An efficient Java implementation of the immediate successors calculation

Clément Guérin, Karell Bertet, and Arnaud Revel

L3i, Laboratory in Computer Science, Image and Interaction,
University of La Rochelle
{clement.guerin,karell.bertet,arnaud.revel}@univ-lr.fr

Abstract. The authors present in this paper an effective Java implementation of the concept immediate successors calculation. It is based on the lattice Java library, developed by K. Bertet and the Limited Objects Access algorithm, proposed by C. Demko and K. Bertet [6] with Java-specific enhancements. This work was motivated by the need of an efficient tool delivering this service in an accessible and popular programming language for a wider research project: eBDtheque. Performances are compared and analyzed.

Keywords: concept lattice, Java implementation, immediate successors, Limited Object Access algorithm, comicbooks

1 Introduction

A Galois lattice, or concept lattice, is a graph providing a representation of all the possible associations between a set of objects, or observations, and their describing attributes. Lattices can be queried, browsed and therefore provide a smart way to represent and retrieve information through a description context. Although they have been introduced a long time ago [1], they were hardly considered helpful for information retrieval before the end of the twentieth century and the work of [9, 3, 18] due to an exponential computational time issue. Even today, using concept lattices to handle a large set of objects described by an even wider set of attributes can be challenging. Indeed, it is still not trivial to use lattice's properties in a big development project, mainly because of the lack of available software frameworks. Being in need of such a tool to browse and query a dataset on comic books, described by an ontology [10], we propose an efficient Java implementation for the calculation of the immediate successors of a concept (i.e. to get the closest refined concepts according to the description of the starting point).

This paper is organized as follows. After introducing our motivations in part 2, the third section reminds how are calculated the immediate successors in the state of the art. The next section details the implemented algorithm and how it has been done in order to be as effective as possible. Section 5 shows some experimentations related to the classical immediate successors algorithm. Finally the last section concludes this paper and brings up some perspectives on our ongoing work.

2 Context and motivation

We are developing a multi-faceted solution to automatically analyze comic books in a) an image processing point of view [19], b) a semantic point of view [10]. One key point is the possibility for a user to retrieve comic books panels, the rectangular frames in which the pictures are drawn, similar to an input query, based on the semantic description of each panel from the dataset [11]. Two panels may share the same kind of content, such as speech balloons, spoken words, objects, characters or even some localized pixel's visual features (e.g colors, textures, shapes, etc.). They can be of the same shape, at the same location in the page or in the narration, drawn by the same person or come from the same volume (Table 1 and Fig. 1 show an example of what could be such shared properties). Possibilities are only limited by the ability of our system to recognize and deduce information. All these heterogeneous pieces of description have to be expressed in a way that can be interrogated and browsed efficiently.

	panel 1	panel 2	panel 3	panel 4	panel 5
contains:balloon	1	1	1	1	0
shape:wide	0	1	0	0	1
shape:high	1	0	0	1	0
size:medium	0	0	1	0	1
contains:tree	0	0	0	1	1

Table 1. Sample context

Fig. 1. Panel 4 from Table 1. Credit [5]

One way to browse data is to guide the user by a relevance feedback mechanism as it can classically be seen in content based image retrieval (CBIR) [20] and machine learning [21]. To an input query, which can be a panel or a set of features, follows a loop of interactions between the computer and the final user that will guide him, hopefully sooner than later, to his answer. At each step, the system returns a set of result panels that share the same attributes, weighted by the estimated relevance of these attributes to the query. The user is then invited to label the panels based on what he considers to be important in his query. The relevant (resp. irrelevant) features are identified with the right (resp. wrong) labeled results, their weight is dynamically adjusted, so the query can be refined to produce a more accurate output. The interaction loop between the user and the system goes on and on until the user is satisfied.

The structure of concept lattices seems to fit particularly well to the task as each concept is made of a set of panels described by a shared set of attributes. The output can be composed of panels from several distinct concepts, chosen for the weight of their attributes. The user is then guided through the lattice structure by his choices without being aware of the underlying mechanism.

As we were working with the Java language, we started looking for a way to handle lattices that could easily be integrated to the main solution. The set of observations is meant to be quite large as it does not become very interesting to retrieve data until the volume gets critical. Because of the extreme heterogeneity of comic books' panels, the growth of the observation set (the pictures) automatically implies the growth of the attribute set (the description of those pictures). Therefore, the implementation has to be efficient when dealing with large sets of both attributes and observations.

3 State of the art

More formally, a concept lattice is defined from a binary table, also denoted a formal context, $(O, I, (\alpha, \beta))$ where O is the set of objects, I the set of attributes, $\alpha(A)$ the set of attributes shared by a subset A of objects, and $\beta(B)$ the set of objects sharing a subset B of attributes. Each node of a concept lattice is denoted a concept (A, B), i.e. a maximal objects-attributes correspondence, verifying $\alpha(A) = B$ and $\beta(B) = A$. A is called the extent of the concept, and B its intent. Two formal concepts (A_1, B_1) and (A_2, B_2) are in relation in the lattice when they verify the following extension-generalization property:

$$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow \left\| \begin{array}{l} A_1 \subseteq A_2 \\ \text{(equivalent to } B_1 \supseteq B_2) \end{array} \right.$$

The whole set of formal concepts fitted out by the order relation $\leq$ is called a concept lattice or a Galois lattice because it verifies the lattice properties, and the cover relation $\prec$ corresponds to the Hasse diagram of the lattice. The concepts $\perp = (O, \alpha(O))$ and $\top = (\beta(I), I)$ respectively correspond to the bottom and the top of the concept lattice. See the book of Ganter and Wille [8] for a more complete description of formal concept lattices.

Numerous generation algorithms have been proposed in the literature [16, 7, 2, 17]. All of these proposed algorithms have a polynomial complexity with respect to the number of concepts (at best quadratic in [17]). The complexity is therefore determined by the size of the lattice (i.e. the number of concepts in the lattice), this size being exactly bounded by $2^{|O+I|}$ in the worst case (when the table context is a diagonal matrix of zeros) and by $|O+I|$ in the best case (diagonal matrix of ones). A formal and experimental comparative study of the different algorithms has been published in [13]. Although all these algorithms generate the same lattice, they propose different strategies. Ganter's NextClosure [7] is the reference algorithm that determines the concepts in lectical order (next, the concepts may be ordered by $\leq$ or $\prec$ to form the concept lattice) while Bordat's algorithm [2] is the first algorithm that computes directly the Hasse diagram of the lattice, by computing immediate successors for each concept, starting from the bottom concept, until all concepts are generated. Immediate successor calculation is appropriate for an on-demand generation inside the structure, useful for a navigation without generating the whole set of concepts.

Bordat's algorithm, independently rediscovered by [14], is issued from a corollary of Bordat's theorem [2] stating that (A', B') is an immediate successor of a concept (A, B) if and only if A' is inclusion-maximal in the following set system $\mathcal{F}_A$ defined on the objects set O by:

$$\mathcal{F}_A = \{\beta(x) \cap A \ : \ x \in I \backslash B\} \tag{1}$$

Immediate successors algorithm first generates the set system $\mathcal{F}_A$ in a linear time ; then inclusion-maximal subsets of $\mathcal{F}_A$ can easily be computed in $\mathcal{O}(|O|^3)$, using an inclusion graph for example. Notice that the inclusion-maximal subsets problem is known to be resolved in $\mathcal{O}(|O|^{2,5})$ using sophisticated data structures ([15, 12]).

It is possible to consider the restriction of a concept lattice to the attributes set. Indeed, a nice result establishes that any concept lattice is isomorphic to the closure lattice defined on the set I of attributes, where each concept is restricted to its attributes part. The closure lattice is composed of all closures - i.e. fixed points - for the closure operator $\varphi = \alpha \circ \beta$. Using the closure lattice instead of the whole concept lattice gives rise to a storage improvement, for example in case of large amount of objects. See the survey of Caspard and Monjardet [4] for more details about closure lattices.

Closure lattices can be generated by an extension of Bordat's algorithm (see Alg. 1) issued from a reformulation of Bordat's Theorem. Indeed, each immediate successor B' of a closure B is obtained by $B' = \alpha(A')$ with $A' \in \mathcal{F}_A$. Since $A' = \beta(x) \cap A = \beta(x) \cap \beta(B) = \beta(x+B)$, thus $B' = \alpha(A') = \alpha(\beta(x+B)) = \varphi(x+B)$. Therefore, immediate successors of a closure B are inclusion-minimal subsets in a set system $\mathcal{F}_B$ defined on the attributes set I by:

$$\mathcal{F}_B = \{\varphi(x + B) \ : \ x \in I \backslash B\} \tag{2}$$

The Limited Object Access algorithm (see Algorithm 2), introduced by Demko and Bertet in 2011, is another extension of Bordat's immediate successors gen-

Name: `Immediates_Successors`
Data: A context $K = (O, I, (\alpha, \beta))$; A closure B of the closure lattice of K
Result: The immediate successors of B in the closure lattice
begin
  Init the set system $\mathcal{F}_B$ with $\emptyset$;
  foreach $x \in I \backslash B$ **do**
    Add $\varphi(x + B)$ to $\mathcal{F}_B$
  end
  $Succ$=minimal inclusion subsets of $\mathcal{F}_B$;
  return $Succ$
end

Algorithm 1: Immediate successors algorithm

eration. Please refer to [6] for a complete description. The key idea behind this algorithm is to improve its efficiency by counting the objects, inspired from relational databases. Instead of considering the subset of observations as the extent of a subset of attributes, the computation of the inclusion-maximal subsets on $\mathcal{F}_A$ is made considering only its cardinality, and an incremental strategy. Four cases have to be considered to test, for each attribute x of $I \backslash B$ and each already inserted potential successor $X \subseteq I \backslash B$, the inclusion between $\varphi(B + X)$ and $\varphi(B + x)$:

1. Merge x with X when $\varphi(B + x) = \varphi(B + X)$.
2. Eliminate x as potential successor of B when $\varphi(B + x) \supset \varphi(B + X)$
3. Eliminate X as potential successor of B when $\varphi(B + x) \subset \varphi(B + X)$
4. Insert x as potential successor of B when x is neither eliminated or merged with X.

The inclusion test between $\varphi(B + X)$ and $\varphi(B + x)$ can easily be performed using the count function c and the following proposition from [6].

Proposition 1 ([6]):

$$\varphi(B + X) \subseteq \varphi(B + x) \Longleftrightarrow c(B + X + x) = c(B + X) \tag{3}$$

The count function c associates to any subset X of attributes the cardinality of the subset $\beta(X)$:

$$c(X) = |\beta(X)| \tag{4}$$

$$\varphi(B) = B + \{x \in I \backslash B \ : \ c(B) = c(B + x)\} \tag{5}$$

The count function corresponds to the notion of support introduced in association rule-mining, and is in particular used by Titanic algorithm [22].

It has been proven to be effective on a large amount of objects with a complexity of $\mathcal{O}((|I| - |B|)^2 * \mathcal{O}(c(B + X)))$. This has to be compared to the complexity of the classical immediate successors algorithm in $\mathcal{O}(|I|^2 * |O|)$.

```
Name: Immediates_Successors_LOA
Data: A context K = (O; I, (α, β)) ; A closed set B of the closed set lattice
      (ℂ_I, ⊆) of K
Result: The immediate successors of B in the lattice
begin
    Init the set system Succ_B with ∅;
    foreach x ∈ I \ B do
        add = true;
        foreach X ∈ Succ_B do
            \\ Case 1: Merge x and X in the same potential successor
            if c(B + x) = c(B + X) then
                if c(B + X + x) = c(B + x) then
                    replace X by X + x in Succ_B;
                    add=false; break;
                end
            end
            \\ Case 2: Eliminate x as potential successor
            if c(B + x) < c(B + X) then
                if c(B + X + x) = c(B + x) then
                    add=false; break;
                end
            end
            \\ Case 3: Eliminate X as potential successor
            if c(B + x) > c(B + X) then
                if c(B + X + x) = c(B + X) then
                    delete X from Succ_B
                end
            end
        end
        \\ Case 4: Insert x as a new potential successor ;
        if add then add {x} to Succ_B;
    end
    return Succ_B;
end
```

Algorithm 2: LOA immediate successors algorithm

The ability of the algorithm to handle huge sets of data is very interesting. The impact of the number of observations on the performances can be limited, depending on the implementation of c, using for example multiple keys and robust algorithms used in databases that do not need to load all data for computing a cardinality [6].

4 Implementation

4.1 Data structures

We choose to implement the Limited Object Access algorithm in Java. The performance of the Limited Object Access presented in [6] comes from a PHP/MySQL implementation, backed up by the efficient SQL indexation system. Its behavior without the help of SQL remains to be seen.

While profiling the execution of a naive implementation, conducted without paying attention to optimization, we noticed that up to 86% of the computation time was used for the calculation of the candidate sets of attributes' extent. The extent of a set of attributes is the intersection of the extents of each of its elements. It appears that the most time consuming step of the extent calculation (up to 87% of the running time) is the intersection of two sets of elements. While it only takes around 50 microseconds to perform, the calls pile grows fast with the size of the dataset, resulting in a delivery time of full seconds, even minutes. A particular attention must be paid to the optimization of the extent calculation, both in terms of calls amount and processing time.

The number of calls is limited to three for a foreach loop, as $c(B+x)$, $c(B+X)$ and $c(B+X+x)$ are consistent and can be computed once and for all at the beginning of the second loop. If they were not, the *count* function would have been called up to 8 times (2 times per if, plus 2 times for the last if).

Classical Java containers, like *HashSet* or *TreeSet*, are well suited for the task of storing the observations and attributes but are a bit too sophisticated for the representation of a simple context's binary table. In fact, observations do not necessarily have to be directly manipulated to compute the extent, a fortiori its cardinality, of a set of attributes. Assuming that the observations are sorted in a final order, each of them can be represented by its index in this order. So the extent of an attribute becomes a binary word whose length is the cardinality of the observations set. In this word, a 1 (resp. a 0) means this index's observation is (resp. is not) part of the extent. Java, as many programming languages, has a *BitSet* class to manipulate and execute operations over such data type.

The extent (resp. intent) of each attribute (resp. observation) of the context is computed and stored as a binary word once and for all at the beginning of the execution. Then, the extent of a set of attributes can be computed using a *logical AND* on the successive extents of all of its elements (see Table 2). The immediate benefit comes from the rapidity of the *AND* operation, performed in less than one microsecond (with a complexity of $\mathcal{O}(\mathrm{E}(n/w))$, n being the length of the bitset and w the size of the memory words used to store the basic *AND* operations). It is more than 50 times as fast as an intersection between two TreeSets for the same number of calls. Furthermore, this sticks to the primal boolean representation of a context (see Table 1) and the lattice (nor any part of it) does not have to be generated at any time.

Each attribute and each object is mapped to its bit index in the binary word for output readability purpose.

	p1	p2	p3	p4	p5
contains:balloon	1	1	1	1	0
shape:high	1	0	0	1	0
contains:tree	0	0	0	1	1
Extent	0	0	0	1	0

Table 2. Extent of the set of attributes $\{b, h, t\}$

5 Experiment

The dataset is made of 848 comic books panels, the observations, and two kinds of attributes. The first category, made of 100 attributes, is shared by the whole set of observations with a proportion going from 2 to 11 attributes for an average of 7. 28 attributes are assigned to a single observation. The second category includes the first one and adds 3433 attributes, leading to an average distribution of 15 attributes per observation. 3403 out of the 3533 attributes belong to less than 3 observations. Only 15 are shared by more than 100 objects.

As this system is meant to be used in a browsing context through relevance feedback, it has to be efficient going both ways from a concept (towards its successors and its predecessors). We ran our algorithm both on the calculation of the immediate successors of the bottom concept $\perp$ and the immediate predecessors of the top concept $\top$. We computed the latter as the immediate successors of $\perp$ on the inverted context (which is rigorously the same thing as calculating the immediate predecessors of $\top$ in the regular context – see Fig. 2 and 3). We choose $\perp$ as starting point because, according to our dataset, it is the concept that is supposed to have the most immediate successors.

Table 3 shows the processing times in seconds of the classical immediate successors algorithm and the Limited Object Access algorithm, both tuned with TreeSet and BitSet data structures for different scenarios corresponding to different complexity values. Processes have been run on a 8GB DDR3 machine, powered by a 2.7GHz quad core Xeon. The results show a significant improvement of the computation time which can be attributed, for one part, to LOA and, for the other part, to the use of binary words.

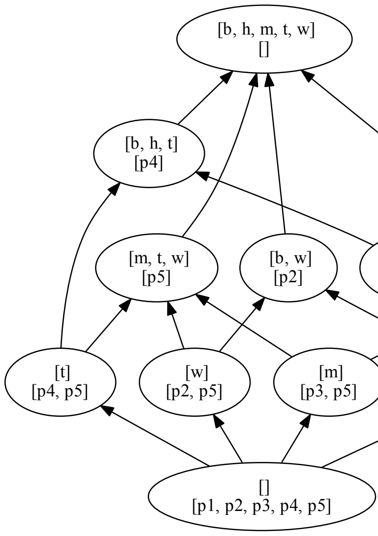

Fig. 2. Concept lattice generated from the context presented in Table 1. Intent: Attributes, Extent: Observations

	Immediate successors		Immediate predecessors	
	$\|O\| = 848$ $\|I\| = 100$	$\|O\| = 848$ $\|I\| = 3533$	$\|O\| = 100$ $\|I\| = 848$	$\|O\| = 3533$ $\|I\| = 848$
Classical + TreeSet	3.06	11767.52	549.76	994.00
Classical + BitSet	0.77	196.58	62.39	9.77
LOA + TreeSet	0.29	11.26	5.65	1183.75
LOA + BitSet	0.02	0.15	0.24	1.20

Table 3. Computation time (in seconds) of the immediate successors of $\perp$ and immediate predecessors of $\top$.

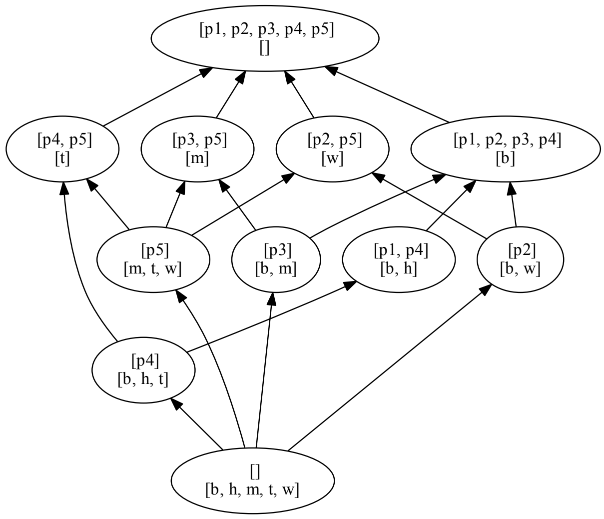

Fig. 3. Concept lattice generated from the inversion of the context presented in Table 1
Intent: Observations, Extent: Attributes

The bitset improvement over the LOA implementation shows a reduction of the execution time going from 14 to over 900 times. The calculation of the immediate predecessors of $\top$ over the set of 3533 attributes is the worst possible case as it results in 848 different concepts of one observation, each of them with a set of around 20 attributes. The running time is almost entirely taken by the million of extent calculations, which is why the gain is more spectacular here.

A test on 500 randomly picked concepts has been run over the third dataset ($|O| = 100, |I| = 848$) resulting in a mean processing time of 0.18 second.

Computation time shrinkage is minimized on the classical method as the optimization only applies on the closure operation, which is a fraction of the global computation time.

We deal here with computation times kept below the second on a reasonable machine. This starts to be interesting in terms of human-machine interaction capabilities where at least a dozen of concept's immediate successors have to be calculated at each step.

6 Conclusion and ongoing work

We presented an efficient Java implementation of the immediate successors calculation. The short processing times on quite large datasets are promising and make the query by navigation through a lattice possible. The source code will be made available soon, along a full Java library to handle lattices.

7 Acknowledgment

This work was supported by the European Regional Development Fund, the region Poitou-Charentes (France), the General Council of Charente Maritime (France) and the town of La Rochelle (France).

References

1. G. Birkhoff. *Lattice theory.* American Mathematical Society, 3d edition, 1967.
2. J.P. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Math. Sci. Hum.*, 96:31–47, 1986.
3. Claudio Carpineto and Giovanni Romano. Ulysses: A lattice-based multiple interaction strategy retrieval interface. In Brad Blumenthal, Juri Gornostaev, and Claus Unger, editors, *Human-Computer Interaction*, volume 1015 of *Lecture Notes in Computer Science*, pages 91–104. Springer Berlin Heidelberg, 1995.
4. N. Caspard and B. Monjardet. The lattice of closure systems, closure operators and implicational systems on a finite set: a survey. *Discrete Applied Mathematics*, 127(2):241–269, 2003.
5. Cyb. *Bubblegôm Gôm vol. 1, pp. 3*, volume 1. Studio Cyborga, Goven, France, 2009.
6. C. Demko and K. Bertet. Generation algorithm of a concept lattice with limited object access. In *Proc. of Concept lattices and Applications (CLA'11)*, pages 113–116, Nancy, France, October 2011.
7. B. Ganter. Two basic algorithms in concept analysis. *Technische Hochschule Darmstadt (Preprint 831)*, 1984.
8. B. Ganter and R. Wille. *Formal Concept Analysis, Mathematical foundations.* Springer Verlag, Berlin, 1999.
9. R. Godin, C. Pichet, and J. Gecsei. Design of a browsing interface for information retrieval. *SIGIR Forum*, 23(SI):32–39, May 1989.
10. Clément Guérin. Ontologies and spatial relations applied to comic books reading. In *PhD Symposium of Knowledge Engineering and Knowledge Management (EKAW)*, Galway, Ireland, October 2012.
11. Clément Guérin, Karell Bertet, and Arnaud Revel. An approach to Semantic Content Based Image Retrieval using Logical Concept Analysis. Application to comicbooks. In *International Workshop "What can FCA do for Artificial Intelligence?" FCA4AI*, pages 53–56, France, August 2012.
12. Munro I. Efficient determination of the transitive closure of a directed graph. *Information Processing Letter*, pages 56–58, 1971.
13. S. Kuznetsov and S. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theorical Artificial Intelligence*, 14(2-3):189–216, 2002.

14. Christian Lindig. Fast concept analysis. *Working with Conceptual Structures-Contributions to ICCS*, pages 235–248, 2002.
15. Fisher M.J. and Meyer A.R. Boolean matrix multiplication and transitive closure. In 12^{th} *Annual Sympsosium on Switching and Automata Theory*, pages 129–131, 1971.
16. E. Norris. An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 23(2), 1978.
17. L. Nourine and O. Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71:199–204, 1999.
18. Uta Priss. Lattice-based information retrieval. *Knowledge Organization*, 27:132–142, 2000.
19. Christophe Rigaud, Norbert Tsopze, Jean-Christophe Burie, and Jean-Marc Ogier. Robust frame and text extraction from comic books. In Young-Bin Kwon and Jean-Marc Ogier, editors, *Graphics Recognition. New Trends and Challenges*, volume 7423 of *Lecture Notes in Computer Science*, pages 129–138. Springer Berlin Heidelberg, 2013.
20. Yong Rui, Thomas S Huang, Michael Ortega, and Sharad Mehrotra. Relevance feedback: a power tool for interactive content-based image retrieval. *Circuits and Systems for Video Technology, IEEE Transactions on*, 8(5):644–655, 1998.
21. Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.
22. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with TITANIC. *Data and Knowledge Engineering*, 42(2):189–222, August 2002.