    { “NodesCount” : 2 },
    { “Nodes” : [
        { “Int” : 0, “Ext” : 0 },
        { “Int” : 1, “Ext” : 1 }
    ]},
    { “ArcsCount” : 1 },
    { “Arcs” : [
        { “S” : 0, “D” : 1 },
    ]},
]}
```

(c) Concept Lattice.

Fig. 2: JSON formats for object and semilattice element descriptions.

parsing overhead, and is more compact than XML. We introduce the following general datatypes: primitives (numbers, strings), sets, ordered sets, rooted trees and general structures, i.e. graphs. *But what kind of data we need to process?* First a dataset from an external source should be converted to JSON and put into an internal collection. This imported dataset corresponds to a δ-function for a PS. Since the target semilattice can be a projection, the descriptions in this semilattice can be different from the descriptions in the imported dataset. For example, an object description can be a graph, while the projection can be a chain which can be kept in more efficient and tractable structure than a general graph. Thus we have two datatypes, one is used for a δ-function and the second is for a semilattice object. To allow for a plugin work with only the descriptions this plugin can work, the plugin specifies the external and internal datatypes by unique ID of that datatype. Figure 2 exemplifies indexes array, which can be used to keep sets, and numbers array, which can be used as the initial description of interval PS.

The next entity for exchanging between FCART and a plugin is a concept lattice. In our case it is a set of concepts with several edges. The concepts extents and intents are referred by special indexes, which come from extent and intent storages. The simple lattice is exemplified in Figure 2c.

Finally, a plugin can have its own running settings, which are given in an arbitrary JSON. For example, this properties allows us to realize a class of projections rather than a given projection, i.g. the projections of a graph to all its subgraphs of no more then k vertices, where k is a parameter of the plugin.

2.2 Pattern Manager Plugin

A semilattice $(D, \sqcap)$ is given in the constructive way by a plugin called “Pattern Manager”. The main operations which should be performed by this plugin are listed in Table 2. The first two properties are description types the plugin can load from a dataset or process as patterns. Patterns here refers to an internal data format of patterns known only by this pattern manager. Loading patterns from a given JSONs, patterns can be intersected or compared. This allows to give a semilattice in the constructive way without enumerating all possible elements of a lattice. Any patterns can be saved in a JSON of a ‘GetPatternType()’ type. And, finally, there are three functions which can creates patterns. To remove a pattern and clear the memory of this pattern, function ‘FreePattern’ is introduced.

2.3 Extent and Intent Storage Plugins

Although Pattern Manager can create a lot of patterns by the intersection or the loading operations, it is not responsible for memory it creates. Plugin ‘Intent Storage’ is a special layer which separates the raw representation of a pattern (an output of a Pattern Manager) and the IDs of intents, which are used in a lattice builder. Moreover, all patterns should pass through an Intent Storage and thus it controls memory for patterns. Intent Storage is responsible for the indexes it creates and, thus, it can be (de)serialized in a unified way in order to preserve the intents between sessions. Finally, as Intent Storage translates some of its call to Pattern Manager, we should initialize Intent Storage by the required Pattern Manager. The functions of Intent Storage are the same as for Pattern Manager but it should be initialized with a Pattern Manager and can be (de)serialized.

Plugin ‘Extent Storage’ is an analog of Intent Storage but for the extents. We know exactly what an extent is, and, thus, the additional layer ‘Extent Manager’ is not necessary. To work with extents in the bottom to top strategy we usually do not require any intersection operations and just add objects to this set. The interface functions of Extent Storage plugin are shown in Table 3.

2.4 Lattice Builder Plugin

Finally, to build a pattern concept lattice, a special plugin “Lattice Builder” is introduced. Lattice Builder takes as an arguments Extent and Intent storages and the path where the result lattice in the describing above format should be saved. It has three functions: ‘Initialize()’ taking Extent and Intent Storage plugins; ‘AddObject(ObjID, IntentID)’ taking a unique ID of an object which should be added to the context with the corresponding description given by its ID; and finally ‘Build()’ building or postprocessing a lattice and writing it to the required file. We should remember that there are two types of algorithms for building a concept lattice: incremental such as AddIntent [7], where after each addition of an object the new lattice is constructed, and non-incremental such as CbO [6], where the lattice is constructed for all objects at once. The function AddObject can be used to construct a lattice in an incremental way by the first algorithms or to collect a context by the algorithms from the second group.

2.5 Organization of Plugins

To allow the efficient implementation of any plugins, they are kept in a dynamic link library with a special API, which is called “Plugin System API”. This library should contain at least three functions listed in Table 4. Function ’GetDescription’ return a JSON array [2], with description of every plugin that can be found

[2] JSON is selected for plugin data representation by the previously mentioned reasons.

in the plugin system. The description contains unique ID of a plugin, type of the plugin, i.e. Patten Manager, Extent or Intent storage, or LatticeBuilder. According to the type of the plugin it contains the map from a plugin functions to the functions realized in the library.

Every plugin in a system should implement the functions listed in Table 5. Which allows to use them in a generalized way. The plugin can describe its parameters, for example the size of graph in a projection, and then load them and save them in a described JSON format. Finally, as some plugins can be initialized by other plugins, a plugin can request another plugin in its description. The initializing plugin is given to the requester as an ID and FCART has API for requesting an address of a plugin be the plugin ID and the function name.

Conclusion

Pattern structures is a very general and powerful technique for knowledge extraction form complex object descriptions with perspectives for working with Big Data. We implemented PSs within an original framework in an efficient and universal way. Current prototype is presented in FCART software and justifies the approach. The project is improving taking into account benchmark and profiling results and requirements of researches.

Acknowledgements: this research received funding from the Basic Research Program at the National Research University Higher School of Economics (Russia) and from the BioIntelligence project (France).

References

1. Neznanov, A.A., Ilvovsky, D.A., Kuznetsov, S.O.: FCART: A New FCA-based System for Data Analysis and Knowledge Discovery. In: Proc. of workshop for FCA Tools and Applications (at ICFCA'2013). (2013)
2. Kuznetsov, S.: Fitting Pattern Structures to Knowledge Discovery in Big Data. In Cellier, P., Distel, F., Ganter, B., eds.: Formal Concept Analysis SE - 17. Volume 7880 of Lecture Notes in Computer Science. Springer (2013) 254–266
3. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. 1st edn. Springer, Secaucus, NJ, USA (1997)
4. Ganter, B., Kuznetsov, S.O.: Pattern Structures and Their Projections. In Delugach, H., Stumme, G., eds.: Conceptual Structures: Broadening the Base SE - 10. Volume 2120 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 129–142
5. Kaytoue, M., Kuznetsov, S.O., Napoli, A., Duplessis, S.: Mining gene expression data with pattern structures in formal concept analysis. Information Sciences **181**(10) (2011) 1989 – 2001
6. Kuznetsov, S.O.: A fast algorithm for computing all intersections of objects in a finite semi-lattice. Automatic documentation and Mathematical linguistics **27**(5) (1993) 11–21
7. Merwe, D.V.D., Obiedkov, S., Kourie, D.: AddIntent: A new incremental algorithm for constructing concept lattices. In Goos, G., Hartmanis, J., Leeuwen, J., Eklund, P., eds.: Concept Lattices. Volume 2961. Springer (2004) 372–385

Appendix

Function	Description
ID=GetObjectType()	Returns the JSON type of an object
ID=GetPatternType()	Returns the JSON type of the patterns it works with
Pttrn=Preprocess(JSON)	Loads JSON description of an object and converts it to the internal pattern type
$Pttrn = a \sqcap b$	Computes semilattice operation between patterns a and b.
$\{True, False\}$=$a \sqsubseteq b$	Checks if pattern a is subsumed by pattern b.
$\{True, False\}$=$(a == b)$	Checks if one pattern is equal to another pattern.

Pttrn=LoadPattern()
JSON=SavePattern(a)
	Convert the internal pattern to/from the JSON with type GetPatternType().

FreePattern(a)	Free memory allocated for pattern a

Table 2: Main functions of Pattern Manager plugin API.

Function	Description
ID=Clone(ID)	Clones the extent with ID. ID==-1 is a special empty extent
AddObject(ID, objID)	Add the object objID to the extent ID
Size=Size(ID)	Returns the number of objects in the extent ID
ObjID= LastAddedObject(ID)	Returns the last added object to the extent ID.

ID=LoadExtent(JSON)
JSON=SaveExtent(ID)
	Loads and saves the extent ID form/to JSON.

Table 3: Main functions of Extent Storage plugin API.

Function	Description
Init()	Initialization of a library
Done()	Deinitialization of a library
JSON=GetDescription()	Returns the description of all the plugins that the given plugin system support

Table 4: Functions of Plugin System API.

Function	Description
Create()	Creation of a plugin object
Destroy()	Destruction of a plugin object
DescribeParams()	Describes what kind of params the plugin can have in both human-readable and machine readable forms
Load/Save Params()	Load or save params in the form described by 'DescribeParams'

Table 5: Main functions of common plugin API.

Towards Knowledge Structuring of Sensor Data Based on FCA and Ontology

Peng Wang[1], Wenhuan Lu[1], Zhaopeng Meng[1], Jianguo Wei[2 *]
and Fran çoise Fogelman-Souli é[3]

[1]School of Computer Software, Tianjin University, Tianjin 300072, China
[2]School of Computer Science and Technology, Tianjin University, Tianjin 300072, China
[3]KXEN SAS, 25 quai Galli éni, 92150 Suresnes, France
{wangp, wenhuan, mengzp, jianguo}@tju.edu.cn, {francoise}@kxen.com

Abstract

With the rapid development of sensor technology, sensor web is increasingly found in all kinds of fields. Modeling the sensor web is a key issue if one wants to achieve efficient sensors network interconnection, sensors resources access, sensors maintenance, etc. Such a model requires various stages: first, ontologies must be defined to represent knowledge of the application domain; then, modeling techniques such as semantics and data mining can be used to predict sensor defects, access, etc. These stages exploit the observed sensor data, which may be very large. Automating the ontology extraction process is key in the face of data volumes and domains variability.

In this paper, we focus on the first stage of designing the sensor equipment ontology, by using the Formal Concept Analysis approach. This ontology will allow sharing knowledge on the sensor equipment, reasoning on its spatial, temporal, and thematic characteristics and constraining further models for better performances (this last point will not be described in this paper and is left for further research). FCA allows extracting the ontology on the basis of observed sensors data: this ontology will directly model concepts and concept hierarchy, through their properties rather than the designer. We illustrate the approach through a small dataset from a drilling platform scenario. First, we initialize an empty set of concepts and properties; we can also get some sensor observations and properties. Second, we create a concept table and add concepts and properties to that table. Third, we use FCA to visualize the lattice of concepts with their properties. Finally, based on the visualization, the designer can assess the ontology or its parts: he can modify it by adding or removing a concept or a property, assigning a property to a concept or removing a property from a concept, until the ontology matches his knowledge domain and fulfills its objective.

By using FCA to design sensor equipment ontology in the drilling platform domain, we show that more implicit concepts and their relationships can be represented for a web sensor network. We believe that this approach can lead to better ontology than previous methods of building semantic sensor web. This preliminary work will be further extended to include the other three ontologies (sensor network, sensor data and sensors), based on a larger data set from a real-world drilling platform.

[*] Corresponding Author