Constructing Iceberg Lattices from Frequent Closures Using Generators

Laszlo Szathmary, Petko Valtchev, Amedeo Napoli, Robert Godin

Constructing Iceberg Lattices from Frequent Closures Using Generators

Laszlo Szathmary[1], Petko Valtchev[1], Amedeo Napoli[2], and Robert Godin[1]

[1] Dépt. d'Informatique UQAM, C.P. 8888,
Succ. Centre-Ville, Montréal H3C 3P8, Canada
Szathmary.L@gmail.com, valtchev.petko@uqam.ca, godin.robert@uqam.ca
[2] LORIA UMR 7503, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
napoli@loria.fr

Abstract. Frequent closures (FCIs) and generators (FGs) as well as the precedence relation on FCIs are key components in the definition of a variety of association rule bases. Although their joint computation has been studied in concept analysis, no scalable algorithm exists for the task at present. We propose here to reverse a method from the latter field using a fundamental property of hypergraph theory. The goal is to extract the precedence relation from a more common mining output, i.e. closures and generators. The resulting order computation algorithm proves to be highly efficient, benefiting from peculiarities of generator families in typical mining datasets. Due to its genericity, the new algorithm fits an arbitrary FCI/FG-miner.

1 Introduction

The discovery of frequent patterns and meaningful association rules is a key data mining task [1] whereby a major challenge is the handling of the huge number of potentially useful patterns and rules. As a possible remedy, various subfamilies have been designed that losslessly represent the entire family of valid associations (see [2] for a survey). Some of the most popular bases involve subfamilies of frequent itemsets (FIs), e.g. closures (FCIs) or generators (FGs), which themselves losslessly represent the entire FI family. Part of these bases further require the precedence relation among closures as well (e.g. the *informative basis*).

The aforementioned three structural components, i.e. FCIs, FGs, and precedence, have been targeted in various configurations and from diverging viewpoints. A range of data mining methods compute generators and closures (e.g. *Titanic* [3] and *A-Close* [4]), and at least one targets closures and their order (e.g. *Charm-L* [5]). In concept analysis, in turn, the focus has been on computing both closures and order in the concept lattice [6], whereas a few methods also output the generators (e.g. [7,8]). Dedicated methods for precedence computation exist as well, yet their reliance on transaction-wise operations hurts their scalability (e.g. [9]).

Here we tackle the efficient computation of all three components. More precisely, we concentrate on the task of computing precedence links from the families

of FCIs and FGs, i.e. from the output of some miners from the literature (see above). In our analysis of the problem, we reverse an argument of Pfaltz for computing generators from closures and precedence [7], using a fundamental result from hypergraph theory. The restated problem amounts to the computation of the minimal transversal graph of a given hypergraph, a popular problem that nevertheless withholds some secrets [10]. Based on an adaptation of a classical algorithm for the task, we provide our algorithm called *Snow* for efficient iceberg lattice construction. Our preliminary experiments show that the strategy is very suitable as the order computation cost is only a fraction of the one for discovering all FCIs and FGs.

The contribution of the paper is therefore threefold. First, we put forward an important interplay between precedence and generators with respect to closures. Second, we show that in practice it can be efficiently exploited to yield either the generators given the FCIs and the precedence relation, or the precedence relation given the FCIs and the generators. Third, the proposed concrete algorithm, *Snow*, provides the capabilities of iceberg lattice construction to any FCI/FG-miner.

The paper is organized as follows. Section 2 provides the basic concepts of frequent itemset mining, concept analysis, and hypergraph theory. In Section 3, we introduce the *Snow* algorithm. Finally, conclusions and future work are discussed in Section 4.

2 Background on Frequent Itemsets, Iceberg Lattices, and Hypergraphs

Here we recall the basic notions of frequent itemset mining, formal concept analysis, and hypergraph theory on which our approach is based.

2.1 Frequent Itemsets and Their Distinguished Subfamilies

Consider the following 5×5 sample dataset: $\mathcal{D} = \{(1,\ ACDE),\ (2,\ ABCDE),\ (3,\ AB),\ (4,\ D),\ (5,\ B)\}$. Throughout the paper, we will refer to this example as **"dataset $\mathcal{D}$"**.

We consider a set of *objects* or *transactions* $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$, a set of *attributes* or *items* $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, and a relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$. A set of items is called an *itemset*. Each transaction has a unique identifier (*tid*), and a set of transactions is called a *tidset*.[3] For an itemset X, we denote its corresponding tidset, often called its *image*, as $t(X)$. For instance, in dataset $\mathcal{D}$, the image of AB is 23, i.e. $t(AB) = 23$. Conversely, $i(Y)$ is the itemset corresponding to a tidset Y. The *length* of an itemset is its cardinality, whereas an itemset of length k is called a k-itemset. The *support* of an itemset X, denoted by $supp(X)$, is the size of its image, i.e. $supp(X) = |t(X)|$. An itemset X is called *frequent*, if its

[3] For convenience, we write an itemset $\{A, B, E\}$ as ABE, and a tidset {2,3} as 23.

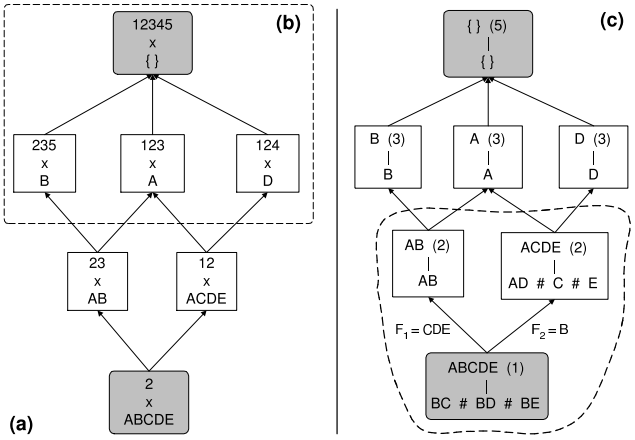

Fig. 1. Concept lattices of dataset $\mathcal{D}$. **(a)** The entire concept lattice. **(b)** An iceberg part of (a) by $min\_supp = 3$ (indicated by a dashed rectangle). **(c)** Although generators are not a formal part of the lattice, they are drawn within the respective nodes

support is not less than a given *minimum support* (denoted by min_supp), i.e. $supp(X) \geq min\_supp$. The image function induces an equivalence relation on $\wp(\mathcal{A})$: $X \cong Z$ iff $t(X) = t(Z)$ [11]. Moreover, an equivalence class has a unique maximum w.r.t. set inclusion and possibly several minima, called *closed* itemset (a.k.a. concept intents in concept analysis [12]) and *generator* itemsets (a.k.a. key-sets in database theory or free-sets), respectively. The support-oriented definitions exploiting the monotony of support upon $\subseteq$ in $\wp(\mathcal{A})$ are as follows:

Definition 1 (closed itemset; generator). *An itemset X is* closed[4] *(*generator*) if it has no proper superset (subset) with the same support (respectively).*

The *closure* operator assigns to an itemset X the maximum of its equivalence class. For instance, in dataset $\mathcal{D}$, the sets AB and AD are generators, and their closures are AB and $ACDE$, respectively (i.e. the equivalence class of AB is a singleton).

The families of frequent closed itemsets (FCIs) and frequent generators (FGs) are well-known reduced representations [13] for the set of all frequent itemsets (FIs). Furthermore, they underlie some non-redundant bases of valid association rules such as the *generic basis* [2]. Yet for other bases, the inclusion order between FCIs is essential, and, in some cases, the precedence order between those. The precedence relation $\prec$, henceforth referred to as merely *precedence*, is defined the following way: $X \prec Z$ iff *(i)* $X \subset Z$, and *(ii)* there exists no Y such that $X \subset Y \subset Z$. Here, X is called the (immediate) *predecessor* of Z.

[4] In the rest of the paper, closed itemsets are abbreviated as **"CIs"**.

The FCI family of a dataset together with $\prec$ compose the *iceberg lattice*, which is a complete meet-semi-lattice (bottomless if $\emptyset$ is the universal itemset). The iceberg corresponds to the frequent part of the CI lattice, also known in concept analysis [12] as the intent lattice of a context. It is dually isomorphic to the concept lattice of the same context (in data mining terms, the lattice of all pairs (tidset, itemset) where both are closed and mutually corresponding). We shall use here the latter structure as visualization basis hence the effective order on our drawings is $\supseteq$ rather than $\subseteq$. In Figure 1, (a) and (b) depict the concept lattice of dataset $\mathcal{D}$ and its iceberg part, respectively.

As we tackle here the computation of an FG-decorated iceberg, i.e. an iceberg lattice where generators are explicitly associated to their closure (see also Figure 1 (c)), existing approaches for related tasks are of interest. In the data mining field, FCIs together with associated FGs have been targeted by a growing set of levelwise FCI-miners such as *Titanic* [3], *A-Close* [14], etc. In contrast, the only case of mining FCIs with precedence that we are aware of is *Charm-L* [5]. The research in concept analysis algorithms has put the emphasis on the set of concepts, or CIs, and less so on precedence (see [6] for a good coverage of the topic). Few methods also compute the generators [7,8], whereas focused procedures retrieve precedence from the concept set, as in [9]. However, all but few concept analysis methods perform at least part of the computation transaction-wise, which makes them impractical for large datasets.

Yet a close examination of the most relevant approaches, i.e. those computing generator-decorated lattices such as in [7], reveals an interesting property that we shall exploit in our own method. In fact, as the latter paper indicates, from the set of all closures and their precedence, one may easily compute the generators for each closure.

The key notion here is the *blocker* of a family of sets (equivalent to a hypergraph transversal as we show below). Thus, given a ground set X and a family of subsets $\mathcal{X} \subseteq \wp(X)$, a blocker of $\mathcal{X}$ is a set $Z \subseteq X$ which intersects every member thereof to a non-empty result ($\forall T \in \mathcal{X}, Z \cap T \neq \emptyset$). A *minimal blocker* is the one which admits no other blocker as a proper subset. Blockers are brought into the closure lattice using the associated *faces*, i.e. the differences between two adjacent closures within the lattice. Formally, given two CIs X_1 and X_2 of a dataset such that $X_1 \prec X_2$, their associated face is $F = X_2 \setminus X_1$.

Example. Consider the closure lattice in Figure 1 (c). In a node, it depicts the corresponding CI, its support and the list of its generators. Let us consider the bottom concept with the closure $ABCDE$. It has two predecessors, thus its faces are: $F_1 = ABCDE \setminus AB = CDE$ and $F_2 = ABCDE \setminus ACDE = B$.

A basic property of the generators of a CI X states that they are the minimal blockers of the family of faces associated to X [7]:

Theorem 1. *Assume a CI X and let $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$ be its family of associated faces. Then a set $Z \subseteq X$ is a minimal generator of X iff X is a minimal blocker of $\mathcal{F}$.*

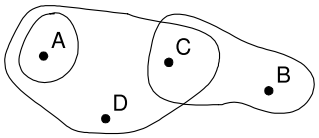

Fig. 2. A hypergraph $\mathcal{H}$, where $V = \{A, B, C, D\}$ and $\mathcal{E} = \{A, BC, ACD\}$

EXAMPLE. The minimal blockers of the family $\{CDE, B\}$ are $\{BC, BD, BE\}$, which is exactly the set of minimal generators of $ABCDE$ (see Figure 1 (c)).

From Theorem 1, Pfaltz devised a method for computing the generators of a closed set from its predecessors in the lattice [15]. Although the precedence links are assumed as input there, their computation, as indicated above, requires expensive manipulations of tidsets. Thus, even though the algorithm itself employs scalable operations, its input is impossible to provide at low cost.

However, this apparent deadlock can be resolved by transposing the problem setting into the more general framework of hypergraphs and transversals.

2.2 Hypergraphs and Their Transversal Graphs

A hypergraph [16] is a generalization of a graph, where edges can connect arbitrary number of vertices.

Definition 2 (hypergraph). *A* hypergraph *is a pair $(V,\mathcal{E})$ of a finite set $V = \{v_1, v_2, \ldots, v_n\}$ and a family $\mathcal{E}$ of subsets of V. The elements of V are called vertices, the elements of $\mathcal{E}$ edges. A hypergraph is* simple *if none of its edges is contained in any other of its edges, i.e. $\forall \mathcal{E}_i, \mathcal{E}_j \in \mathcal{E} : \mathcal{E}_i \subseteq \mathcal{E}_j \Rightarrow i = j$.*

EXAMPLE. The hypergraph $\mathcal{H}$ in Figure 2 is not simple because the edge A is contained in the edge ACD.

A transversal of a hypergraph $\mathcal{H}$ is a subset of its vertices intersecting each edge of $\mathcal{H}$. A *minimal* transversal does not contain any other transversal as proper subset.

Definition 3 (transversal). *Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. A set $T \subseteq V$ is called a* transversal *of $\mathcal{H}$ if it meets all edges of $\mathcal{H}$, i.e. $\forall E \in \mathcal{E} : T \cap E \neq \emptyset$. A transversal T is called* minimal *if no proper subset T' of T is a transversal.*

Clearly, the notion of (*minimal*) *blocker* in the work of Pfaltz [7] is equivalent to the notion of (minimal) transversal.

EXAMPLE. The hypergraph $\mathcal{H}$ in Figure 2 has two minimal transversals: AB and AC. The sets ABC and ACD are transversals but they are not minimal.

Definition 4 (transversal hypergraph). *The family of all minimal transversals of $\mathcal{H}$ constitutes a hypergraph on V called the* transversal hypergraph *of $\mathcal{H}$, which is denoted by $Tr(\mathcal{H})$.*

EXAMPLE. Considering the hypergraph $\mathcal{H}$ in Figure 2, $Tr(\mathcal{H}) = \{AB, AC\}$.

An obvious property of $Tr(\mathcal{H})$ is that it is necessarily simple. Next, a duality exists between a simple hypergraph and its transversal hypergraph [16]:

Proposition 1. *Let $\mathcal{G}$ and $\mathcal{H}$ be two simple hypergraphs. Then $\mathcal{G} = Tr(\mathcal{H})$ if and only if $\mathcal{H} = Tr(\mathcal{G})$.*

Consequently, computing twice the transversal hypergraph of a given simple hypergraph yields the initial hypergraph.

Corollary 1 (duality). *Let $\mathcal{H}$ be a simple hypergraph. Then $Tr(Tr(\mathcal{H})) = \mathcal{H}$.*

EXAMPLE. Consider the following *simple* hypergraph: $\mathcal{G} = \{A, BC\}$. Then, $\mathcal{G}' = Tr(\mathcal{G}) = Tr(\{A, BC\}) = \{AB, AC\}$, and $Tr(\mathcal{G}') = Tr(\{AB, AC\}) = \{A, BC\}$.

From a computational point of view, the extraction of $Tr(\mathcal{G})$ from $\mathcal{G}$ is a tough problem as the former can be exponentially larger than the latter. In fact, the exact complexity class of the problem is still not known [10]. Yet many algorithms for the task exist and perform well in practice since the worst case rarely occurs. For instance, an incremental algorithm due to Berge [16] computes the $Tr(\mathcal{G})$ as the final member of a sequence of hypergraphs, each representing the transversal hypergraph of a subgraph of $\mathcal{G}$. It is noteworthy that the algorithm in [7] follows a similar pattern in computing minimal blockers of a set family.

3 The Snow Algorithm

Snow computes precedence links on FCIs from associated generators by exploiting the duality with faces.

3.1 Underlying Structural Results

As indicated in the previous section, a minimal blocker of a family of sets is an identical notion to a minimal transversal of a hypergraph. This trivially follows from the fact that each hypergraph $(V,\mathcal{E})$ is nothing else than a family of sets drawn from $\wp(V)$. Now following Theorem 1, we conclude that given a CI X, the associated generators compose the transversal hypergraph of its family of faces $\mathcal{F}$ seen as the hypergraph $(X,\mathcal{F})$.

Next, further to the basic property of a transversal hypergraph, we conclude that $(X,\mathcal{F})$ is necessarily simple. In order to apply Proposition 1, we must also show that the family of generators associated to a CI, say $\mathcal{G}$, forms a simple hypergraph. Yet this holds trivially due to the definition of generators. We can therefore advance that both families represent two mutually corresponding hypergraphs.

Table 1. Input of *Snow* on dataset $\mathcal{D}$ by $min\_supp = 1$

FCI (supp)	FGs
AB (2)	AB
$ABCDE$ (1)	BE; BD; BC
A (3)	A

FCI (supp)	FGs
B (3)	B
$ACDE$ (2)	E; C; AD
D (3)	D

Property 1. Let X be a closure and let $\mathcal{G}$ and $\mathcal{F}$ be the family of its generators and the family of its faces, respectively. Then, for the underlying hypergraphs it holds that $Tr(X, \mathcal{G}) = (X, \mathcal{F})$ and $Tr(X, \mathcal{F}) = (X, \mathcal{G})$.

EXAMPLE. Let us again consider the bottom concept in Figure 1 (c) with $ABCDE$ as its CI. It has three generators: BC, BD, and BE. The transversal hypergraph of the generator family is $Tr(\{BC, BD, BE\}) = \{CDE, B\}$. That is, it corresponds exactly to the family of faces as computed above.

3.2 The Algorithm

The *Snow* algorithm exploits Property 1 by computing faces from generators. Thus, its input is made of FCIs and their associated FGs. Several algorithms can be used to produce this input, e.g. *A-Close* [4], *Titanic* [3], *Zart* [17], *Eclat-Z* [18], etc. Table 1 depicts a sample input of *Snow*.

On such data, *Snow* first computes the faces of a CI as the minimal transversals of its generator hypergraph. Next, each difference of the CI X with a face yields a predecessor of X in the closure lattice.

Example 1. Consider again $ABCDE$ with its generator family $\{BC, BD, BE\}$. First, we compute its transversal hypergraph: $Tr(\{BC, BD, BE\}) = \{CDE, B\}$. The two faces $F_1 = CDE$ and $F_2 = B$ indicate that there are two predecessors for $ABCDE$, say Z_1 and Z_2, where $Z_1 = ABCDE \setminus CDE = AB$, and $Z_2 = ABCDE \setminus B = ACDE$. Application of this procedure for all CIs yields the entire precedence relation for the CI lattice. □

The pseudo code of *Snow* is given in Algorithm 1. As input, *Snow* receives a set of CIs and their associated generators. The `identifyOrCreateTopCI` procedure looks for a CI whose support is 100%. If it does not find one, then it creates it by taking an empty set as the CI with 100% support and a void family of generators (see Figure 1 (c) for an example). The `getMinTransversals` function computes the transversal hypergraph of a given hypergraph. More precisely, given the family of generators of a CI X, the function returns the family of faces of X. It is noteworthy that any algorithm for transversal computation in a hypergraph would be appropriate here. In our current implementation, we use an optimized version of Berge's algorithm henceforth referred to as *BergeOpt* that we do not present here due to space limitations. The `getPredecessorCIs`

Algorithm 1 (Snow):

Description: build iceberg lattice from FCIs and FGs
Input: a set of CIs and their associated generators

```
1) identifyOrCreateTopCI(setOfFCIsAndFGs);
2) // find the predecessor(s) for each concept:
3) for all c in setOfFCIsAndFGs {
4)     setOfFaces ← getMinTransversals(c.generators);
5)     predecessorCIs ← getPredecessorCIs(c.closure, setOfFaces);
6)     loop over the CIs in predecessorCIs (p) {
7)         connect(c, p);
8)     }
9) }
```

function calculates the differences between a CI X and the family of faces of X. The function returns the set of all CIs that are predecessors of X. The `connect` procedure links the current CI to its predecessors.

For a running example, see Example 1.

3.3 Experimental Results

The *Snow* algorithm was implemented in Java in the Coron data mining platform [19].[5] The experiments were carried out on a bi-processor Intel Quad Core Xeon 2.33 GHz machine with 4 GB RAM running under Ubuntu GNU/Linux. All times reported are real, wall clock times.

For the experiments, we used several real and synthetic dataset benchmarks. Database characteristics are shown in Table 2 (top). The chess and connect datasets are derived from their respective game steps. The Mushrooms database describes mushrooms characteristics. These three datasets can be found in the UC Irvine Machine Learning Database Repository. The pumsb, C20D10K, and C73D10K datasets contain census data from the PUMS sample file. The synthetic datasets T20I6D100K and T25I10D10K, using the IBM Almaden generator, are constructed according to the properties of market basket data. Typically, real datasets are very dense, while synthetic data are usually sparse.

Table 2 (bottom left and right) provides a summary of the experimental results. The first column specifies the various minimum support values for each of the datasets (low for the sparse dataset, higher for dense ones). The second and third columns comprise the number of FCIs and the execution time of *Snow* (given in seconds). The CPU time does not include the cost of computing FCIs and FGs since they are assumed as given.

As can be seen, *Snow* is able to discover the order very efficiently in both sparse and dense datasets. To explain the reason for that, recall that the only

[5] http://coron.loria.fr

Table 2. Top: database characteristics. **Bottom:** response times of *Snow*

database name	# records	# non-empty attributes	# attributes (in average)	largest attribute
T20I6D100K	100,000	893	20	1,000
T25I10D10K	10,000	929	25	1,000
chess	3,196	75	37	75
connect	67,557	129	43	129
pumsb	49,046	2,113	74	7,116
MUSHROOMS	8,416	119	23	128
C20D10K	10,000	192	20	385
C73D10K	10,000	1,592	73	2,177

min_supp	# concepts (including top)	*Snow* (finding order)
T20I6D100K		
0.75%	4,711	0.11
0.50%	26,209	0.36
0.25%	149,218	3.24
T25I10D10K		
0.40%	83,063	1.07
0.30%	122,582	2.73
0.20%	184,301	4.48
chess		
65%	49,241	0.85
60%	98,393	1.77
55%	192,864	3.95
connect		
65%	49,707	0.54
60%	68,350	0.78
55%	94,917	1.82

min_supp	# concepts (including top)	*Snow* (finding order)
pumsb		
80%	33,296	1.95
78%	53,418	4.10
76%	82,539	7.08
MUSHROOMS		
20%	1,169	0.05
10%	4,850	0.17
5%	12,789	0.47
C20D10K		
0.50%	132,952	3.04
0.40%	151,394	4.37
0.30%	177,195	4.29
C73D10K		
65%	47,491	1.51
60%	108,428	3.97
55%	222,253	10.13

computationally intensive step in *Snow* is the transversal hypergraph construction. Thus, the total cost heavily depends on the efficiency of that step. Furthermore, to find out why the underlying algorithm *BergeOpt* performs so well, we investigated the size of its input data. Figure 3 shows the distribution of hypergraph sizes in the datasets T20I6D100K, MUSHROOMS, chess, and C20D10K.[6] Note that we obtained similar hypergraph-size distributions in the other four datasets too. Figure 3 indicates that most hypergraphs only have 1 edge, which is a trivial case, whereas large hypergraphs are relatively rare. As a consequence, *BergeOpt* and thus *Snow* perform very efficiently.

We interpret the above results as an indication that the good performance of *Snow* is independent of the density of the dataset. In other terms, provided that the input hypergraphs do not contain too many edges, i.e. there are only few

[6] For instance, the dataset T20I6D100K by $min\_supp = 0.25\%$ contains 149,019 1-edged hypergraphs, 171 2-edged hypergraphs, 25 3-edged hypergraphs, 0 4-edged hypergraphs, 1 5-edged hypergraph, and 1 6-edged hypergraph.

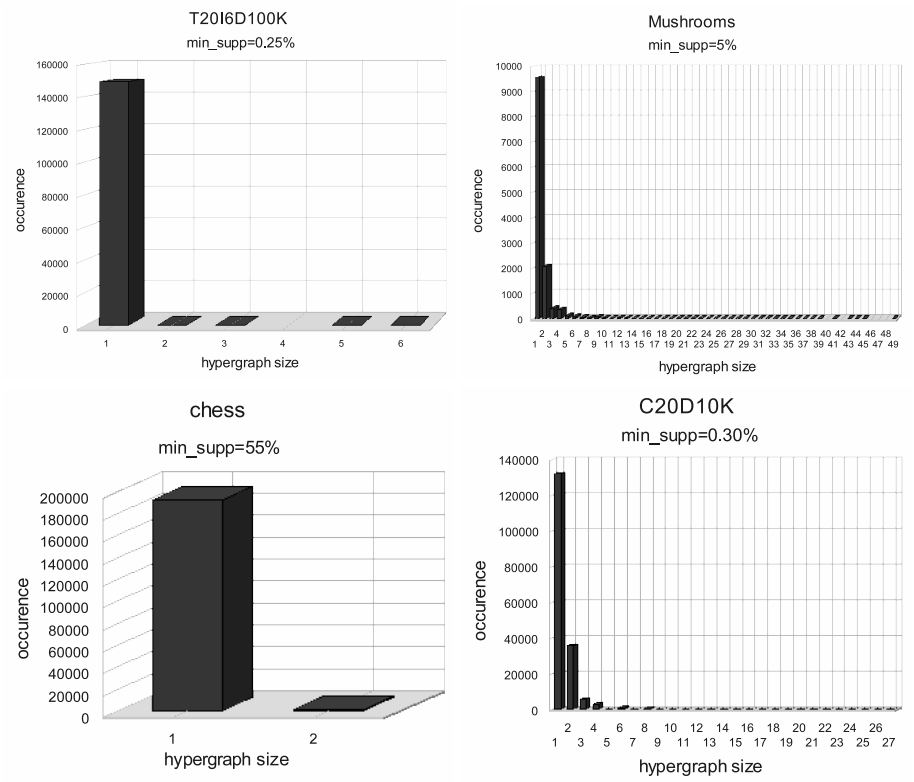

Fig. 3. Distribution of hypergraph sizes

FGs per FCIs, the computation is very fast. A natural question arises with this observation: does the modest number of FGs in each class hold for all realistic datasets in the literature? If not, could one profile those datasets which meet this condition?

4 Conclusion

The computation of precedence of FCIs is a challenging task due to the potentially huge number of these. Indeed, as most of the existing algorithms rely on dimensions that may grow rapidly, they remain impractical for large datasets.

We presented here an approach for elegantly solving the problem starting from a rather common mining output, i.e. FCIs and their FGs, which is typically provided by levelwise FCI-miners. To that end, we reverse a computation principle initially introduced by Pfaltz in closure systems by translating it beforehand within the minimal transversal framework of hypergraph theory. Although the cost of the transversal hypergraph problem is potentially non-polynomial,

the contributed algorithm, *Snow*, proved to be highly efficient in practice largely due to the low number of FGs associated to an FCI.

Based on this observation, we claim that *Snow* can enhance in a generic yet efficient manner any FCI/FG-miner, thus transforming the latter into an iceberg lattice constructor. Beside the possibility to compute valuable association rule bases from its output, the resulting method could also compete on the full-fledged concept lattice construction field.

On the methodological side, our study underlines the duality between generators and order w.r.t. closures: either can be used in combination with FCIs to yield the other one. It rises the natural question of whether FCIs alone could be used to efficiently retrieve both precedence and FGs. Conversely, it would be interesting to examine whether FCIs can in turn be easily computed from order and generators, yet this is more of a discrete mathematics challenge rather than data mining concern.

References

1. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: Proc. of the 20th Intl. Conf. on Very Large Data Bases (VLDB '94), San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1994) 487–499
2. Kryszkiewicz, M.: Concise Representations of Association Rules. In: Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery. (2002) 92–109
3. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing Iceberg Concept Lattices with TITANIC. Data and Knowledge Engineering **42**(2) (2002) 189–222
4. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In: Proc. of the 7th Intl. Conf. on Database Theory (ICDT '99), Jerusalem, Israel (1999) 398–416
5. Zaki, M.J., Hsiao, C.J.: Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. IEEE Transactions on Knowledge and Data Engineering **17**(4) (2005) 462–478
6. Carpineto, C., Romano, G.: Concept Data Analysis: Theory and Applications. John Wiley & Sons, Ltd (2004)
7. Pfaltz, J.L.: Incremental Transformation of Lattices: A Key to Effective Knowledge Discovery. In: Proc. of the First Intl. Conf. on Graph Transformation (ICGT '02), Barcelona, Spain (Oct 2002) 351–362
8. Nehme, K., Valtchev, P., Rouane, M.H., Godin, R.: On Computing the Minimal Generator Family for Concept Lattices and Icebergs. In Ganter, B., Godin, R., eds.: Proc. of the 3rd Intl. Conf. on FCA. LNCS 3403, Lens (France), Springer (2005) 192–207
9. Nourine, L., Raynaud, O.: A fast algorithm for building lattices. Inf. Process. Lett. **71**(5–6) (1999) 199–204
10. Eiter, T., Gottlob, G.: Identifying the Minimal Transversals of a Hypergraph and Related Problems. SIAM Journal on Computing **24**(6) (1995) 1278–1304
11. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining frequent patterns with counting inference. SIGKDD Explor. Newsl. **2**(2) (2000) 66–75
12. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer, Berlin/Heidelberg (1999)

13. Calders, T., Rigotti, C., Boulicaut, J.F.: A Survey on Condensed Representations for Frequent Sets. In Boulicaut, J.F., Raedt, L.D., Mannila, H., eds.: Constraint-Based Mining and Inductive Databases. Volume 3848 of Lecture Notes in Computer Science., Springer (2004) 64–80
14. Pasquier, N.: Mining association rules using formal concept analysis. In: Proc. of the 8th Intl. Conf. on Conceptual Structures (ICCS '00), Shaker-Verlag (Aug 2000) 259–264
15. Pfaltz, J.L., Taylor, C.M.: Scientific Knowledge Discovery through Iterative Transformation of Concept Lattices. In: Proc. of the Workshop on Discrete Applied Mathematics in conjunction with the 2nd SIAM Intl. Conf. on Data Mining, Arlington, VA, USA (2002) 65–74
16. Berge, C.: Hypergraphs: Combinatorics of Finite Sets. North Holland, Amsterdam (1989)
17. Szathmary, L., Napoli, A., Kuznetsov, S.O.: ZART: A Multifunctional Itemset Mining Algorithm. In: Proc. of the 5th Intl. Conf. on Concept Lattices and Their Applications (CLA '07), Montpellier, France (Oct 2007) 26–37
18. Szathmary, L., Valtchev, P., Napoli, A., Godin, R.: An Efficient Hybrid Algorithm for Mining Frequent Closures and Generators. In: Proc. of the 6th Intl. Conf. on Concept Lattices and Their Applications (CLA '08), Olomouc, Czech Republic (2008) (submitted).
19. Szathmary, L.: Symbolic Data Mining Methods with the Coron Platform. PhD Thesis in Computer Science, Univ. Henri Poincaré – Nancy 1, France (Nov 2006)