Fast Mining of Iceberg Lattices: A Modular Approach Using Generators

Laszlo Szathmary[1], Petko Valtchev[1], Amedeo Napoli[2], Robert Godin[1], Alix Boc[1], and Vladimir Makarenkov[1]

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

Abstract. Beside its central place in FCA, the task of constructing the concept lattice, i.e., concepts plus Hasse diagram, has attracted some interest within the data mining (DM) field, primarily to support the mining of association rule bases. Yet most FCA algorithms do not pass the scalability test fundamental in DM. We are interested in the iceberg part of the lattice, alias the frequent closed itemsets (FCIs) plus precedence, augmented with the respective generators (FGs) as these provide the starting point for nearly all known bases. Here, we investigate a modular approach that follows a workflow of individual tasks that diverges from what is currently practiced. A straightforward instantiation thereof, *Snow-Touch*, is presented that combines past contributions of ours, *Touch* for FCIs/FGs and *Snow* for precedence. A performance comparison of *Snow-Touch* to its closest competitor, *Charm-L*, indicates that in the specific case of dense data, the modularity overhead is offset by the speed gain of the new task order. To demonstrate our method's usefulness, we report first results of a genome data analysis application.

1 Introduction

Association discovery [1] in data mining (DM) is aimed at pinpointing the most frequent patterns of *items*, or *itemsets*, and the strongest *associations* between items dug in a large *transaction* database. The main challenge here is the potentially huge size of the output. A typical way out is to focus on a *basis*, i.e. a reduced yet lossless representation of the target family (see a list in [2]). Many popular bases are either formulated in terms of FCA or involve structures that do. For instance, the *minimal non-redundant association rules* [3] require the computation of the *frequent closed itemsets* (FCI) and their respective *frequent generators* (FGs), while the *informative basis* involves the inclusion-induced precedence links between FCIs.

We investigate the computation of iceberg lattices, i.e., FCIs plus precedence, together with the FGs. In the DM literature, several methods exist that

target FCIs by first discovering the associated FGs (e.g. the levelwise FCI miners *A-Close* [4] and *Titanic* [5]). More recently, a number of extensions of the popular FCI miner *Charm* [6] have been published that output two or all three of the above components. The basic one, *Charm-L* [7], produces FCIs with precedence links (and could be regarded as a lattice construction procedure). Further extensions to *Charm-L* produce the FGs as well (see [8,9]).

In the FCA field, the frequency aspect of concepts has been mostly ignored whereas generators have rarely been explicitly targeted. Historically, the first method whose output combines closures, generators and precedence has been presented in [10] yet this fact is covered by a different terminology and a somewhat incomplete result (see explanations below). The earliest method to explicitly target all three components is to be found in [11] while an improvement was published in [12]. Yet all these FCA-centered methods have a common drawback: They scale poorly on large datasets due to repeated scans of the entire database (either for closure computations or as an incremental restructuring technique). In contrast, *Charm-L* exploits a *vertical* encoding of the database that helps mitigate the cost of the impact of the large object (a.k.a. transaction) set.

Despite a diverging *modus operandi*, both FCA and data mining methods follow the same overall algorithmic schema: they first compute the set of concepts/FCIs and the precedence links between them and then use these as input in generator/FG calculation.

However efficient *Charm-L* is, its design is far from optimal: For instance, FCI precedence is computed at FCI discovery, benefiting from no particular insight. Thus, many FCIs from distant parts of the search space are compared. We therefore felt that there is space for improvement, e.g., by bringing in techniques operating locally. An appealing track seemed to lay in the exploration of an important duality from hypergraph theory to inverse the computation dependencies between individual tasks (and thus define a new overall workflow). To clarify this point, we chose to investigate a less intertwined algorithmic schema, i.e. by a modular design so that each individual task could be targeted by the best among a pool of alternative methods.

Here, we describe a first step in our study, *Snow-Touch*, which has been assembled from existing methods by wiring them w.r.t. our new schema. Indeed, our method relies on *Charm* for mining FCIs and on the vertical FG miner *Talky-G*, which are put together into a combined miner, *Touch* [13], by means of an FGs-to-FCIs matching mechanism. The *Snow* method [14] extracts the precedence links from FCIs and FGs.

The pleasant surprise with *Snow-Touch* was that, when a Java implementation thereof was experimentally compared to *Charm-L* (authors' version in C++) on a wide range of data, our method prevailed on all dense datasets. This was not readily anticipated as the modular design brought a computational overhead, e.g. the extra matching step. Moreover, *Snow-Touch* proved to work well with real-world data, as the first steps of a large-scale analysis of genomic data indicate.

In summary, we contribute here a novel computation schema for iceberg lattices with generators (hence a **new lattice construction approach**). Moreover, we derive an efficient FCI/FG/precedence miner (especially on dense sets). We also demonstrate the practical usefulness of *Snow-Touch* as well as of the global approach for association mining based on generic rules.

The remainder of the paper is as follows: Background on pattern mining, hypergraphs, and vertical pattern mining is provided in Section 2. In Section 3 we present the different modules of the *Snow-Touch* algorithm. Experimental evaluations are provided in Section 4 and conclusions are drawn in Section 5.

2 Background

In the following, we summarize knowledge about relevant structures from frequent pattern mining and hypergraph theory (with parallels drawn with similar notions from FCA) as well as about efficient methods for mining them.

2.1 Basic facts from pattern mining and concept analysis

In pattern mining, the input is a database (comparable to an FCA formal context). Records in the database are called *transactions* (alias objects), denoted here $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$. A transaction is basically subsets of a given total set of *items* (alias *attributes*), denoted here $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$. Except for its *itemset*, a transaction is explicitly identified through a unique identifier, a *tid* (a set of identifiers is thus called a *tidset*). Throughout this paper, we shall use the following database as a running example (the **"dataset $\mathcal{D}$"**): $\mathcal{D} = \{(1,\ ACDE)$, $(2,\ ABCDE)$, $(3,\ AB)$, $(4,\ D)$, $(5,\ B)\}$.

The standard $'$ derivation operators from FCA are denoted differently in this context. Thus, given an itemset X, the tidset of all transactions comprising X in their itemsets is the *image* of X, denoted $t(X)$ (e.g. $t(AB) = 23$). We recall that an itemset of length k is called a k-itemset. Moreover, the (absolute) *support* of an itemset X, $supp : \wp(\mathcal{A}) \to \mathbb{N}$, is $supp(X) = |t(X)|$. An itemset X is called *frequent*, if its support is not less than a user-provided *minimum support* (denoted by *min_supp*). Recall as well that, in $[X]$, the equivalence class of X induced by $t()$, the extremal elements w.r.t. set-theoretic inclusion are, respectively, the unique maximum X'' (a.k.a. *closed* itemset or the *concept intent*), and its set of minimums, a.k.a. the *generator* itemsets. In data mining, an alternative definition is traditionally used stating that an itemset X is *closed* (*a generator*) if it has no proper superset (subset) with the same support. For instance, in our dataset $\mathcal{D}$, B and C are generators, whose respective closures are B and $ACDE$.

In [6], a subsumption relation is defined as well: X *subsumes* Z, iff $X \supset Z$ and $supp(X) = supp(Z)$. Obviously, if Z *subsumes* X, then Z cannot be a generator. In other words, if X is a generator, then all its subsets Y are generators as well[3]. Formally speaking, the generator family forms a downset within the Boolean lattice of all itemsets $\langle \wp(\mathcal{A}), \subseteq \rangle$.

[3] Please notice that the dual property holds for non generators.

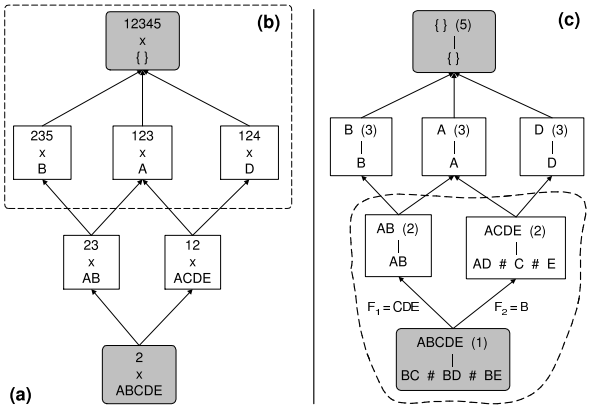


Fig. 1. Concept lattices of dataset $\mathcal{D}$. **(a)** The entire concept lattice. **(b)** An iceberg part of (a) with $min\_supp = 3$ (indicated by a dashed rectangle). **(c)** The concept lattice with generators drawn within their respective nodes

The FCI and FG families losslessly represent the family of all frequent itemsets (FIs) [15]. They jointly compose various non-redundant bases of valid association rules, e.g. the *generic basis* [2]. Further bases require the inclusion order $\leq$ between FCIs or its transitive reduction $\prec$, i.e. the precedence relation.

In Fig. 1 (adapted from [14]), views (a) and (b) depict the concept lattice of dataset $\mathcal{D}$ and its iceberg part, respectively. Here we investigate the efficient computation of the three components of an association rule basis, or what could be spelled as the generator-decorated iceberg (see Fig. 1 (c)).

2.2 Effective mining methods for FCIs, FGs, and precedence links

Historically, the first algorithm computing all closures with their generators and precedence links can be found in [10] (although under a different name in a somewhat incomplete manner). Yet the individual tasks have been addressed separately or in various combinations by a large variety of methods.

First, the construction of all concepts is a classical FCA task and a large number of algorithms exist for it using a wide range of computing strategies. Yet they scale poorly as FCI miners due to their reliance on object-wise computations (e.g. the incremental acquisition of objects as in [10]). These involve to a large number of what is called *data scans* in data mining that are known to seriously deteriorate the performances. In fact, the overwhelming majority of FCA algorithms would suffer on the same drawback as they have been designed under the assumption that the number of objects and the number of attributes remain in the same order of magnitude. Yet in data mining, there is usually a much larger number of transactions than there are items.

As to generators, they have attracted significantly less attraction in FCA as a standalone structure. Precedence links, in turn, are sometimes computed by

concept mining FCA algorithms beside the concept set. Here again, objects are heavily involved in the computation hence the poor scaling capacity of these methods. The only notable exception to this rule is the method described in [16] which was designed to deliberately avoid referring to objects by relying exclusively on concept intents.

When all three structures are considered together, after [10], efficient methods for the combined task have been proposed, among others, in [11,12].

In data mining, mining FCIs is also a popular task [17]. Many FCI miners exist and a good proportion thereof would output FGs as a byproduct. For instance, levelwise miners such as *Titanic* [5] and *A-Close* [17], use FGs as entry points into the equivalence classes of their respective FCIs. In this field, the FGs, under the name of free-sets [15], have been targeted by dedicated miners. Precedence links do not seem to play a major role in pattern mining since few miners would consider them. In fact, to the best of our knowledge, the only mainstream FCI miner that would also output the Hasse diagram of the iceberg lattice is *Charm-L* [8]. In order to avoid multiple data scans, *Charm-L* relies on a specific encoding of the transaction database, called *vertical*, that takes advantage of the aforementioned asymmetry between the number of transactions and the number of items. Moreover, two ulterior extensions thereof [7,9] would also cover the FGs for each FCI, making them the primary competitors for our own approach.

Despite the clear discrepancies in their modus operandi, both FCA-centered algorithms and FCI/FG miners share their overall algorithmic schema. Indeed, they first compute the set of concepts/FCIs and the precedence links between them and then use these as input in generator/FG calculation. The latter task can be either performed along a levelwise traversal of the equivalence class of a given closure, as in [8] and [10], or shaped as the computation of the minimal transversals of a dedicated hypergraph[4], as in [11,12] and [9].

While such a schema could appear more intuitive from an FCA point of view (first comes the lattice, then the generators which are seen as an "extra"), it is less natural and eventually less profitable for data mining. Indeed, while a good number of association rule bases would require the precedence links in order to be constructed, FGs are used in a much larger set of such bases and may even constitute a target structure of their own (see above). Hence, a more versatile mining method would only output the precedence relation (and compute it) as an option, which is not possible with the current design schema. More precisely, the less rigid order between the steps of the combined task would be: (1) FCIs, (2) FGs, and (3) precedence. This basically means that precedence needs to be computed at the end, independently from FG and FCI computations (but may rely on these structures as input). Moreover, the separation of the three steps insures a higher degree of modularity in the design of the concrete methods following our schema: Any combination of methods that solve an individual task could be used, leaving the user with a vast choice. On the reverse side of the coin,

[4] Termed alternatively as *(minimal) blockers* or *hitting sets*.

total modularity comes with a price: if FGs and FCIs are computed separately, an extra step will be necessary to match an FCI to its FGs.

We describe hereafter a method fitting the above schema which relies exclusively on existing algorithmic techniques. These are combined into a single global procedure, called *Snow-Touch* in the following manner: The FCI computation is delegated to the *Charm* algorithm which is also the basis for *Charm-L*. FGs are extracted by our own vertical miner *Talky-G*. The two methods together with an FG-to-FCI matching technique form the *Touch* algorithm [13]. Finally, precedence is retrieved from FCIs with FGs by the *Snow* algorithm [14] using a ground duality result from hypergraph theory.

In the remainder of this section we summarize the theoretical and the algorithmic background of the above methods which are themselves briefly presented and illustrated in the next section.

2.3 Hypergraphs, transversals, and precedence in closure semi-lattices

The generator computation in [11] exploits the tight interdependence between the intent of a concept, its generators and the intents of its immediate predecessor concepts. Technically speaking, a generator is a *minimal blocker* for the family of *faces* associated to the concept intent and its predecessor intents[5].

Example. Consider the closed itemset (CI) lattice in Figure 1 (c). The CI $ABCDE$ has two faces: $F_1 = ABCDE \setminus AB = CDE$ and $F_2 = ABCDE \setminus ACDE = B$.

It turns out that *blocker* is a different term for the widely known *hypergraph transversal* notion. We recall that a hypergraph [18] is a generalization of a graph where edges can connect arbitrary number of vertices. Formally, it is a pair $(V,\mathcal{E})$ made of a basic vocabulary $V = \{v_1, v_2, \ldots, v_n\}$, the *vertices*, and a family of sets $\mathcal{E}$, the *hyper-edges*, all drawn from V.

A set $T \subseteq V$ is called a *transversal* of $\mathcal{H}$ if it has a non-empty intersection with all the edges of $\mathcal{H}$. A special case are the minimal transversals that are exploited in [11].

Example. In the above example, the minimal transversals of $\{CDE, B\}$ are $\{BC, BD, BE\}$, hence these are the generators of $ABCDE$ (see Figure 1 (c)).

The family of all minimal transversals of $\mathcal{H}$ constitutes the *transversal hypergraph* of $\mathcal{H}$ ($Tr(\mathcal{H})$). A duality exists between a simple hypergraph and its transversal hypergraph [18]: For a simple hypergraph $\mathcal{H}$, $Tr(Tr(\mathcal{H})) = \mathcal{H}$. Thus, the faces of a concept intent are exactly the minimal transversals of the hypergraph composed by its generators.

Example. The bottom node in Figure 1 (c) labelled $ABCDE$ has three generators: BC, BD, and BE while the transversals of the corresponding hypergraph are $\{CDE, B\}$.

[5] A face is the set-theoretic difference between the intents of two concepts bound by a precedence link.

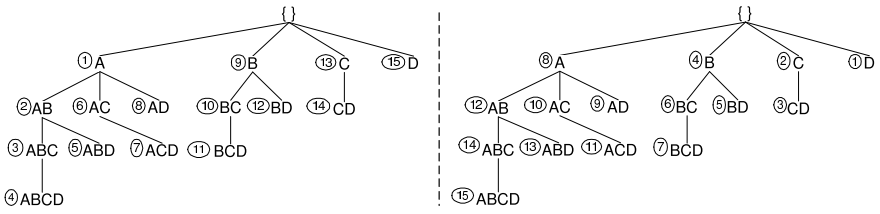


Fig. 2. Left: pre-order traversal with *Eclat*; **Right:** reverse pre-order traversal with *Talky-G*

2.4 Vertical Itemset Mining

Miners from the literature, whether for plain FIs or FCIs, can be roughly split into breadth-first and depth-first ones. Breadth-first algorithms, more specifically the *Apriori*-like [1] ones, apply levelwise traversal of the pattern space exploiting the anti-monotony of the frequent status. Depth-first algorithms, e.g., *Closet* [19], in contrast, organize the search space into a prefix-tree (see Figure 2) thus factoring out the effort to process common prefixes of itemsets. Among them, the *vertical* miners use an encoding of the dataset as a set of pairs (item, tidset), i.e., $\{(i, t(i))|i \in \mathcal{A}\}$, which helps avoid the costly database re-scans.

Eclat [20] is a plain FI miner relying on a vertical encoding at a depth-first traversal of a tree structure, called IT-tree, whose nodes are $X \times t(X)$ pairs. *Eclat* traverses the IT-tree in a pre-order way, from left-to-right [20] (see Figure 2). *Charm* adapts the computing schema of *Eclat* to the rapid construction of the FCIs [6]. It is knowingly one of the fastest FCI-miners, hence its adoption as a component in *Touch* as well as the idea to look for similar technique for FGs. However, a vertical FG miner would be closer to *Eclat* than to *Charm* as it requires no specific speed-up during the traversal (recall that FGs form a downset). In contrast, there is a necessary shift in the test focus w.r.t. *Eclat*: Instead of supersets, subsets need to be examined to check candidate FGs. This, in turn, requires that all such subsets are already tested at the moment an itemset is examined. In other terms, the IT-tree traversal order needs to be a linear extension of $\subseteq$ order between itemsets.

3 The Snow-Touch Algorithm

We sketch below the key components of *Snow-Touch* i.e. *Talky-G*, *Touch*, and *Snow*.

3.1 Talky-G

Talky-G is a vertical FG miner that constructs an IT-tree in a depth-first right-to-left manner [13].

Traversal Of The Generator Search Space

Traversing $\wp(\mathcal{A})$ so that a given set X is processed after all its subsets induces a $\subseteq$-complying traversal order, i.e. a linear extension of $\subseteq$.

In FCA, a similar technique is used by the *Next-Closure* algorithm [21]. The underlying *lectic* order is rooted in an implicit mapping of $\wp(\mathcal{A})$ to $[0 \dots 2^{|\mathcal{A}|}-1]$, where a set image is the decimal value of its characteristic vector w.r.t. an arbitrary ordering $rank : \mathcal{A} \leftrightarrow [1..|\mathcal{A}|]$. The sets are then listed in the increasing order of their mapping values which represents a depth-first traversal of $\wp(\mathcal{A})$. This encoding yields a depth-first *right-to-left* traversal (called *reverse pre-order traversal* in [22]) of the IT-tree representing $\wp(\mathcal{A})$.

Example. See Figure 2 for a comparison between the traversal strategies in *Eclat* (left) and in *Talky-G* (right). Order-induced ranks of itemsets are drawn next to their IT-tree nodes.

The Algorithm

The algorithm works the following way. The IT-tree is initialized by creating the root node and hanging a node for each frequent item below the root (with its respective tidset). Next, nodes below the root are examined, starting from the right-most one. A 1-itemset p in such a node is an FG iff $supp(p) < 100\%$ in which case it is saved to a dedicated list. A recursive exploration of the subtree below the current node then ensues. At the end, all FGs are comprised in the IT-tree.

During the recursive exploration, all FGs from the subtree rooted in a node are mined. First, FGs are generated by "joining" the subtree's root to each of its sibling nodes laying to the right. A node is created for each of them and hung below the subtree's root. The resulting node's itemset is the union of its parents' itemsets while its tidset is the intersection of the tidsets of its parents. Then, all the direct children of the subtree's root are processed recursively in a right-to-left order.

When two FGs are joined to form a candidate node, two cases can occur. Either we obtain a new FG, or a valid FG cannot be generated from the two FGs. A candidate FG is the union of the input node itemsets while its tidset is the intersection of the respective tidsets. It can fail the FG test either by insufficient support (non frequent) or by a strict FG-subset of the same support (which means that the candidate is a proper superset of an already found FG with the same support).

Example. Figure 3 illustrates *Talky-G* on an input made of the dataset $\mathcal{D}$ and a $min\_supp = 1$ (20%). The node ranks in the traversal-induced order are again indicated. The IT-tree construction starts with the root node and its children nodes: Since no universal item exists in $\mathcal{D}$, all items are FGs and get a node below the root. In the recursive extension step, the node E is examined first: Absent right siblings, it is skipped. Node D is next: the candidate itemset DE

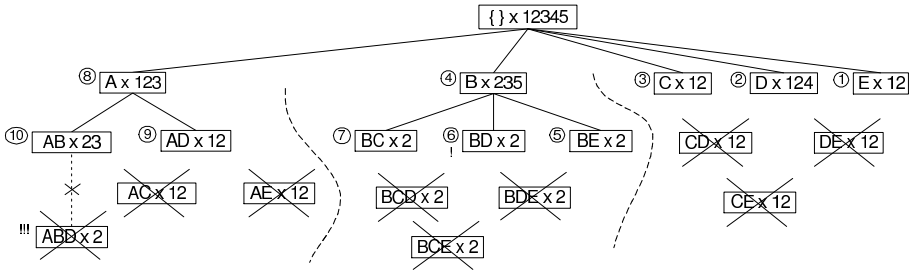


Fig. 3. Execution of *Talky-G* on dataset $\mathcal{D}$ with $min\_supp = 1$ (20%)

fails the FG test since of the same support as E. With C, both candidates CD and CE are discarded for the same reason. In a similar fashion, the only FGs in the subtree below the node of B are BC, BD, and BE. In the case of A, these are AB and AD. ABD fails the FG test because of BD.

Fast Subsumption Checking

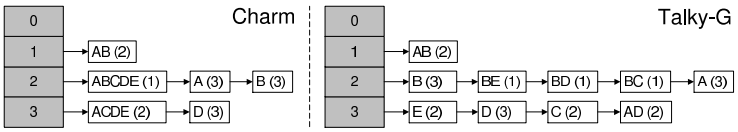
During the generation of candidate FGs, a subsumer itemset cannot be a generator. To speed up the subsumption computation, *Talky-G* adapts the hash structure of *Charm* for storing frequent generators together with their support values. Thus, as tidsets are used for hashing of FGs, two equivalent itemsets get the same hash value. Hence, when tracking a potential subsumee for a candidate X, we check within the corresponding list in the hash table for FGs Y having (i) the same support as X and, if positive outcome, (ii) proper subsets of X (see details in [13]).

Example. The hash structure of the IT-tree in Figure 3 is drawn in Figure 4 (top right). The hash table has four entries that are lists of itemsets. The hash function over a tidset is the modulo 4 of the sum of all tids. For instance, to check whether ABD subsumes a known FG, we take its hash key, 2 mod 4 = 2, and check the content of the list at index 2. In the list order, B is discarded for support mismatch, while BE fails the subset test. In contrast, BD succeeds both the support and the inclusion tests so it invalidates the candidate ABD.

3.2 The Touch Algorithm

The *Touch* algorithm has three main features, namely **(1)** extracting frequent closed itemsets, **(2)** extracting frequent generators, and **(3)** associating frequent generators to their closures, i.e. identifying frequent equivalence classes.

Finally, our method matches FGs to their respective FCIs. To that end, it exploits the shared storage technique in both *Talky-G* and *Charm*, i.e. the hashing on their images (see Figure 4 (top)). The calculation is immediate: as the hash value of a FG is the same as for its FCI, one only needs to traverse



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

Fig. 4. Top: hash tables for dataset $\mathcal{D}$ with $min\_supp = 1$. **Top left:** hash table of *Charm* containing all FCIs. **Top right:** hash table of *Talky-G* containing all FGs. **Bottom:** output of *Touch* on dataset $\mathcal{D}$ with $min\_supp = 1$

the FG hash and for each itemset lookup the list of FCI associated to its own hash value. Moreover, setting both lists to the same size, further simplifies the procedure as both lists will then be located at the same offset within their respective hash tables.

Example. Figure 4 (top) depicts the hash structures of *Charm* and *Talky-G*. Assume we want to determine the generators of $ACDE$ which is stored at position 3 in the hash structure of *Charm*. Its generators are also stored at position 3 in the hash structure of *Talky-G*. The list comprises three members that are subsets of $ACDE$ with the same support: E, C, and AD. Hence, these are the generators of $ACDE$. The output of *Touch* is shown in Figure 4 (bottom).

3.3 The Snow Algorithm

Snow computes precedence links on FCIs from associated FGs [14]. *Snow* exploits the duality between hypergraphs made of the generators of an FCI and of its faces, respectively to compute the latter as the transversals of the former. Thus, its input is made of FCIs and their associated FGs. Several algorithms can be used to produce this input, e.g. *Titanic* [5], *A-Close* [4], *Zart* [23], *Touch*, etc. Figure 4 (bottom) 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 closed itemset lattice.

Example. 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. □

Table 1. Database characteristics

database name	# records	# non-empty attributes	# attributes (in average)	largest attribute
T25I10D10K	10,000	929	25	1,000
MUSHROOMS	8,416	119	23	128
CHESS	3,196	75	37	75
CONNECT	67,557	129	43	129

4 Experimental Evaluation

In this section we discuss practical aspects of our method. First, in order to demonstrate that our approach is computationally efficient, we compare its performances on a wide range of datasets to those of *Charm-L*. Then, we present an application of *Snow-Touch* to the analysis of genomic data together with an excerpt of the most remarkable gene associations that our method helped to uncover.

4.1 Snow-Touch vs. Charm-L

We evaluated *Snow-Touch* against *Charm-L* [8,9]. The experiments were carried out on a bi-processor Intel Quad Core Xeon 2.33 GHz machine running Ubuntu GNU/Linux with 4 GB RAM. All times reported are real, wall clock times, as obtained from the Unix *time* command between input and output. *Snow-Touch* was implemented entirely in Java. For performance comparisons, the authors' original C++ source of *Charm-L* was used. *Charm-L* and *Snow-Touch* were executed with these options: `./charm-l -i input -s min_supp -x -L -o COUT -M 0 -n`; `./leco.sh input min_supp -order -alg:dtouch -method:snow -nof2`. In each case, the standard output was redirected to a file. The diffset optimization technique [24] was activated in both algorithms.[6]

Benchmark datasets. For the experiments, we used several real and synthetic dataset benchmarks (see Table 1). The synthetic dataset T25, using the IBM Almaden generator, is constructed according to the properties of market basket data. The MUSHROOMS database describes mushrooms characteristics. The CHESS and CONNECT datasets are derived from their respective game steps. The latter three datasets can be found in the UC Irvine Machine Learning Database Repository. Typically, real datasets are very dense, while synthetic data are usually sparse. Response times of the two algorithms on these datasets are presented in Figure 5.

Charm-L. *Charm-L* represents a state-of-the-art algorithm for closed itemset lattice construction [8]. *Charm-L* extends *Charm* to directly compute the lattice while it generates the CIs. In the experiments, we executed *Charm-L* with a switch to compute (minimal) generators too using the *minhitset* method. In [9], Zaki and Ramakrishnan present an efficient method for calculating the generators, which is actually the generator-computing method of Pfaltz and Taylor [25].

[6] *Charm-L* uses diffsets by default, thus no explicit parameter was required.

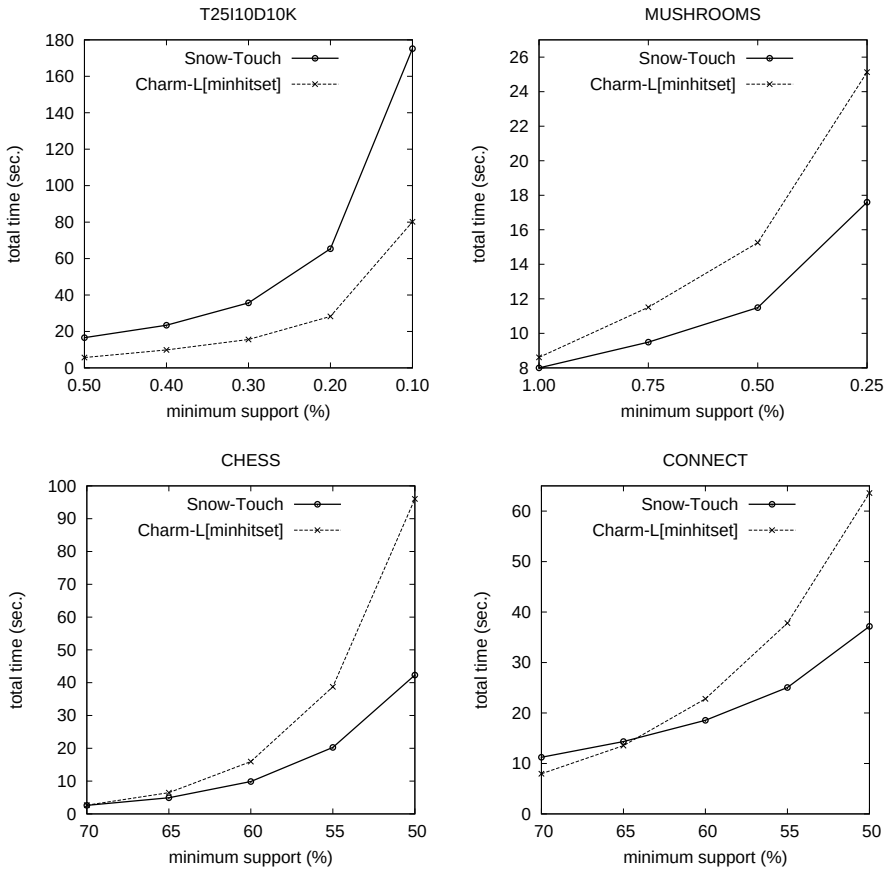


Fig. 5. Response times of *Snow-Touch* and *Charm-L*.

This way, the two algorithms (*Snow-Touch* and *Charm-L*) are *comparable* since they produce *exactly the same output*.

Performance on sparse datasets. On T25, *Charm-L* performs better than *Snow-Touch*. We have to admit that sparse datasets are a bit problematic for our algorithm. The reason is that T25 produces long sparse bitvectors, which gives some overhead to *Snow-Touch*. In our implementation, we use bitvectors to store tidsets. However, as can be seen in the next paragraph, our algorithm outperforms *Charm-L* on all the dense datasets that were used during our tests.

Performance on dense datasets. On MUSHROOMS, CHESS and CONNECT, we can observe that *Charm-L* performs well only for high values of support. Below a certain threshold, *Snow-Touch* gives lower response times, and the gap widens as the support is lowered. When the minimum support is set low enough, *Snow-Touch* can be several times faster than *Charm-L*. Considering that *Snow-Touch* is implemented in Java, we believe that a good C++ implementation could be several orders of magnitude faster than *Charm-L*.

According to our experiments, *Snow-Touch* can construct the concept lattices faster than *Charm-L* in the case of dense datasets. From this, we draw the hypothesis that our direction towards the construction of FG-decorated concept lattices is more beneficial than the direction of *Charm-L*. That is, it is better to extract first the FCI/FG-pairs and then determine the order relation between them than first extracting the set of FCIs, constructing the order between them, and then determining the corresponding FGs for each FCI.

4.2 Analysis of Antibiotic Resistant Genes

We looked at the practical performance of *Snow-Touch* on real-world genomic dataset whereby the goal was to discover meaningful associations between genes in entire genomes seen as items and transactions, respectively.

The genomic dataset was collected from the website of the National Center for Biotechnology Information (NCBI) with a focus on genes from microbial genomes. At the time of writing (June 2011), 1,518 complete microbial genomes were available on the NCBI website.[7] For each genome, its list of genes was collected (for instance the genome with ID CP002059.1 has two genes, *rnpB* and *ssrA*). Only 1,250 genomes out of the 1,518 proved non empty; we put them in a binary matrix of 1,250 rows $\times$ 125,139 columns. With an average of 684 genes per genome we got 0.55% density (i.e., large yet sparse dataset with an imbalance between numbers of rows and of columns).

The initial result of the mining task was the family of *minimal non-redundant association rules* ($\mathcal{MNR}$), which are directly available from the output of *Snow-Touch*. We sorted them according to the confidence. Among all strong associations, the bioinformaticians involved in this study found most appealing the rules describing the behavior of antibiotic resistant genes, in particular, the *mecA* gene. *mecA* is frequently found in bacterial cells. It induces a resistance to antibiotics such as *Methicillin*, *Penicillin*, *Erythromycin*, etc. [26]. The most commonly known carrier of the gene *mecA* is the bacterium known as MRSA (methicillin-resistant *Staphylococcus aureus*).

At a second step, we were narrowing the focus on a group of three genes, *mecA* plus *ampC* and *vanA* [27]. *ampC* is a beta-lactam-resistance gene. AmpC beta-lactamases are typically encoded on the chromosome of many gram-negative bacteria; it may also occur on *Escherichia coli*. *AmpC* type beta-lactamases may also be carried on plasmids [26]. Finally, the gene *vanA* is a vancomycin-resistance gene typically encoded on the chromosome of gram-positive bacteria such as *Enterococcus*. The idea was to relate the presence of these three genes to the presence or absence of any other gene or a combination thereof.

Table 2 shows an extract of the most interesting rules found by our algorithm. These rules were selected from a set of 18,786 rules.

For instance, rule (1) in Table 2 says that the gene *mecA* is present in 85.71% of cases when the set of genes {*clpX*, *dnaA*, *dnaI*, *dnaK*, *gyrB*, *hrcA*, *pyrF*}

[7] http://www.ncbi.nlm.nih.gov/genomes/lproks.cgi

Table 2. An extract of the generated minimal non-redundant association rules. After each rule, the following measures are indicated: support, confidence, support of the left-hand side (antecedent), support of the right-hand side (consequent)

(1) {*clpX*, *dnaA*, *dnaI*, *dnaK*, *gyrB*, *hrcA*, *pyrF*} → {*mecA*} (supp=96 [7.68%]; **conf=0.857 [85.71%]**; suppL=112 [8.96%]; suppR=101 [8.08%])
(2) {*clpX*, *dnaA*, *dnaI*, *dnaK*, *nusG*} → {*mecA*} (supp=96 [7.68%]; **conf=0.835 [83.48%]**; suppL=115 [9.20%]; suppR=101 [8.08%])
(3) {*clpX*, *dnaA*, *dnaI*, *dnaJ*, *dnaK*} → {*mecA*} (supp=96 [7.68%]; **conf=0.828 [82.76%]**; suppL=116 [9.28%]; suppR=101 [8.08%])
(4) {*clpX*, *dnaA*, *dnaI*, *dnaK*, *ftsZ*} → {*mecA*} (supp=96 [7.68%]; **conf=0.828 [82.76%]**; suppL=116 [9.28%]; suppR=101 [8.08%])
(5) {*clpX*, *dnaA*, *dnaI*, *dnaK*} → {*mecA*} (supp=97 [7.76%]; **conf=0.815 [81.51%]**; suppL=119 [9.52%]; suppR=101 [8.08%])
(6) {*greA*, *murC*, *pheS*, *rnhB*, *ruvA*} → {*ampC*} (supp=99 [7.92%]; **conf=0.227 [22.71%]**; suppL=436 [34.88%]; suppR=105 [8.40%])
(7) {*murC*, *pheS*, *pyrB*, *rnhB*, *ruvA*} → {*ampC*} (supp=99 [7.92%]; **conf=0.221 [22.15%]**; suppL=447 [35.76%]; suppR=105 [8.40%])
(8) {*dxs*, *hemA*} → {*vanA*} (supp=29 [2.32%]; **conf=0.081 [8.15%]**; suppL=356 [28.48%]; suppR=30 [2.40%])
(9) {*dxs*} → {*vanA*} (supp=30 [2.40%]; **conf=0.067 [6.73%]**; suppL=446 [35.68%]; suppR=30 [2.40%])

is present in a genome. The above rules have a direct practical use. In one such scenario, they could be used to suggest which antibiotic should be taken by a patient depending on the presence or absence of *certain* genes in the infecting microbe.

5 Conclusion

We presented a new design schema for the task of mining the iceberg lattice and the corresponding generators out of a large context. The target structure directly involved in the construction of a number of association rule bases and hence is of a certain importance in the data mining field. While previously published algorithms follow the same schema, i.e., construction of the iceberg lattice (FCIs plus precedence links) followed by the extraction of the FGs, our approach consists in inferring precedence links from the previously mined FCIs with their FGs.

We presented an initial and straightforward instanciation of the new algorithmic schema that reuses existing methods for the three steps: the popular *Charm* FCI miner, our own method for FG extraction, *Talky-G* (plus an FGs-to-FCIs matching procedure), and the Hasse diagram constructor *Snow*. The resulting iceberg plus FGs miner, *Snow-Touch*, is far from an optimal algorithm, in particular due to redundancies in the first two steps. Yet an implementation thereof within the Coron platform (in Java) has managed to outperform its natural

competitor, *Charm-L* (in C++) on a wide range of datasets, especially on dense ones.

To level the playing ground, we are currently re-implementing *Snow-Touch* in C++ and expect the new version to be even more efficient. In a different vein, we have tested the capacity of our approach to support practical mining task by applying it to the analysis of genomic data. While a large number of associations usually come out of such datasets, many of the redundant with respect to each other, by limiting the output to only the generic ones, our method helped focus the analysts' attention to a smaller number of significant rules.

As a next step, we are studying a more integrated approach for FCI/FG construction that requires no extra matching step. This should result in substantial efficiency gains. On the methodological side, our study underlines the duality between generators and order w.r.t. FCIs: either can be used in combination with FCIs to yield the other one. It rises the natural question of whether FCIs alone, which are output by a range of frequent pattern miners, could be used to efficiently retrieve first precedence, and then FGs.

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, Morgan Kaufmann (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. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining Minimal Non-Redundant Association Rules Using Frequent Closed Itemsets. In: Proc. of the Computational Logic (CL '00). Volume 1861 of LNAI., Springer (2000) 972–986
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. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing Iceberg Concept Lattices with Titanic. Data and Knowl. Eng. **42**(2) (2002) 189–222
6. Zaki, M.J., Hsiao, C.J.: CHARM: An Efficient Algorithm for Closed Itemset Mining. In: SIAM Intl. Conf. on Data Mining (SDM' 02). (Apr 2002) 33–43
7. Zaki, M.J.: Mining Non-Redundant Association Rules. Data Mining and Knowledge Discovery **9**(3) (2004) 223–248
8. Zaki, M.J., Hsiao, C.J.: Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. IEEE Trans. on Knowl. and Data Eng. **17**(4) (2005) 462–478
9. Zaki, M.J., Ramakrishnan, N.: Reasoning about Sets using Redescription Mining. In: Proc. of the 11th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD '05), Chicago, IL, USA (2005) 364–373
10. Godin, R., Missaoui, R.: An incremental concept formation approach for learning from databases. Theoretical Computer Science Journal (133) (1994) 387–419
11. 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
12. Le Floc'h, A., Fisette, C., Missaoui, R., Valtchev, P., Godin, R.: JEN : un algorithme efficace de construction de générateurs pour l'identification des règles d'association. Nouvelles Technologies de l'Information **1**(1) (2003) 135–146

13. Szathmary, L., Valtchev, P., Napoli, A., Godin, R.: Efficient Vertical Mining of Frequent Closures and Generators. In: Proc. of the 8th Intl. Symposium on Intelligent Data Analysis (IDA '09). Volume 5772 of LNCS., Lyon, France, Springer (2009) 393–404
14. Szathmary, L., Valtchev, P., Napoli, A., Godin, R.: Constructing Iceberg Lattices from Frequent Closures Using Generators. In: Discovery Science. Volume 5255 of LNAI., Budapest, Hungary, Springer (2008) 136–147
15. 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
16. Baixeries, J., Szathmary, L., Valtchev, P., Godin, R.: Yet a Faster Algorithm for Building the Hasse Diagram of a Galois Lattice. In: Proc. of the 7th Intl. Conf. on Formal Concept Analysis (ICFCA '09). Volume 5548 of LNAI., Darmstadt, Germany, Springer (May 2009) 162–177
17. 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
18. Berge, C.: Hypergraphs: Combinatorics of Finite Sets. North Holland, Amsterdam (1989)
19. Pei, J., Han, J., Mao, R.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. (2000) 21–30
20. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New Algorithms for Fast Discovery of Association Rules. In: Proc. of the 3rd Intl. Conf. on Knowledge Discovery in Databases. (August 1997) 283–286
21. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer, Berlin/Heidelberg (1999)
22. Calders, T., Goethals, B.: Depth-first non-derivable itemset mining. In: Proc. of the SIAM Intl. Conf. on Data Mining (SDM '05), Newport Beach, USA. (Apr 2005)
23. 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
24. Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining (KDD '03), New York, NY, USA, ACM Press (2003) 326–335
25. Pfaltz, J.L., Taylor, C.M.: Scientific Knowledge Discovery through Iterative Transformation of Concept Lattices. In: Proc. of the SIAM Workshop on Data Mining and Discrete Mathematics, Arlington, VA, USA (2002) 65–74
26. Philippon, A., Arlet, G., Jacoby, G.A.: Plasmid-Determined AmpC-Type β-Lactamases. Antimicrobial Agents and Chemotherapy **46**(1) (2002) 1–11
27. Schwartz, T., Kohnen, W., Jansen, B., Obst, U.: Detection of antibiotic-resistant bacteria and their resistance genes in wastewater, surface water, and drinking water biofilms. Microbiology Ecology **43**(3) (2003) 325–335