

Faster Concept Analysis

Adam D. Troy¹, Guo-Qiang Zhang^{1*}, and Ye Tian²

¹ Department of EECS, Case Western Reserve University
Cleveland, Ohio 44022, U.S.A.

² Information Technology Division
Cleveland Clinic Health Systems, Ohio, U.S.A.

Abstract. We introduce a simple but efficient, multistage algorithm for constructing concept lattices (MCA). A concept lattice can be obtained as the closure system generated from attribute concepts (dually, object concepts). There are two strategies to use this as the basis of an algorithm: (a) forming intersections by joining one attribute concept at a time, and (b) repeatedly forming pairwise intersections starting from the attribute concepts. A straightforward translation of (b) to an algorithm suggests that pairwise intersection be performed among all cumulated concepts. MCA is parsimonious in forming the pairwise intersections: it only performs such operations among the newly formed concepts from the previous stage, instead of cumulatively. We show that this parsimonious multistage strategy is complete: it generates all concepts. To make this strategy really work, one must overcome the need to eliminate duplicates (and potentially save time even further), since concepts generated at a later stage may have already appeared in one of the earlier stages. As considered in several other algorithms in the literature [5], we achieve this by an auxiliary search tree which keeps all existing concepts as paths from the root to a flagged node or a leaf. The depth of the search tree is bounded by the total number of attributes, and hence the time complexity for concept lookup is bounded by the logarithm of the total number of concepts. For constructing lattice diagrams, we adapt a sub-quadratic algorithm of Pritchard [9] for computing subset partial orders to constructing the Hasse diagrams. Instead of the standard expected quadratic complexity, the Pritchard approach achieves a worst-case time $O(N^2/\log N)$. Our experimental results showed significant improvements in speed for a variety of input profiles against three leading algorithms considered in the comprehensive comparative study [5]: Bordat, Chein, and Norris.

1 Introduction

The expanding roles of Formal Concept Analysis (FCA) in many areas make the development of efficient algorithms an important component in any application involving contexts with size beyond small examples. In [5], Kuznetsov and Obiedkov provide an extensive survey and comparative experimental evaluation of algorithms for FCA in the literature.

In this paper we introduce a simple but efficient, multistage algorithm for constructing concept lattices (MCA). Given a formal context, a concept lattice can be obtained

* Corresponding author. Email contact: gq@case.edu

as the closure system generated from attribute concepts (dually, object concepts). There are two strategies in the literature to use this as the basis of an algorithm:

- (a) forming intersections by joining one attribute concept at a time – this falls into the class of algorithms often called incremental concept analysis, and
- (b) repeatedly forming pairwise intersections starting from the attribute concepts.

Bordat [1], Norris [8], and CbO [5] use strategy (a), while Chein [2] and our approach use strategy (b).

A straightforward translation of (b) to an algorithm suggests that pairwise intersection be performed among all cumulated concepts. MCA is parsimonious in forming the pairwise intersections: it only performs such operations among the newly formed concepts from the previous stage, instead of cumulatively. We show that this parsimonious multistage strategy is correct: it generates all concepts. We further demonstrate the speed improvements through a set of experimental evaluations.

In comparative evaluation of algorithms for FCA, several important issues must be carefully considered, in no particular order [5]:

1. Whether or not the computation of order relation (i.e. diagram graph) is separated as a different phase than the construction of concept set.
2. Whether or not the computed concept set contains redundant or repeated concepts.
3. Whether or not the intent and extent of concepts are both maintained throughout an algorithm.
4. Whether the concept set is formed by joining one attribute concept (dually, object concept) at a time iteratively, or alternatively, the concept set is formed by setting the initial concept set to include all attribute concepts (dually, object concepts).

Against these features, our MCA is unique in that it

1. separates the computation of diagram graph as a different phase so we can take advantage of sophisticated sub-quadratic partial ordering algorithms proposed by Pritchard [9];
2. maintains a non-redundant set of concepts and uses an auxiliary search tree for quick concept lookup;
3. keeps only the attribute (dually, object) set, or the intent of concepts throughout the algorithm to eliminate the overhead of extent maintenance (the extent can be looked up afterwards in an efficient way if needed, but it is neither necessary for determining the concepts nor for diagram construction);
4. forms the concept system by a multistage intersection operation from the initial concept set consisting of all object concepts.

Of all the algorithms in the literature (see [5]), our algorithm is the closest in spirit to Chein [2], with key distinctions described in items 1, 3, and 4 above (note that Pritchard's algorithm appeared much later). Particularly, each of our stages is independent of the previous ones in that we do not need to modify any concept sets in the previous stage, as is done in Chein's algorithm. The need to mark off concepts in the previous stage and keep it in the current stage is dictated by a simple theoretical justification (Prop. 3, Section4). By developing a more elaborate combinatorial argument, the

need to mark off concepts in the previous stage is eliminated, with substantial saving in computational time. Correctness of the latter strategy is non-trivial (Prop. 4, Section 4).

To demonstrate that the theoretical advantage of MCA translates to tangible improvement in practice, we performed comparative experimental evaluations against the leading performers from the Kuznetsov-Obiedkov survey [5]: Bordat, Chein, and Norris. The experimental results demonstrated significant improvements in the construction of concept set. They also demonstrated interesting improvements in the diagram graph construction when concept lattice sizes are low-degree polynomials (in terms of context size), by employing Pritchard’s approach.

In performing the comparative experimental evaluation, we have been careful in taking into account a number of factors that may influence the result, as suggested in [5]:

- Using a common computational environment for all algorithms under consideration. We implemented Bordat, Chein, MCA, and Norris all in Python, with optimization strategy applied as long as we see fit to do it.
- Using a variety of input context, with varying parameters testing different aspects of each algorithm evaluated.
- Validating the algorithms for both manual and test cases to make sure that the results from all the algorithms agree with each other. We do this by visually inspecting the concept lattice diagrams rendered using both ConExp [13] and Graphviz [4].

In the end, we are surprised that significant improvements in concept analysis algorithm can still be made, particularly by using a simple idea (with a more demanding theoretical justification). Maybe this is exactly Occam’s Razor at work.

2 Preliminaries

We follow the notation of [3] in this paper. Readers are referred to [3] and [14] for further details. For any set A , let $\mathcal{P}(A)$ denote the powerset of A . A subset C of the powerset $\mathcal{P}(A)$ is called a *closure system* on A if C is closed under arbitrary intersections, i.e., for every $X \subseteq C$, $\bigcap X \in C$. By convention, the whole space A is always a member of a closure system C . A *closure operator* on A is a (self-map) function $\varphi : \mathcal{P}(A) \rightarrow \mathcal{P}(A)$ which is inflationary ($X \subseteq \varphi(X)$), monotonic ($X \subseteq Y \Rightarrow \varphi(X) \subseteq \varphi(Y)$), and idempotent ($\varphi(\varphi(X)) = \varphi(X)$).

Proposition 1. *Define a closed set with respect to a closure operator $\varphi : \mathcal{P}(A) \rightarrow \mathcal{P}(A)$ to be a fixed point of φ . Then closed sets of φ are precisely sets of the form $\varphi(X)$. The collection of closed sets $\{\varphi(X) | X \in \mathcal{P}(A)\}$ forms a closure system on A .*

For closure systems C_1 and C_2 over A , let $C := C_1 \cap C_2$. One can check that C is again a closure system over A . In general, arbitrary intersections of closure systems remain a closure system.

Lemma 1. *Let A be a set. Then the set of all closure systems over A forms a (meta) closure system over $\mathcal{P}(A)$. Every subset of $\mathcal{P}(A)$ generates a closure system over A , which is the smallest closure system containing the starting subset.*

Concept lattices can be viewed as a closure system generated from a subset of a certain powerset.

Proposition 2. *Let $\mathbf{K} = (G, M, I)$ be a formal context. Then its concept lattice \mathcal{BK} is isomorphic to the closure system generated by the set $\{\{g\}' \mid g \in G\}$ and dually, \mathcal{BK} is inverse-isomorphic to the closure system generated by the set $\{\{m\}' \mid m \in M\}$.*

This brings flexibility for procedures for constructing concept lattices. For example, one can partition G into $A \cup B = G$ with $A \cap B = \emptyset$, find the closure system generated by $\{\{a\}' \mid a \in A\}$ and $\{\{b\}' \mid b \in B\}$, respectively, and then find the closure system generated by the union of the two closure systems. This view provides an easy-to-understand, straightforward way to justify the correctness of the class of “divide-and-conquer” algorithms in the literature (for which correctness proofs are often omitted).

3 MCA and Example

For terminology, we call object concepts or attribute concepts in a neutral way primitive concepts. In each particular setting, “primitive concept” will refer to either object concept or attribute concept, but not both.

Because concept sets can be viewed as closure systems generated by primitive concepts, an immediate idea is to start from these singleton-generated concepts and repeatedly perform intersections until no new concepts can be formed. Although correct, this naïve approach may involve redundant computations in two ways. One is that intersections of different primitive concepts may give the same resulting set, and hence removing redundancy can reduce the number of potential intersections needed. The second is that pairwise intersections may need only be performed on a subset of existing concepts, instead of all existing ones cumulated, even though no redundancy exists. Being parsimonious in both results in a multistage algorithm, as in Algorithm 1.

To illustrate how MCA works, we applied it to the “Living Beings” context from [3]. The context is redisplayed here for easy reference:

	a	b	c	d	e	f	g	h	i
1	×	×					×		
2	×	×					×	×	
3	×	×	×				×	×	
4	×		×				×	×	×
5	×	×		×		×			
6	×	×	×	×		×			
7	×		×	×	×				
8	×		×	×		×			

We have the following concepts, generated in stages:

```

Input: context  $(G, M, I)$ 
Output: concept set  $C$ 
1 Insert  $M$  in SearchTree;
2 Insert each member of  $C_1$  in SearchTree if it is not already in, where  $C_1 := \{\{g\} | g \in G\}$ ;
3  $i = 2$ ;
4 while  $|C_{i-1}| > 1$  do
5    $C_i = \{\}$ ;
6   for each pair of (distinct) concepts  $c_j, c_k$  in  $C_{i-1}$  do
7      $candidate = c_j \cap c_k$ ;
8     if  $candidate$  not in SearchTree then
9        $C_i = C_i \cup \{candidate\}$ ;
10    end
11     $i + +$ ;
12  end
13 end
14  $C = \bigcup_i C_i$ ;

```

Algorithm 1: The Multistage Concept Analysis Algorithm

Stage 0		
	0	abcdefghi
	1	abg
	2	abgh
	3	abcgh
	4	acghi
	5	abdf
	6	abcdf
	7	acde
	8	acdf

Stage 1			
	1	9	abg
		10	ag
		11	ab
		12	a
	2	13	agh
	3	14	acgh
		15	abc
		16	ac
	4		
	5	17	ad
		18	adf
	6	19	acd

The second column in the table for Stage 1 indicates the concept (number) in the previous stage that has been used to obtain potential new pairwise intersections. Each new concept is given a consecutive number, which can be used for reference and book-keeping for the next stage, particularly for manual examples.

Several remarks are in order. First, we did not display Stage 2 which produces no new concepts although this step is needed to ensure the proper termination of the algorithm. Second, the total number of generated concepts is 19, the same as illustrated in [3]. Third, during the “execution” MCA we referred neither to the original context, nor to the extent of any concept. One can easily incorporate a data structure for looking up the extent of a concept, *after* the concepts are already determined by MCA. In contrast, many algorithms in the literature need to have both extent and intent to work properly.

4 Correctness

The correctness of MCA can be shown by induction on the number of primitive concepts used in an intersection. The following simple observation is the theoretical basis of Chein's algorithm. We mention it here because the justification for our MCA algorithm is based on a similar idea, but a more elaborate combinatorial argument is needed.

For notational preparation, define $S \pitchfork T := \{s \cap t \mid s \in S, t \in T\}$, where S and T are collections of subsets. With respect to a given formal context (G, M, I) , define $L_0 := \{G\}$, $L_1 := \{\{g\}' \mid g \in G\}$, and for $i > 1$, $L_i := L_{i-1} \pitchfork L_{i-1}$.

Proposition 3 (Chein). *With respect to a given formal context (G, M, I) ,*

$$L = \bigcup_{0 \leq i \leq |G|} L_i,$$

where L is the set of concepts of (G, M, I) .

The proof amounts to an easy induction and we briefly highlight the inductive step. Suppose L_i contains all concepts determined by intersections of $i \geq 0$ primitive concepts. Then, L_{i+1} , formed by pairwise intersections of concepts in L_i , contains all concepts obtained by intersections of $i+1$ primitive concepts. This can be seen by rewriting an intersection A of $i+1$ primitive concepts as the pairwise intersection of two concepts B and C in L_i : $A = B \cap C$, where B is the intersection of the first i primitive concepts used for A , and C is the intersection of the primitive concepts omitting the first one, as the following equation shows:

$$\bigcap_{1 \leq k \leq i+1} \{g_k\}' = \bigcap_{1 \leq k \leq i} \{g_k\}' \cap \bigcap_{2 \leq k \leq i+1} \{g_k\}'.$$

Note that L_{i+1} may contain concepts obtained by intersections of other than $i+1$ primitive concepts, because of potential degenerations as well as concepts using $2i$ primitives. There may also be redundancies in that we cannot ensure that L_i and L_{i+1} are non-overlapping. Non-overlapping can be checked by looking up a search tree which contains all existing concepts, as the implementations of several existing algorithms in the literature [5].

Our MCA algorithm uses a different sequence of sets, defined as follows:

$$\begin{aligned} S_0 &:= \{G\}, \\ S_1 &:= \{\{g\}' \mid g \in G\}, \text{ and} \\ S_{i+1} &:= (S_i \pitchfork S_i) - \bigcup_{1 \leq k \leq i} S_k \end{aligned}$$

for $i \geq 1$. The key distinction from Chein lies in the removal of all existing concepts $\bigcup_{1 \leq k \leq i} S_k$ when forming S_{i+1} , for each stage. This way, only newly generated (and necessary) concepts are kept for subsequent stages, resulting in potentially huge savings in computational cost.

Proposition 4 (Correctness of MCA). *With respect to a given formal context (G, M, I) ,*

$$L = \bigcup_{0 \leq i \leq |G|} S_i.$$

Before providing a proof, we need a helper equivalence relation.

Definition 1. *Let (G, M, I) be a formal context. Two sets $X, Y \subseteq G$ are equivalent if*

$$\bigcap \{\{g'\} | g \in X\} = \bigcap \{\{g'\} | g \in Y\}.$$

When X and Y are equivalent, we write $X \equiv Y$. We call a subset of G irreducible if it is not equivalent to any of its proper subsets.

Proof (Proposition 4). It suffices to show by (course of value) induction in i that each concept of the form $\bigcap \{\{g'\} | g \in X\}$ with X irreducible and $|X| = i$ belongs to S_t , where $t = 1 + \lceil \log_2 i \rceil$. In other words, t is an integer such that $2^{t-2} < i \leq 2^{t-1}$ for $i \geq 2$ (we fix $t = 0$ for $i = 0$ and $t = 1$ for $i = 1$).

The base cases ($i = 0, 1$) follow from the definition of S_i . For the induction step, assume that for some $k > 0$, all irreducible subsets X with $|X| = j \leq k$ determine concepts $\bigcap \{\{g'\} | g \in X\}$ belonging to $S_{1+\lceil \log_2 j \rceil}$. We show that for an irreducible set Y with $|Y| = k + 1$, $\bigcap \{\{g'\} | g \in Y\}$ belongs to $S_{1+\lceil \log_2 (k+1) \rceil}$.

Let $Y \subseteq G$ be an irreducible set with $|Y| = k + 1$. We have

$$\bigcap \{\{g'\} | g \in Y\} = \bigcap \{\{g'\} | g \in Y_1\} \cap \bigcap \{\{g'\} | g \in Y_2\},$$

where Y_1, Y_2 are subsets of Y with sizes equal to $\lceil (k+1)/2 \rceil$, $|Y_1 \cap Y_2| \leq 1$, and $Y = Y_1 \cup Y_2$. In other words, Y_1, Y_2 is a partition of Y into two equal-sized subsets when $k + 1$ is even, and Y_1, Y_2 is *almost* an equal-sized partition of Y when $k + 1$ is odd – they share a single common element.

When $k + 1$ is even, Y_1 and Y_2 are themselves irreducible. We have, by induction hypothesis, $\bigcap \{\{g'\} | g \in Y_i\} \in S_{1+\lceil \log_2 (k+1)/2 \rceil}$ for $i = 1, 2$. Therefore, $\bigcap \{\{g'\} | g \in Y\} \in S_{1+\lceil \log_2 (k+1) \rceil}$ by the definition of S_i .

When $k + 1$ is odd, Y_1 and Y_2 must also be irreducible since any subset of an irreducible set is irreducible. By induction hypothesis, we have $\bigcap \{\{g'\} | g \in Y_i\} \in S_{1+\lceil \log_2 (k+2)/2 \rceil}$ for $i = 1, 2$. However, $\lceil \log_2 (k + 2) \rceil = \lceil \log_2 (k + 1) \rceil$ when k is even. Therefore, $\bigcap \{\{g'\} | g \in Y\} \in S_{1+\lceil \log_2 (k+1) \rceil}$ again. \square

From the above proof we can see that, incidentally,

$$L = \bigcup_{0 \leq i \leq 1 + \lceil \log_2 |G| \rceil} S_i.$$

5 Computing Diagram Graph Using Pritchard's Sub-quadratic Algorithm

In [9,10], Pritchard addresses the following problem: given a collection

$$\mathcal{F} = \{S_1, S_2, \dots, S_k\},$$

where $S_i \subseteq D$ for a fixed set D , compute the *complete subset graph*, where the vertices are members of \mathcal{F} , and there is an edge from S to S' iff $S' \subseteq S$.

Some notations are needed first. For any subset y of D , let

$$\mathcal{F}.y := \{x \in \mathcal{F} \mid y \subseteq x\}.$$

Then $\mathcal{F}.d$ stands for the sub-collection of all subsets in \mathcal{F} containing d , and $x \in \mathcal{F}.y$ iff $y \subseteq x$ iff $x \in \bigcap_{d \in y} \mathcal{F}.d$. Therefore,

Lemma 2 (Pritchard). *For any $y \in \mathcal{F}$, $\mathcal{F}.y = \bigcap_{d \in y} \mathcal{F}.d$.*

Hence, the intersection $\bigcap_{d \in y} \mathcal{F}.d$ contains all supersets of y . Now suppose \mathcal{F} is the collection of all concepts (intents, say). To find all the parents of x in the concept lattice algorithmically, one first finds $\mathcal{F}.d$ for each $d \in x$, and then computes the intersection $\bigcap_{d \in x} \mathcal{F}.d$, and then remove y from it. To find all upper neighbors of y , one can further remove all sets in $\bigcup \{\mathcal{F}.z \mid z \in \bigcap_{d \in x} \mathcal{F}.d - \{y\}\}$, from $\bigcap_{d \in x} \mathcal{F}.d - \{y\}$. This gives us the following algorithm, which achieves the optimal bound $O(N^2/\log N)$ as shown in [9], where N is the sum of the cardinalities of all the sets in the collection \mathcal{F} .

The lattice graph construction algorithm is given in Algorithm 2. It is quite elegant. Inputs to this algorithms are the attribute set M and the concept set C computed by some other algorithm, e.g. MCA. Lines 1–3 construct a list, S_m , for each attribute $m \in M$ which contains the index of each concept that contains the particular attribute. Lines 4–6, the linking phase, identify the supersets P_c , of each concept by computing the intersection of the concept list, S_m (computed in the previous step), for each attribute m belonging to concept, c . The final phase, lines 7–9, removes the links to concepts other than direct parents by removing those concepts that are the parents of the parents of the current concept. This is done by removing the union of the parental list P_i for each parent concept i with concept c in P_c .

The second phase, lines 4–6, is the workhorse step of this algorithm. The speed advantage of this algorithm stems from only having to compute $|c|$ number of (the number of attributes in a concept, generally a small number) chained intersections to find the parents for each concept. The intersections are fast to compute because the sets over which they operate are usually short, particularly after one or more intersections in the chain is already computed. The first step can also be completed cheaply as part of the concept construction algorithm. The final step largely consists of union operations, which are relatively inexpensive.

6 Comparative Experimental Study

This section compares the performances of various FCA algorithms in the literature with our algorithms for MCA (Algorithm 1) and diagram graph construction (Algorithm 2). In particular we evaluate the concept construction algorithms of Chein [2] and Norris [8], the diagram graph construction algorithm of Valtchev [12], and the concept and lattice construction algorithm of Bordat [1], against the corresponding algorithms using Algorithm 1 and Algorithm 2, respectively. We also implemented Lindig's algorithm [6] but the result was not interesting enough to be included. All algorithms were

```

Input: attribute set  $M$ , concept set  $C$ 
Output: parent sets  $P_i$ 
1 for  $m \in M$  do
2   |  $S_m = \{c | c \in C \wedge m \in c\}$ ;
3 end
4 for  $c \in C$  do
5   |  $P_c = \bigcap \{S_m | m \in c\}$ ;
6 end
7 for each  $P_c$  do
8   |  $P_c = P_c / \bigcup \{P_i | i \in P_c\}$ 
9 end

```

Algorithm 2: Diagram Graph Construction

implemented in Python on a MacBook Pro 2.33 GhZ and 2 GB RAM. The algorithms are expected to uniformly perform better using C and a desktop computer.

Contexts used in our experiments were randomly generated according to these parameters: $|G|$ for the number of objects, $|M|$ for the number of elements, and $|g'|$ for the number of attributes per object.

6.1 Concept Set Only

Here we compare the performances of algorithms that just compute a concept set. In applications such as data mining, concept sets (as opposed to the complete lattice diagram) are of primary interest. We compare Algorithm 1 with the algorithms of Chein [2] and Norris [8]. These are the best performing algorithms reported in [5]. We timed each algorithm on contexts with various sizes and densities where density refers to the number of attributes belonging to an object. Varying these parameters gives a more complete picture of the performance of each algorithm.

As can be seen from Figs. 1a–1c in the Appendix, MCA outperformed the other algorithms in all experiments. Chein was only slightly slower than MCA on sparse contexts (Fig. 1a), but Norris was significantly slower. On medium density contexts (Fig. 1b), Chein and Norris were similar but much slower than MCA. On denser contexts (Fig. 1c), Chein lagged far behind Norris, which was closely behind MCA.

6.2 Lattice Diagram Only

The Pritchard lattice diagram construction algorithm (Algorithm 2) was compared with the algorithm of Valtchev, Missaoui and Lebrun [12]. As discussed earlier, Algorithm 2 computes all the upper-neighbors (supersets) of each node and then removes those which are not direct parents. Valtchev-Missaoui-Lebrun begins at the top of the lattice and then recursively identifies the lower neighbors of each concept. In comparing the two algorithms we computed the concept set using MCA and then timed the construction of lattices using both algorithms. Experiments were run using the same parameters as given in the previous subsection. Figs. 2–4 show the superior performance of the Pritchard in all cases. This may be attributed to the computing of supersets rather than all lower neighbors.

6.3 Concept Set and Lattice Diagram Together

Here we compare the performance of MCA+Pritchard with Bordat [1]. In [5], Bordat was featured as the best algorithm that constructed the concepts and lattice diagrams simultaneously. Our experiments revealed mixed results. For sparse contexts (Fig. 5), MCA+Pritchard outperformed Bordat by a fair amount. For medium density contexts (Fig. 6), MCA+Pritchard outperformed Bordat until $|G| = 75$, and then a reversal occurred. For denser contexts (Fig. 7), Bordat outperformed MCA when the number of objects exceeded 30.

7 Conclusion

The introduced Multistage Concept Analysis (MCA) Algorithm is simple but also efficient, demonstrated through a rigorous theoretical analysis and an experimental evaluation. It is the fastest algorithm we have seen so far for constructing concept set.

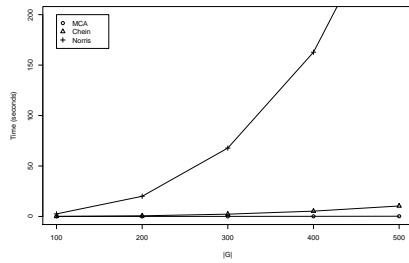
This paper also brings Pritchard's Algorithm to concept analysis. When concept generation and diagram generation are combined, MCA+Pritchard outperforms Valtchev-Missaoui-Lebrun for contexts with lattice sizes bounded by a low-degree polynomial in $|G|$ or $|M|$. For contexts whose lattice sizes grow faster than low-degree polynomials, incremental, neighborhood approaches are expected to perform better, since the time complexity of these algorithms is bounded by a product of the lattice size and a low-degree polynomial in $|G|$ and $|M|$, rather than a sub-quadratic function of the lattice size. It would be interesting to bring into the picture more recent incremental algorithms, such as those proposed in [7,11].

References

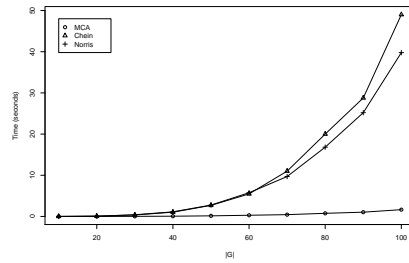
1. J.P. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Math. Sci. Hum.*, Vol 96: 31 - 47, 1986.
2. M. Chein. Algorithme de recherche des sous-matrices premieres d'une matrice. *Bull. Math. Soc. Sci. Math. R.S. Roumanie*, Vol 13: 21 - 25, 1969.
3. B. Ganter and R. Wille. Formal Concept Analysis. Springer, Berlin, 1999.
4. <http://www.graphviz.org/>
5. Sergei O. Kuznetsov, Sergei A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Intell.* 14(2-3): 189-216, 2002.
6. C. Lindig. Fast Concept Analysis. *Working with Conceptual Structures - Contributions to ICCS 2000*, 2000.
7. D. van der Merwe, S. Obiedkov and D. Kourie. AddIntent: A new incremental algorithm for constructing concept lattices. *Lecture Notes in Computer Science* Vol 2961: 372-385, 2004.
8. E.M. Norris. An algorithm for computing the maximal rectangles in a binary relation. *Rev. Roumaine Math. Pures et Appl.*, Vol 23(2): 243 - 250, 1978.
9. P. Pritchard. On computing the subset graph of a collection of sets. *J. Algorithms* 33(2): 187-203, 1999.
10. P. Pritchard. A fast bit-parallel algorithm for computing the subset partial order. *Algorithmica* 24(1): 76-86, 1999.

11. P. Valtchev, R. Missaoui, R. Godin and M. Meridji. Generating frequent itemsets incrementally: two novel approaches based on Galois lattice theory. *Journal of Experimental & Theoretical Artificial Intelligence* 14(2/3): 115-142, 2002.
12. P. Valtchev, R. Missaoui and P. Lebrun. A fast algorithm for building the Hasse diagram of a Galois lattice. *dans Actes du Colloque LaCIM 2000*, pp. 293-306, Montreal, 2000.
13. S. Yevtushenko. ConExp. <http://sourceforge.net/projects/conexp>
14. G.-Q. Zhang. Chu spaces, formal concepts, and domains. *Electronic Notes in Computer Science*, Vol. 83, 16 pages, 2003.
15. G.-Q. Zhang and G. Shen. Approximable Concepts, Chu spaces, and information systems. In V. De Paiva and V. Pratt (eds.) *Theory and Applications of Categories*, Special Volume on Chu Spaces: Theory and Applications, Vol. 17, No. 5, pp. 80-102, 2006.

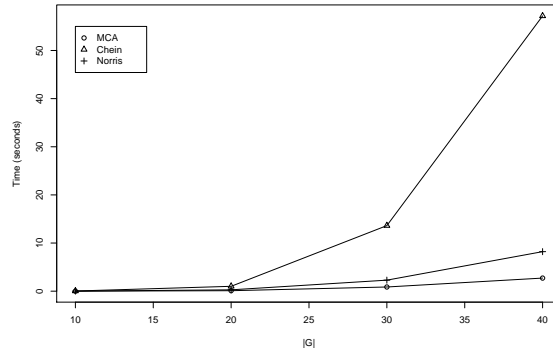
A Figures



(a) $|g'| = 4$



(b) $|g'| = 15$



(c) $|g'| = 30$

Fig. 1: Comparison of time to compute concept set only – MCA vs. Chein and Norris for contexts with $|M| = 100$, $|g'| = 4$ (sub-figure 1a), $|g'| = 15$ (sub-figure 1b), $|g'| = 30$ (sub-figure 1c), and $|G|$ between 10 and 50.

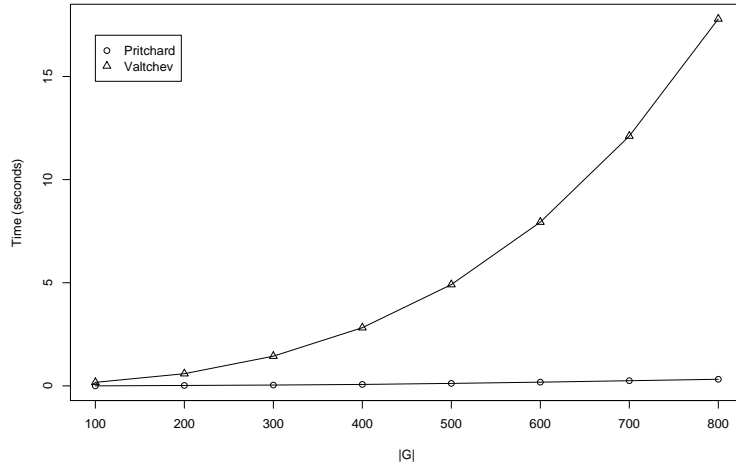


Fig. 2: Comparison of time to construct lattice diagrams only – Pritchard vs. Valtchev for concept sets from contexts with $|M| = 100$, $|g'| = 4$ and $|G|$ between 100–800.

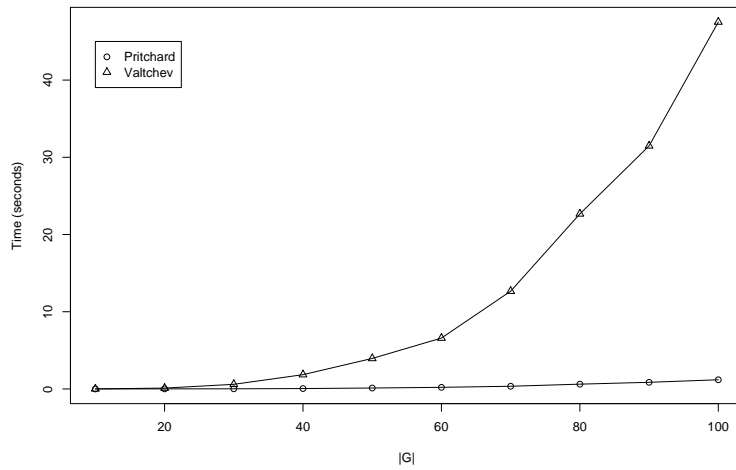


Fig. 3: Comparison of time to construct lattice diagrams only – Pritchard vs. Valtchev for concept sets from contexts with $|M| = 100$, $|g'| = 15$ and $|G|$ between 10–100.

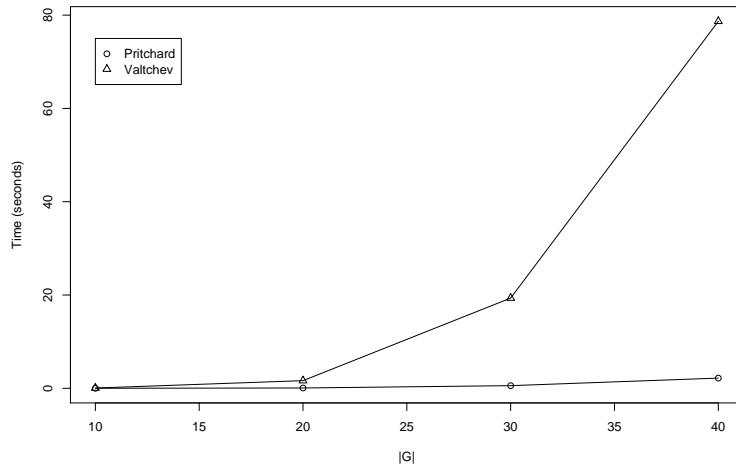


Fig. 4: Comparison of time to construct lattice diagram only – Pritchard vs. Valtchev for concept sets from contexts with $|M| = 100$, $|g'| = 30$ and $|G|$ between 10–40.

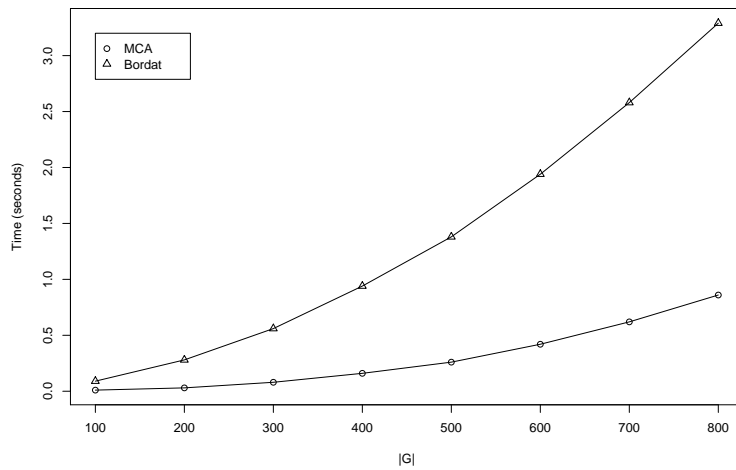


Fig. 5: Comparison of time to compute concept set and lattice diagram – MCA+Pritchard vs. Bordat for contexts with $|M| = 100$, $|g'| = 4$ and $|G|$ between 100–800.

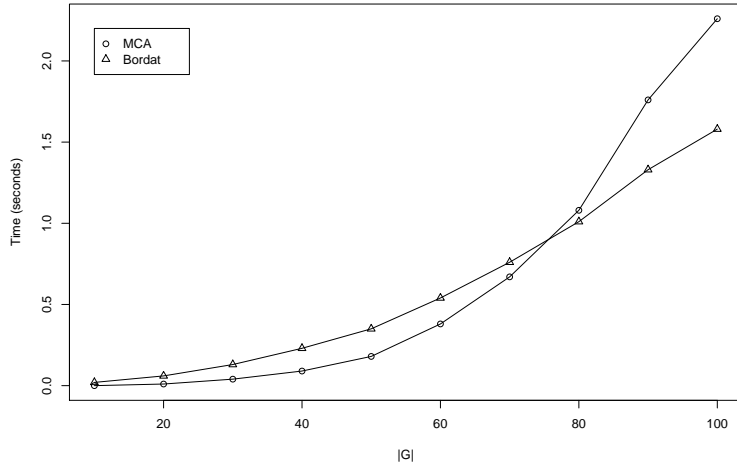


Fig. 6: Comparison of time to compute concept set and lattice diagram – MCA+Pritchard vs. Bordat for contexts with $|M| = 100$, $|g'| = 15$ and $|G|$ between 10–100.

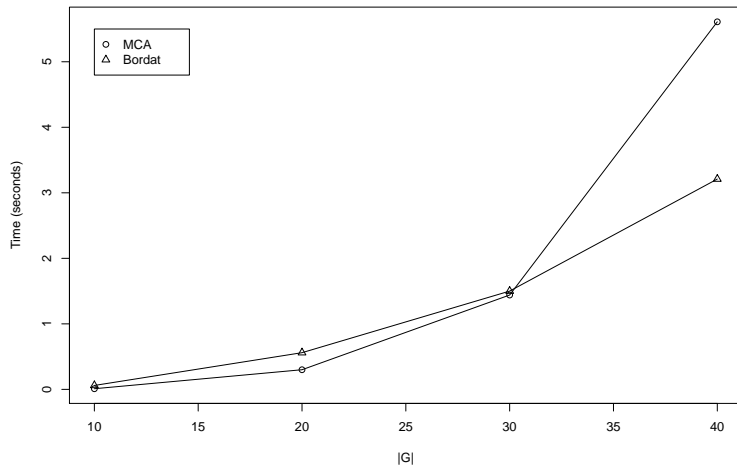


Fig. 7: Comparison of time to compute concept set and lattice diagram – MCA+Pritchard vs. Bordat for contexts with $|M| = 100$, $|g'| = 30$ and $|G|$ between 10–40.