INTERNATIONAL JOURNAL OF GENERAL SYSTEMS, 2015
http://dx.doi.org/10.1080/03081079.2015.1072928

Taylor & Francis
Taylor & Francis Group

A lattice-free concept lattice update algorithm

Jan Outrata

Department of Computer Science, Palacký University Olomouc, Olomouc, Czech Republic

ABSTRACT

Upon a change of input data, one usually wants an update of output computed from the data rather than recomputing the whole output over again. In Formal Concept Analysis, update of concept lattice of input data when introducing new objects to the data can be done by any of the so-called incremental algorithms for computing concept lattice. The algorithms use and update the lattice while introducing new objects to input data one by one. The present concept lattice of input data without the new objects is thus required by the computation. However, the lattice can be large and may not fit into memory. In this paper, we propose an efficient algorithm for updating the lattice from the present and new objects only, not requiring the possibly large concept lattice of present objects. The algorithm results as a modification of the Close-by-One algorithm for computing the set of all formal concepts, or its modifications like Fast Close-by-One, Parallel Close-by-One or Parallel Fast Close-by-One, to compute new and modified formal concepts and the changes of the lattice order relation only. The algorithm can be used not only for updating the lattice when new objects are introduced but also when some existing objects are removed from the input data or attributes of the objects are changed. We describe the algorithm, discuss efficiency issues and present an experimental evaluation of its performance and a comparison with the AddIntent incremental algorithm for computing concept lattice.

ARTICLE HISTORY
Received 15 June 2014
Accepted 15 December 2014

KEYWORDS
Concept lattice; update algorithm; formal concepts

1. Introduction

In applications of Formal Concept Analysis (FCA) (Ganter and Wille 1999; Wille 1982), the input data are often not fixed during the life of the application and concept lattice is not computed from the data once. The changes of input data result in corresponding changes of concept lattice of the data. To compute the new concept lattice, one would like to compute the changes only and "update" the present lattice instead of recomputing the whole lattice again from the new data.

A particular change of object-attribute relational data which are processed by FCA, namely the introduction of new objects described by given values of attributes, can be

CONTACT Jan Outrata jan.outrata@upol.cz

This paper is a revised and extended version of the paper presented at the CLA 2013 conference and included in the proceedings. The algorithms for computing concept lattice and its update were substantially redesigned and improved, a pseudocode of the latter, which was not included in the conference paper, was added, and the experimental evaluation was extended (more benchmark data-sets, memory usage).

handled by the so-called incremental algorithms for computing concept lattice. Several such algorithms were in the past described in the literature, for instance Norris's (1978), Godin, Missaoui, and Alaoui (1995), the incremental concept lattice update algorithms presented in Carpineto and Romano (2004) (Object Intersections) or a generic scheme and a framework for such algorithms in Valtchev, Hacene, and Missaoui (2003) and Valtchev, Missaoui, and Godin (2008), respectively. A more recent one, considered one of the up-to-date most efficient incremental algorithms for computing concept lattice, is the AddIntent algorithm (van der Merwe, Obiedkov, and Kourie 2004). The algorithms build/update the lattice incrementally by adding objects of input data, one by one, to the present concept lattice computed so far from already processed objects (starting from the first object and the empty concept lattice). The lattice is requisite for the computation and the present lattice of data before the introduction of new objects is required for the update of the lattice after adding the objects. However, for bigger input data that lattice becomes quickly very large and may not fit into memory (or even hard disk) for the computation. Moreover, an application which uses the lattice may (need) not store it at all – due to the space requirements it may even be impossible – or may store it only partially. Alternatively, the lattice can be (partially) stored at a different, e.g. presentation only, place than the computation place. Or there can be other drawbacks or disadvantages of keeping the whole lattice for the computation. In addition, handling of the other changes of input data like deletion or alteration (i.e. changing of values of attributes) of existing objects would call for more or less extensive modifications of a particular incremental algorithm.

In the following, we propose an efficient algorithm for updating the concept lattice, i.e. computing the lattice changes resulting by input data changes, from the present (already processed) and new input data objects only. The algorithm does not require the concept lattice of present objects for the computation, thus its memory requirements are negligible in comparison to incremental concept lattice algorithms. The algorithm results as a modification of Kuznetsov's Close-by-One (CbO) algorithm (Kuznetsov 1989, 1993, 1999) for computing the set of all formal concepts or any of its recent derivatives including Fast Close-by-One (FCbO) (Outrata and Vychodil 2012), Parallel Close-by-One (PCbO) (Krajca, Outrata, and Vychodil 2010b) or Parallel Fast Close-by-One (PFCbO) (Krajca, Outrata, and Vychodil 2010a). When introducing new objects to input data (resulting in new data) the modified algorithm computes and outputs the resulting new and modified formal concepts only together with the respective changes in the lattice order relation.

Note here that:

(1) [new formal concept] …a concept in new data with (some of the) introduced objects in its object set (extent) and with attribute set (intent) not equal to intent of any concept in the data before the update;
[modified formal concept] …a concept in new data with the same intent as some existing concept in data before the update and enlarging its extent by (some of the) introduced objects;
[old formal concept] …all other formal concepts in new data as well as the data before update;

(2) introducing new objects to input data cannot result in removal of formal concepts from the concept lattice (Godin, Missaoui, and Alaoui 1995; Valtchev and Missaoui 2001);

(3) since the concept lattice is not required and not used by the introduced algorithm, the computed changes of the lattice are output only and have to be applied where the (part of the) lattice is stored.

Moreover, with the same algorithm, we can easily handle also the change of input data consisting of deleting existing objects. Just consider the new objects from above to be some objects of input data to be deleted from the data and the present objects to be the objects which remain in the input data after the deletion. The computed formal concepts are then to be interpreted as the concepts to remove or modify, together with the "inverse" changes of the lattice order relation. Obviously, the change by altering objects can be handled by first deleting and then by introducing the altered objects, interpreting the output formal concepts accordingly. Of course, there is a possibility to combine the necessary actions of creating, modifying and deleting formal concepts into a single, more efficient, algorithm but this would make the algorithm (and its description) just needlessly more complicated. Finally, changes of input data by introducing new and deleting or altering existing attributes can be handled (better than by altering objects) by the algorithm redescribed with objects and attributes switched, or by the present algorithm with objects and attributes switched in the input data (transposed data) and in the output formal concepts.

The algorithm is described in Section 2, for the case of changing input data by introducing new objects, including an illustrative example. In Section 3, we present an experimental evaluation of performance of the algorithm and a comparison with the AddIntent algorithm (van der Merwe, Obiedkov, and Kourie 2004).

2. The algorithm

We describe the algorithm as a modification of the FCbO algorithm (Outrata and Vychodil 2012), as it happens to be one of the up-to-date most efficient (sequential/serial) derivatives of CbO (Kirchberg et al. 2012). In essence, the modifications equally apply also to other CbO derivatives and CbO itself. A deep knowledge of FCbO is not required in the description, however, a basic knowledge is beneficial. We recall (the parts of) the algorithm when necessary, otherwise we refer to Outrata and Vychodil (2012). The modification consists of two parts: (1) computing new and modified formal concepts only when introducing new objects to input data and (2) determining changes in the lattice order relation. We describe the parts separately in Sections 2.1 and 2.2.

In the description below, we assume a reader is familiar with basics of FCA; see Carpineto and Romano (2004), Ganter and Wille (1999) for reference if needed. We only briefly review our notation used in the rest of the paper. Input object-attribute data (formal context) are denoted by the triplet $\langle X, Y, I\rangle$, with assumed finite nonempty sets of objects $X = \{0, 1, \ldots, m\}$ and attributes $Y = \{0, 1, \ldots, n\}$, and $I \subseteq X \times Y$ being an incidence relation with $\langle x, y\rangle \in I$ saying that object $x \in X$ has attribute $y \in Y$. Concept-forming operators defined on I as usual (Ganter and Wille 1999) are denoted by $^{\uparrow_I} : 2^X \mapsto 2^Y$ and $^{\downarrow_I} : 2^Y \mapsto 2^X$. $\mathcal{B}(X, Y, I) = \{\langle A, B\rangle \in 2^X \times 2^Y \mid A = B^{\downarrow_I}, B = A^{\uparrow_I}\}$ denotes the set of all formal concepts in I and the partial order relation on $\mathcal{B}(X, Y, I)$, forming together a concept lattice, is denoted by $\leq$ and defined also as usual: $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ if $A_1 \subseteq A_2$ (iff $B_2 \subseteq B_1$).

2.1. New and modified concepts

In this section we describe how to compute new and modified formal concepts when introducing new objects to input data (i.e. updating the data). Let us suppose we are introducing to input data $\langle X, Y, I\rangle$ (finite nonempty set of) new objects $X_N = \{m+1, \ldots, m_U\}$, not present in X ($X_N \cap X = \emptyset$) and sharing (finite nonempty set of) attributes $Y_N = \{i, \ldots, k\}$. We do not assume any overlap of Y_N and Y (Y_N can contain new attributes not present in Y) but in the usual scenario we have $Y_N \subseteq Y$. Denote the incidence relation between X_N and Y_N by $N \subseteq X_N \times Y_N$ and the new/updated input data with new objects added by the triplet $\langle X_U, Y_U, I_U\rangle$. Here, $X_U = X \cup X_N = \{0, \ldots, m_U\}$, $Y_U = Y \cup Y_N = \{0, \ldots, n_U\}$, $n_U = k$ if $k > n$ and $n_U = n$ otherwise, and $I_U \subseteq X_U \times Y_U$ such that $I_U \cap (X \times Y) = I$, $I_U \cap (X_N \times Y_N) = N$ and $I_U \cap (X \times (Y_N \setminus Y)) = I_U \cap (X_N \times (Y \setminus Y_N)) = \emptyset$. Hence I_U is the union of I and N both extended to X_U and Y_U.

First, observe that attributes of Y_N which are not shared by any of objects X_N cannot participate in any new nor modified formal concept of $\mathcal{B}(X_U, Y_U, I_U)$, with the exception of the formal concept $\langle Y_U^{\downarrow_{I_U}}, Y_U\rangle$. Hence we will assume in the following, without loss of generality, that all attributes of Y_N are shared by at least one object from X_N. For an algorithm for computing new and modified formal concepts only it is then sufficient to process attributes Y_N only.

Now, for any $B_U = B^{\downarrow_{I_U}\uparrow_{I_U}} \subseteq Y_N$, if $B_U = B = B^{\downarrow_I\uparrow_I}$ then formal concept $\langle A_U, B_U\rangle \in \mathcal{B}(X_U, Y_U, I_U)$ with the same intent $B_U = B$ as formal concept $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ is a modified formal concept enlarging the extent of $\langle A, B\rangle$ by objects $A_U \setminus A \subseteq X_N$ if (and only if) $A_U \supset A$ (otherwise no objects from X_N share the attributes $B_U = B$ and $\langle A_U, B_U\rangle = \langle A, B\rangle$ is an old formal concept). Otherwise, if $B_U \subset B = B^{\downarrow_I\uparrow_I}$ (since the $\supset$ inclusion cannot happen since $X_U \supset X$) then the formal concept $\langle A_U, B_U\rangle \in \mathcal{B}(X_U, Y_U, I_U)$ is a new formal concept and the formal concept $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ is called its generator (Godin, Missaoui, and Alaoui 1995; Valtchev and Missaoui 2001). In both cases, $A_U \cap X = A$.

We use the above presented ideas to modify the FCbO algorithm described in Outrata and Vychodil (2012) as recursive procedure FastGenerateFrom. The procedure computes and lists the set $\mathcal{B}(X, Y, I)$ of all formal concepts of input data $\langle X, Y, I\rangle$. We modify the procedure to compute and list the new and modified formal concepts of $\langle X_U, Y_U, I_U\rangle$ only when adding new objects X_N described by attributes Y_N to $\langle X, Y, I\rangle$. The modified procedure UpdateFastGenerateFrom is depicted in Algorithm 1. The original procedure FastGenerateFrom is thoroughly described in Outrata and Vychodil (2012), so we describe only the modifications introduced in UpdateFastGenerateFrom.

The procedure accepts as its arguments an initial formal concept $\langle A, B\rangle$ of $\langle X_U, Y_U, I_U\rangle$, the first attribute $y \in Y_N$ to be processed and a set $\{N_y \subseteq Y_U \mid y \in Y_U\}$ of subsets of attributes Y_U; see Section 2.2 or Outrata and Vychodil (2012) for the meaning of the set. The procedure further uses local variables *queue* as a temporary storage for computed formal concepts and M_y ($y \in Y_U$) as sets of attributes which are used in place of N_y for further invocations of the procedure. After invocation, the procedure recursively descends through the space of new and modified formal concepts of $\langle X_U, Y_U, I_U\rangle$ resulted by adding new objects X_N described by attributes Y_N to $\langle X, Y, I\rangle$.

When invoked, UpdateFastGenerateFrom first checks if $(A \cap X)^{\uparrow_{I_U}}$ does not equal B (line 1), in which case $\langle A, B\rangle$ is a new formal concept (see above) which is listed as such. If, otherwise, $(A \cap X)^{\uparrow_{I_U}}$ equals B and if further $(A \cap X) \subset A$ (line 4), $\langle A, B\rangle$ is a modified

Algorithm 1: Procedure UpdateFastGenerateFrom($\langle A, B\rangle, y, \{N_y \mid y \in Y_U\}$), cf. Outrata and Vychodil (2012)

Input : formal concept $\langle A, B\rangle$ of $\langle X_U, Y_U, I_U\rangle$, attribute $y \in Y_N$ (or a number $\geq n_U$) and set $\{N_y \subseteq Y_U \mid y \in Y_U\}$ of subsets of attributes Y_U
Output: lists all new and modified formal concepts of $\langle X_U, Y_U, I_U\rangle$

```
1  if (A ∩ X)^{↑I_U} ≠ B then
2  |  list ⟨A, B⟩ as new (e.g. print it on screen or store it);
3  else
4  |  if (A ∩ X) ⊂ A then
5  |  |  list ⟨A, B⟩ as modified (e.g. print it on screen or store it);
6  |  else
7  |  |  return
8  |  end
9  end
10 if B = Y_U or y > n_U then
11 |  return
12 end
13 for j from n_U downto y do
14 |  set M_j to N_j;
15 |  if j ∉ B and j ∈ Y_N and N_j ∩ Y_{U,j} ⊆ B ∩ Y_{U,j} then
16 |  |  set C to A ∩ {j}^{↓I_U};
17 |  |  set D to C^{↑I_U};
18 |  |  if B ∩ Y_{U,j} = D ∩ Y_{U,j} then
19 |  |  |  put ⟨⟨C, D⟩, j + 1⟩ to queue;
20 |  |  else
21 |  |  |  set M_j to D;
22 |  |  end
23 |  end
24 end
25 while get ⟨⟨C, D⟩, j⟩ from queue do
26 |  UpdateFastGenerateFrom((⟨C, D⟩, j, {M_y | y ∈ Y_U}));
27 end
28 return
```

formal concept and it is as such listed. If $(A \cap X) = A$, $\langle A, B\rangle$ is an old formal concept and the procedure returns. Afterwards the procedure behaves exactly as the original procedure FastGenerateFrom (see Outrata and Vychodil 2012 for full description) with two exceptions. First, it goes through attributes j from n_U down to y (line 13) instead of from y up to n_U as in the original procedure described in Outrata and Vychodil (2012). The reversed order is not important here and it only changes the order in which formal concepts are computed and listed (for listing namely to the lectic order of Ganter's NextClosure algorithm (Ganter 1984); see Outrata and Vychodil 2012). However, the order will become important in the lattice order relation computation part described in Section 2.2. Second, the procedure goes through attributes $j \in Y_N$ only (line 15) as clarified above. Let us only recall that the set $Y_{U,j} \subseteq Y_U$ is defined by

$$Y_{U,j} = \{y \in Y_U \mid y < j\}.$$

In order to list all new and modified formal concepts of $\langle X_U, Y_U, I_U\rangle$, each of them exactly once, we invoke UpdateFastGenerateFrom with $\langle \emptyset^{\downarrow I_U}, \emptyset^{\downarrow I_U \uparrow I_U}\rangle$, y being the first attribute in Y_N and $\{N_y = \emptyset \mid y \in Y_U\}$ as its initial arguments. The correctness of Algorithm 1 follows from the correctness of the FCbO algorithm (Outrata and Vychodil

2012) with the reversed order of processing attributes (also discussed in that paper) and the above description of its modification.

Example 1: We illustrate Algorithm 1 on the following example. Consider an input data given in a usual way (rows correspond to objects, columns to attributes and table entries indicate whether an object has an attribute) by the upper part, rows 0 to 2, of the table depicted below (left). The data induces 8 formal concepts C_1 to C_8 depicted on the right.

I	0	1	2	3	4	5
0	×		×	×		
1		×		×	×	×
2	×	×				×
3		×		×		×
4	×		×	×		×

$$C_1 = \langle \{0,1,2\}, \emptyset \rangle, \qquad C_5 = \langle \{0,2\}, \{0\} \rangle,$$
$$C_2 = \langle \{0,1\}, \{3\} \rangle. \qquad C_6 = \langle \{0\}, \{0,2,3\} \rangle,$$
$$C_3 = \langle \{1,2\}, \{1,5\} \rangle, \qquad C_7 = \langle \{2\}, \{0,1,5\} \rangle,$$
$$C_4 = \langle \{1\}, \{1,3,4,5\} \rangle, \qquad C_8 = \langle \emptyset, \{0,1,2,3,4,5\} \rangle,$$

$$C_1^* = \langle \{0,1,2,3,4\}, \emptyset \rangle, \qquad C_{11} = \langle \{1,3\}, \{1,3,5\} \rangle,$$
$$C_9 = \langle \{1,2,3,4\}, \{5\} \rangle, \qquad C_5^* = \langle \{0,2,4\}, \{0\} \rangle,$$
$$C_2^* = \langle \{0,1,3,4\}, \{3\} \rangle, \qquad C_{12} = \langle \{2,4\}, \{0,5\} \rangle,$$
$$C_{10} = \langle \{1,3,4\}, \{3,5\} \rangle, \qquad C_6^* = \langle \{0,4\}, \{0,2,3\} \rangle,$$
$$C_3^* = \langle \{1,2,3\}, \{1,5\} \rangle, \qquad C_{13} = \langle \{4\}, \{0,2,3,5\} \rangle.$$

We introduce to the data two new objects represented by rows 3 and 4 of the lower part of the table. The introduction results in induction of five new formal concepts C_9 to C_{13} and five modified formal concepts C^* depicted below the old formal concepts. The concepts are listed down-right in order in which they are listed by procedure UpdateFastGenerateFrom.

Remark 1: Algorithm 1 can be easily used to list all formal concepts of a input data $\langle X_C, Y_C, I_C \rangle$ in an incremental way, i.e. processing the data object by object, as incremental algorithms for computing concept lattice do. Namely, we invoke procedure UpdateFastGenerateFrom with initial arguments repeatedly for each object $i \in X_C = \{0, \ldots, n_C\}$, setting $\langle X, Y, I \rangle := \langle \{0, \ldots, i-1\}, \bigcup_{j=0}^{i-1} \{j\}^{\uparrow I_C}, I_C \cap (\{0, \ldots, i-1\} \times \bigcup_{j=0}^{i-1} \{j\}^{\uparrow I_C}) \rangle$, $X_N := \{i\}$, $Y_N := \{i\}^{\uparrow I_C}$ and $N := X_N \times Y_N$ for each invocation, and filter out listed modified formal concepts listing new formal concepts only. Such a computation of all formal concepts of data is, however, quite inefficient due to many repeated computations of formal concepts listed as modified, see the performance evaluation in Section 3. The concepts are filtered out from listing but, however, necessary to compute in order to decide whether a concept is new or modified. Incremental algorithms iterate over the (so far) incrementally computed and stored concept lattice to decide and compute new formal concepts which is more efficient. A trade-off, as discussed in the introduction Section 1, is that the concept lattice is required for the computation and must be stored by the incremental algorithms. Algorithm 1, on the other hand, requires input data only and does not need to store anything.

2.2. Lattice order relation

In this section, we describe how to determine the changes in the concept lattice order relation which need to be done with the addition of new formal concepts to the lattice.

However, in order to determine the changes in concept lattice order relation by extending the modified FCbO algorithm introduced in the previous Section 2.1, we first need to determine the concept lattice order relation alone. This is due to the fact that the FCbO algorithm, as well as the modification, compute the set of formal concepts of input data only. So, in the following, we describe an extension of the FCbO algorithm (Outrata and Vychodil 2012) to determine the concept lattice order relation $\leq$ on the set of all formal concepts $\mathcal{B}(X, Y, I)$ computed by FCbO. As a result, we create an algorithm for computing concept lattice of $\langle X, Y, I\rangle$. In fact, we will not determine the whole order relation but rather its cover relation. Recall that the cover relation on $\mathcal{B}(X, Y, I)$ for $\leq$ is defined such that a formal concept $\langle A_2, B_2\rangle$ covers a formal concept $\langle A_1, B_1\rangle$ if $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ and there is no formal concept $\langle A_3, B_3\rangle$ distinct from both $\langle A_1, B_1\rangle$ and $\langle A_2, B_2\rangle$ such that $\langle A_1, B_1\rangle \leq \langle A_3, B_3\rangle \leq \langle A_2, B_2\rangle$ (i.e. the cover relation is the transitive reduct of $\leq$). And since we do not store and use the computed concept lattice in our final algorithm for updating the lattice, we will not store and use the computed formal concepts nor the concept lattice order cover relation in the extended FCbO algorithm as well. Besides listing the formal concepts, we will only list the pairs of formal concepts to be created in the relation.

We now describe how to extend the FCbO algorithm, as described in Outrata and Vychodil (2012), in order to determine the concept lattice order cover relation on the set of computed formal concepts, i.e. to compute the concept lattice. For a pseudocode of FCbO, we can use the pseudocode depicted in Algorithm 1 if we look apart from the modifications to FCbO introduced there (the check whether $\langle A, B\rangle$ is a new or modified formal concept between lines 1 and 9 and going through attributes from Y_N only at line 15). In that case, we obtain the original procedure FastGenerateFrom from Outrata and Vychodil (2012) with the reversed order of attribute processing. To follow, we will need a little knowledge of the FCbO algorithm now. The algorithm, see the pseudocode of procedure UpdateFastGenerateFrom, for all attributes $j \in Y$ such that $y \leq j \leq n$ which are not present in B (lines 13 and 15), computes a formal concept $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle$ (lines 16 and 17) from a formal concept $\langle A, B\rangle$ if j passes the (FCbO) test $N_j \cap Y_j \subseteq B \cap Y_j$ (line 15). Attributes N_j are just "saved" attributes D from some previous recursive invocation of the procedure (see lines 21 and 26). The concept is further processed (and listed) only if it passes also the canonicity test $B \cap Y_j = D \cap Y_j$ (line 18). Note that the first test fails if $(N_j \setminus B) \cap Y_j \neq \emptyset$ and the second test fails if $(D \setminus B) \cap Y_j \neq \emptyset$.

Next, certainly $\langle C, D\rangle \leq \langle A, B\rangle$. However, the formal concepts need not fulfil the definition of cover relation. To verify it, we use a test known from Lindig's NextNeighbor algorithm (Lindig 2000). The test is based on the fact that a formal concept $\langle A, B\rangle \neq \langle Y^{\downarrow}, Y\rangle$ covers a formal concept $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle, j \notin B$ iff $(A \cap \{k\}^{\downarrow})^{\uparrow} = D$ for all attributes $k \in D \setminus B$ (cf. Theorem 1 in Lindig 2000). Now, in the test we can conveniently utilize the above FCbO and canonicity tests of $\langle C, D\rangle$! If either of the tests fails due to some $k \in (N_j \setminus B) \cap Y_j$, resp. $k \in (D \setminus B) \cap Y_j$, then $\langle C, D\rangle$ does not cover $\langle A, B\rangle$ if $(A \cap \{k\}^{\downarrow})^{\uparrow} \neq D$. However, if otherwise, $(A \cap \{k\}^{\downarrow})^{\uparrow} = D$ for all such k and also $(A \cap \{l\}^{\downarrow})^{\uparrow} = D$ for all $l \in N_j \setminus B$, resp. $l \in D \setminus B, l > j$, then $\langle C, D\rangle$ can be listed as covering $\langle A, B\rangle$ when computed

for some of those k instead of for j. Hence, in any case, $\langle C, D\rangle$ which fails the FCbO or canonicity tests is not listed as covering $\langle A, B\rangle$. The sufficient and necessary conditions are to test $(A \cap \{k\}^{\downarrow})^{\uparrow}$, resp. $(A \cap \{l\}^{\downarrow})^{\uparrow}$, for all the attributes $l > j$ before and $k < j$ after testing $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle, j \notin B$. That explains the reversed order of processing attributes j from n down to y instead of from y up to n as in the original FCbO procedure in Outrata and Vychodil (2012). If $\langle C, D\rangle$ passes both the FCbO and canonicity tests, i.e. $(N_j \setminus B) \cap Y_j = D \setminus B) \cap Y_j = \emptyset$, and $(A \cap \{l\}^{\downarrow})^{\uparrow} = D$ for all $l \in N_j \setminus B$, resp. $l \in D \setminus B$, $l > j$, $\langle C, D\rangle$ is listed as covering $\langle A, B\rangle$.

Finally, recall that formal concepts $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle$ are computed from $\langle A, B\rangle$ in each invocation of (Update)FastGenerateFrom for attributes $j \geq y$ only. In order to determine and list also formal concepts $\langle E, F\rangle = \langle A \cap \{i\}^{\downarrow}, (A \cap \{i\}^{\downarrow})^{\uparrow}\rangle$ as covering $\langle A, B\rangle$ for attributes $0 \leq i < y - 1$ which are not present in B (note that $0 \leq i = y - 1$ is always present in B since $\langle A, B\rangle$ was computed by adding $y - 1$ in the previous recursive stage of the procedure), we need to compute and test the concepts. And due to the above condition of testing $(A \cap \{k\}^{\downarrow})^{\uparrow}$ for all attributes $k < j$ after testing $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle$, we need to do it after processing formal concepts $\langle C, D\rangle$ above. Unfortunately, due to the (necessary) order in which formal concepts are computed by the FCbO with reversed order of processing of attributes, the concepts $\langle E, F\rangle$ will be again computed in some of the future stages of the algorithm after the computation of formal concepts $\langle C, D\rangle$. And, since formal concepts are not stored in FCbO nor in our extension, we have to compute the concepts $\langle E, F\rangle$ repeatedly. On the other hand, to verify that $\langle E, F\rangle$ and $\langle A, B\rangle$ fulfil the definition of cover relation we can test $\langle E, F\rangle$ within the Lindig's NextNeighbor algorithm test used above for testing the formal concepts $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle$ for attributes $j \geq y$.

The modified procedure LatticeFastGenerateFrom, which implements the above presented ideas to the original FCbO procedure FastGenerateFrom, is depicted in Algorithm 2. LatticeFastGenerateFrom extends FastGenerateFrom by determining the concept lattice order cover relation on the set of formal concepts computed by FastGenerateFrom, i.e. to compute the concept lattice. As with the procedure UpdateFastGenerateFrom in Section 2.1, we describe only the modifications introduced to FastGenerateFrom.

The modifications extend the original FCbO procedure by including the processing of attributes in the reversed order (lines 6 and 23), integrating the test of fulfilling the definition of cover relation by pairs of a formal concept computed from $\langle A, B\rangle$ and $\langle A, B\rangle$ and by recomputing and testing the concepts $\langle E, F\rangle = \langle A \cap \{i\}^{\downarrow}, (A \cap \{i\}^{\downarrow})^{\uparrow}\rangle, i \notin B$ for attributes $0 \leq i < y - 1$ (lines 23 to 33). The test of fulfilling the definition of cover relation is performed according to the description above, only in a slightly modified form using the *min* local variable (lines 5, 8, 14, 16 and 27, 28, 30) borrowed from the original description of Lindig's NextNeighbor algorithm in Lindig (2000). If a pair of formal concepts passes the test it is listed as to be created in the cover relation (lines 15 and 29). Note again that the reversed order of attribute processing only changes the order in which formal concepts are computed and listed (for listing to the lectic order of Ganter's NextClosure algorithm, see Outrata and Vychodil 2012). The other modifications do not change the original FCbO algorithm as well.

In order to output the concept lattice of $\langle X, Y, I\rangle$, that is to list all formal concepts of $\langle X, Y, I\rangle$, each of them exactly once, together with all pairs of formal concepts in the cover relation of concept lattice of $\langle X, Y, I\rangle$, each pair listed exactly once, we invoke

Algorithm 2: Procedure LatticeFastGenerateFrom($\langle A, B\rangle, y, \{N_y \mid y \in Y\}$), cf. Outrata and Vychodil (2012)

Input : formal concept $\langle A, B\rangle$ of $\langle X, Y, I\rangle$, attribute $y \in Y$ (or a number $\geq n$) and set $\{N_y \subseteq Y \mid y \in Y\}$ of subsets of attributes Y
Output: lists all formal concepts of $\langle X, Y, I\rangle$ and all pairs of formal concepts in the cover relation of concept lattice of $\langle X, Y, I\rangle$

```
list ⟨A, B⟩ (e.g. print it on screen or store it);
if B = Y or y > n then
    return
end
set min to Y;
for j from n downto y do
    set M_j to N_j;
    set min to min \ {j};
    if j ∉ B and N_j ∩ Y_j ⊆ B ∩ Y_j then
        set C to A ∩ {j}↓;
        set D to C↑;
        if B ∩ Y_j = D ∩ Y_j then
            put ⟨⟨C, D⟩, j + 1⟩ to queue;
            if B ∩ min = D ∩ min then
                list ⟨⟨C, D⟩, ⟨A, B⟩⟩ (e.g. print C and A, or D and B, on screen or store them);
                set min to min ∪ {j};
            end
        else
            set M_j to D;
        end
    end
end
for i from y − 2 downto 0 do
    if i ∉ B then
        set E to A ∩ {i}↓;
        set F to E↑;
        set min to min \ {i};
        if B ∩ min = F ∩ min then
            list ⟨⟨E, F⟩, ⟨A, B⟩⟩ (e.g. print E and A, or F and B, on screen or store them);
            set min to min ∪ {i};
        end
    end
end
while get ⟨⟨C, D⟩, j⟩ from queue do
    LatticeFastGenerateFrom (( ⟨C, D⟩, j, {M_y | y ∈ Y}) );
end
return
```

LatticeFastGenerateFrom with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$, $y = 0$ and $\{N_y = \emptyset \mid y \in Y\}$ as its initial arguments. The correctness of Algorithm 2 follows from the correctness of FCbO algorithm (Outrata and Vychodil 2012) with the reversed order of processing attributes and the above description of its extension.

2.2.1. Update of concept lattice

Now, the determination of changes in (the cover relation of) concept lattice of $\langle X, Y, I\rangle$ which need to be done in reaction to the introduction of new objects to input data is then a matter of proper merge of procedures UpdateFastGenerateFrom and LatticeFast GenerateFrom in Algorithms 1 and 2, respectively.

The merged procedure LatticeUpdateFastGenerateFrom is depicted in Algorithm 3. We needed to take care of two things in the merge. First, we want to list only (new) pairs of formal concepts to be created in the lattice order relation which contain a new formal

concept. Pairs not containing a new formal a concept do not constitute a change in the lattice order relation (those are old pairs). To verify this, we moved the test on whether a formal concept $\langle A, B\rangle$ is new (line 1 in UpdateFastGenerateFrom) from the beginning of the procedure to places before listing the pairs of a formal concept newly computed from $\langle A, B\rangle$ and $\langle A, B\rangle$ (lines 22 and 38). The moved test was adjusted for the newly computed formal concepts $\langle C, D\rangle$ and $\langle E, F\rangle$, respectively. In the case of $\langle C, D\rangle$, the test had to be moved already before the concept is put to the temporary storage (represented by local variable *queue*, line 19) and we had to test also whether the concept is modified or old (lines 17 and 18, note the new local variable *nnew*). Since we need to store and pass in the recursive call the information whether the concept is new or modified (new argument *new* and lines 19, 45 and 46) and use it in the listing of the concept at the beginning of the procedure (line 1) and in the listings of (new) pairs.

Second, only attributes $j \geq y$ from Y_N are processed in UpdateFastGenerateFrom (line 15). In order to determine and list also formal concepts $\langle E, F\rangle = \langle A \cap \{j\}^{\downarrow I_U}, (A \cap \{j\}^{\downarrow I_U})^{\uparrow I_U}\rangle$ covering $\langle A, B\rangle$ for attributes $j \geq y$ not in Y_N, we have to process the attributes together with attributes $i < y - 1$ (line 23 in LatticeFastGenerateFrom), only before them. Thus, the extended loop at line 32 in LatticeUpdateFastGenerateFrom.

In order to output the update of concept lattice of $\langle X_U, Y_U, I_U\rangle$, that is to list all new and modified formal concepts of $\langle X_U, Y_U, I_U\rangle$, each of them exactly once, together with all new pairs of formal concepts (i.e. containing a new formal concept) in the cover relation of concept lattice of $\langle X_U, Y_U, I_U\rangle$, each pair listed exactly once, we invoke LatticeUpdateFastGenerateFrom with $\langle \emptyset^{\downarrow I_U}, \emptyset^{\downarrow I_U \uparrow I_U}\rangle$, *new* = *true/false* depending on whether $\langle \emptyset^{\downarrow I_U}, \emptyset^{\downarrow I_U \uparrow I_U}\rangle$ is new or modified, y being the first attribute in Y_N and $\{N_y = \emptyset \,|\, y \in Y_U\}$ as its initial arguments. Whether $\langle \emptyset^{\downarrow I_U}, \emptyset^{\downarrow I_U \uparrow I_U}\rangle$ is new or modified is checked before the invocation of LatticeUpdateFastGenerateFrom using the test at line 17 of the procedure (using $\langle C, D\rangle = \langle \emptyset^{\downarrow I_U}, \emptyset^{\downarrow I_U \uparrow I_U}\rangle$). The correctness of Algorithm 3 follows from the correctness of Algorithm 1 and Algorithm 2 and the above description.

Remark 2: Note that we handle only creation of new pairs of formal concepts in the cover relation in our concept lattice update algorithm. The changes in the relation triggered as a reaction to the introduction of new objects to input data include, however, also deletion of pairs of formal concepts. Those are the pairs $\langle\langle G, H\rangle, \langle I, J\rangle\rangle$ of modified or old formal concepts only (the pairs would be represented by "transitive edges" in the Hasse diagram of concept lattice). The pairs shall be deleted in the relation and "replaced" by sequences of (new) pairs containing new formal concepts $\langle G, H\rangle \leq \langle K, L\rangle \leq \langle I, J\rangle$ (and the "transitive edges" be deleted). We cannot handle those deletions in our algorithm since it would require storing parts of the concept lattice (order relation) and that is something we do not (want to) do in our algorithm. The deletions shall be done by an application which uses (and stores the parts of) the concept lattice cover relation.

Example 2: We illustrate Algorithms 2 and 3 for the input data from Example 1. Below (top left) there is depicted concept lattice consisting of the 8 formal concepts C_1 to C_8, with the concepts and pairs of concepts in cover relation of the lattice, below the lattice, listed in the order in which they are listed by procedure LatticeFastGenerateFrom.

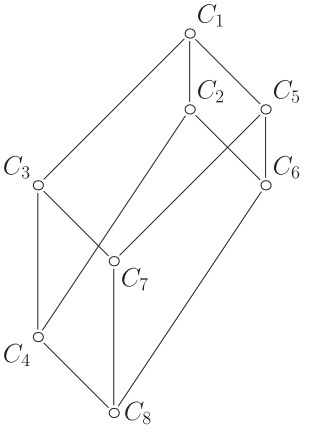
Next to the lattice, there is then the concept lattice and below the listing of the 8 formal concepts a listing of the five new formal concepts C_9 to C_{13} and new pairs in the cover relation after the introduction of the two new objects (rows 3 and 4 in the table in

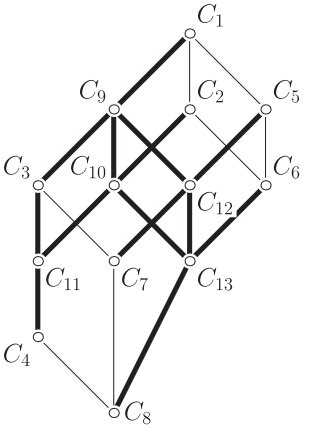
Algorithm 3: Procedure LatticeUpdateFastGenerateFrom($\langle A, B\rangle, new, y, \{N_y \mid y \in Y\}$), cf. Outrata and Vychodil (2012)

Input : formal concept $\langle A, B\rangle$ of $\langle X_U, Y_U, I_U\rangle$, $new = true/false$ depending on whether $\langle A, B\rangle$ is new or modified, attribute $y \in Y_N$ (or a number $\geq n_U$) and set $\{N_y \subseteq Y_U \mid y \in Y_U\}$ of subsets of attributes Y_U
Output: lists all new and modified formal concepts of $\langle X_U, Y_U, I_U\rangle$ and all new pairs of formal concepts in the cover relation of concept lattice of $\langle X_U, Y_U, I_U\rangle$

if $new = true$ **then**
  list $\langle A, B\rangle$ as new (e.g. print it on screen or store it);
else
  list $\langle A, B\rangle$ as modified (e.g. print it on screen or store it);
end
if $B = Y_U$ **or** $y > n_U$ **then**
  return
end
set min **to** Y_U;
for j **from** n_U **downto** y **do**
  set M_j **to** N_j;
  set min **to** $min \setminus \{j\}$;
  if $j \notin B$ **and** $j \in Y_N$ **and** $N_j \cap Y_{U,j} \subseteq B \cap Y_{U,j}$ **then**
    set C **to** $A \cap \{j\}^{\downarrow I_U}$;
    set D **to** $C^{\uparrow I_U}$;
    if $B \cap Y_{U,j} = D \cap Y_{U,j}$ **then**
      $nnew = (\,(C \cap X)^{\uparrow I_U} \neq D)$;
      if $nnew = true$ **or** $(C \cap X) \subset C$ **then**
        put $\langle\langle C, D\rangle, nnew, j + 1\rangle$ **to** *queue*;
      end
      if $B \cap min = D \cap min$ **then**
        if $new = true$ **or** $nnew = true$ **then**
          list $\langle\langle C, D\rangle, \langle A, B\rangle\rangle$ (e.g. print C and A, or D and B, on screen or store them);
        end
        set min **to** $min \cup \{j\}$;
      end
    else
      set M_j **to** D;
    end
  end
end
for i **from** n_U **downto** y **not in** Y_N **and from** $y - 2$ **downto** 0 **do**
  if $i \notin B$ **then**
    set E **to** $A \cap \{i\}^{\downarrow I_U}$;
    set F **to** $E^{\uparrow I_U}$;
    set min **to** $min \setminus \{i\}$;
    if $B \cap min = F \cap min$ **then**
      if $new = true$ **or** $(E \cap X)^{\uparrow I_U} \neq F$ **then**
        list $\langle\langle E, F\rangle, \langle A, B\rangle\rangle$ (e.g. print E and A, or F and B, on screen or store them);
      end
      set min **to** $min \cup \{i\}$;
    end
  end
end
while get $\langle\langle C, D\rangle, nnew, j\rangle$ **from** *queue* **do**
  LatticeFastGenerateFrom $(\langle C, D\rangle, nnew, j, \{M_y \mid y \in Y_U\})$;
end
return

Example 1) into the data. The new pairs are depicted in bold face in the lattice and the listing of concepts and pairs of concepts are again listed down-right in order in which they are listed by a procedure LatticeUpdateFastGenerateFrom. The pairs $\langle C_4, C_2\rangle$, $\langle C_3, C_1\rangle$, $\langle C_4, C_3\rangle$, $\langle C_8, C_6\rangle$ and $\langle C_7, C_5\rangle$ which do not exist any more in the cover relation after the update were deleted in the depiction, as the using application would do as explained above in Remark 2.





C_1,
$C_2 : \langle C_2, C_1\rangle, \langle C_4, C_2\rangle, \langle C_6, C_2\rangle,$
$C_3 : \langle C_3, C_1\rangle, \langle C_7, C_3\rangle,$
$C_4 : \langle C_4, C_3\rangle, \langle C_8, C_4\rangle,$
$C_5 : \langle C_5, C_1\rangle,$
$C_6 : \langle C_6, C_5\rangle, \langle C_8, C_6\rangle,$
$C_7 : \langle C_7, C_5\rangle,$
$C_8 : \langle C_8, C_7\rangle.$

$C_9 : \langle C_9, C_1^*\rangle, \langle C_{10}, C_9\rangle, \langle C_3^*, C_9\rangle, \langle C_{12}, C_9\rangle,$
$C_{10} : \langle C_{10}, C_2^*\rangle, \langle C_{11}, C_{10}\rangle, \langle C_{13}, C_{10}\rangle,$
$C_{11} : \langle C_{11}, C_3^*\rangle, \langle C_4, C_{11}\rangle,$
$C_{12} : \langle C_{12}, C_5^*\rangle, \langle C_{13}, C_{12}\rangle, \langle C_7, C_{12}\rangle,$
$C_{13} : \langle C_{13}, C_6^*\rangle, \langle C_8, C_{13}\rangle.$

3. Experimental evaluation

The asymptotic worst-case time complexity of Algorithm 1 remains the same as for the FCbO (and CbO) algorithm, $O(|\mathcal{B}(X, Y, I)| \cdot |Y|^2 \cdot |X|)$, because when "introducing" all objects to empty data it actually performs FCbO. The complexity of Algorithms 2 and 3 is in the worst case $|Y|$ factor higher but on average they perform a constant factor slower than FCbO.

We have run several experiments to evaluate the performance of the presented algorithms. For the listing of all formal concepts and computing concept lattice of input data, we also compared the algorithms with the AddIntent incremental algorithm (van der Merwe, Obiedkov, and Kourie 2004) for computing concept lattice. In the comparison, we also run Algorithms 1 and 3 in the way processing input data incrementally, as mentioned in Remark 1. This was done for the sake of presenting a fairer comparison with the

true incremental algorithm although such a usage of our algorithms is not efficient (as mentioned in the remark).

In the experiments, we used our implementations of the presented algorithms in ANSI C, which are modifications of our (performance efficient) FCbO algorithm implementation used for performance evaluation in Outrata and Vychodil (2012). The implementation of the AddIntent algorithm was borrowed from one of the authors of van der Merwe, Obiedkov, and Kourie (2004).

The experiments were run on otherwise idle 32-bit i386 hardware (2 × 2 Intel Xeon 3.2 GHz, 3 GB RAM). We were interested in the performance of all algorithms measured by running time and memory usage. We have run the algorithms on both synthetic randomly generated data with various size and percentage of ×s in the table (fill ratio, with normal distribution) as well as with six selected real benchmark data-sets from the UCI Machine Learning Repository (Bache and Lichman 2013) (mushroom, anonymous-msweb, T10I4D100K and vote) and the Coron Data Mining Platform (Kaytoue et al. 2010) (c20d10k and t25i10d10k).

In the first set of experiments, we evaluated Algorithm 1 for updating the set of all formal concepts (computing new and modified concepts) when introducing new objects to input data. The performance for introducing a single new object (with randomly generated attributes) to random data with 100,000 objects is illustrated in Figure 1. The graphs show the dependency of time required to compute the update on the number of attributes, of data with fixed table fill ratio 5% (the graph on the left) and on the fill ratio of tables with fixed 150 attributes (the graph in the middle). Figure 1 (right) then illustrates the performance for introducing a growing number of new objects to random data with the number of objects being 100,000 minus the number of the new objects, 200 attributes and 5% table fill ratio. Note that the number of new objects in the graph is in logarithmic scale (base 10). The illustration for the benchmark data-sets, of the performance of introducing a growing number of last objects of the data to the data without the objects, is presented in Figure 2. In the graphs, the solid line is for data with objects ordered as in the original data-set and the dashed line is for data with randomly ordered objects (the time is an average from three orderings).

We can see from the graphs showing the performance of updating the set of all formal concepts when introducing a single new object to the data (Figure 1, left and middle) that this operation is extremely fast compared to computing all formal concepts (the latter took 7423 seconds for 500 attributes and 363 seconds for fill ratio 10%!). The reason is of course computing only a bare fraction of new and modified formal concepts among all concepts. Introducing more new objects (Figures 1, right, and 2) is much faster too, up to a limit of the number of new objects depending on the total number of objects in data and then being close to the time of computing all formal concepts (compare for benchmark data-sets with the FCbO times in Table 1). Note also that running times for mushroom and c20d10k data-sets differ for original and random orderings of objects.

In the second set of experiments, we evaluated Algorithm 2 for determining the cover relation of concept lattice of input data. The time and memory performance for random data is illustrated in Figure 3. The graphs on the left show the dependency of required time (top) and memory (bottom) on the number of attributes, of data with 10,000 objects and fixed table fill ratio 5%. The graphs in the middle show the dependency on the fill ratio of tables with 1000 objects and fixed 100 attributes. Note that the memory usage in the graphs

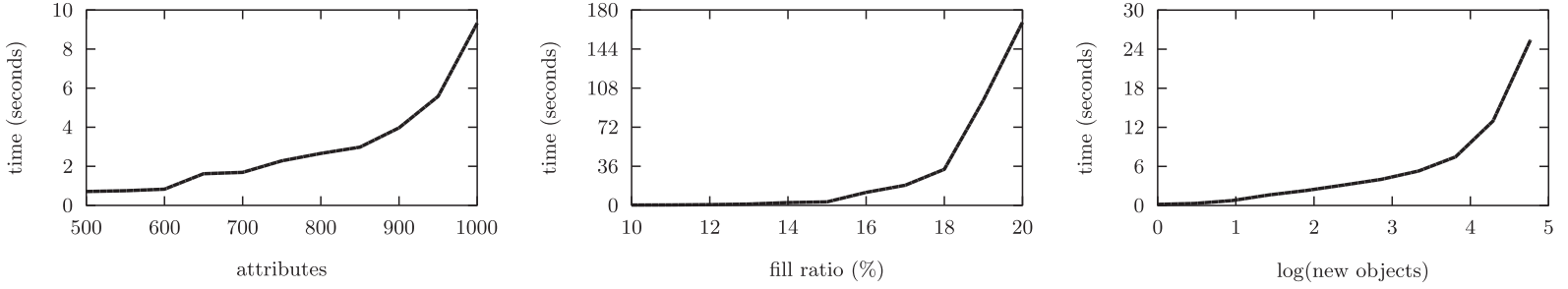


Figure 1. Running time of Algorithm 1 on random data dependent on number of attributes (on the left, introducing a single new object), on fill ratio (percentage of $\times$s, in the middle, introducing a single new object) and on number of new objects (on the right).

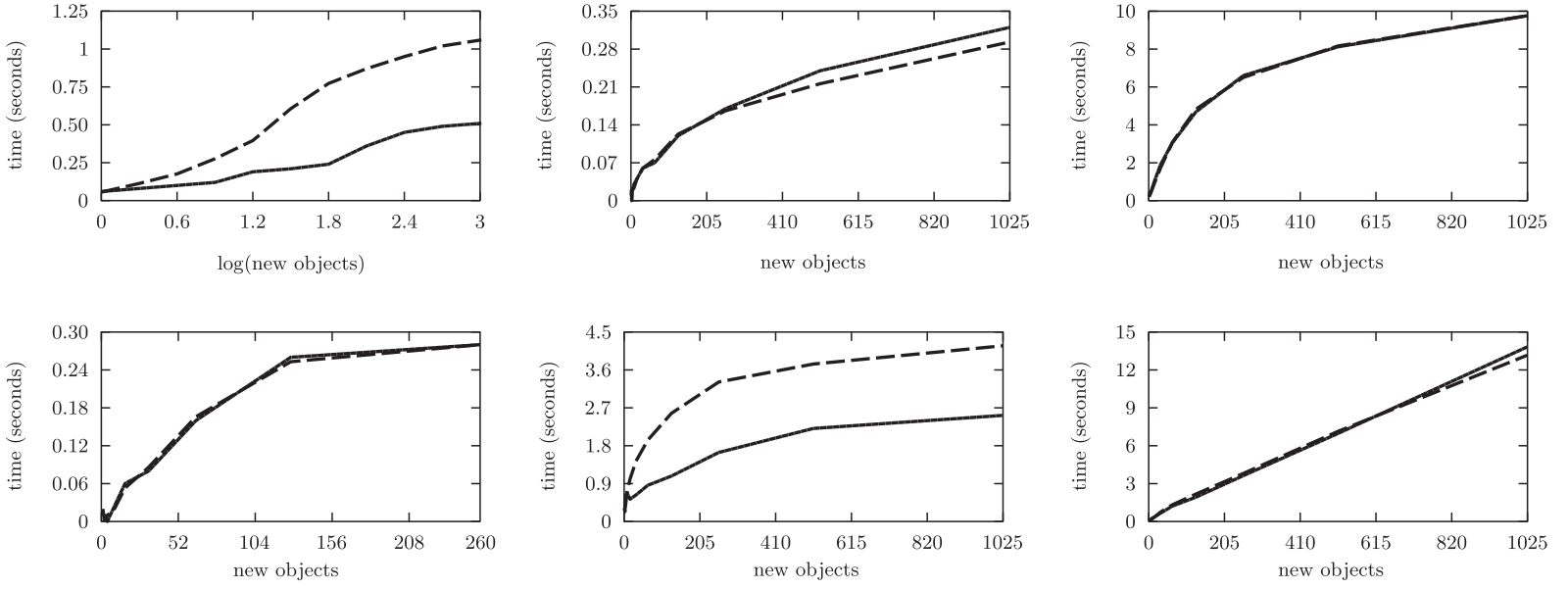


Figure 2. Running time of Algorithm 1 dependent on number of new (last) objects for mushroom (top left), anonymous-msweb (top middle), T10I4D100K (top right), vote (bottom left), c20d10k (bottom middle) and t25i10d10k data-sets (bottom right), solid line – orig. object ordering, dashed line – random object ordering.

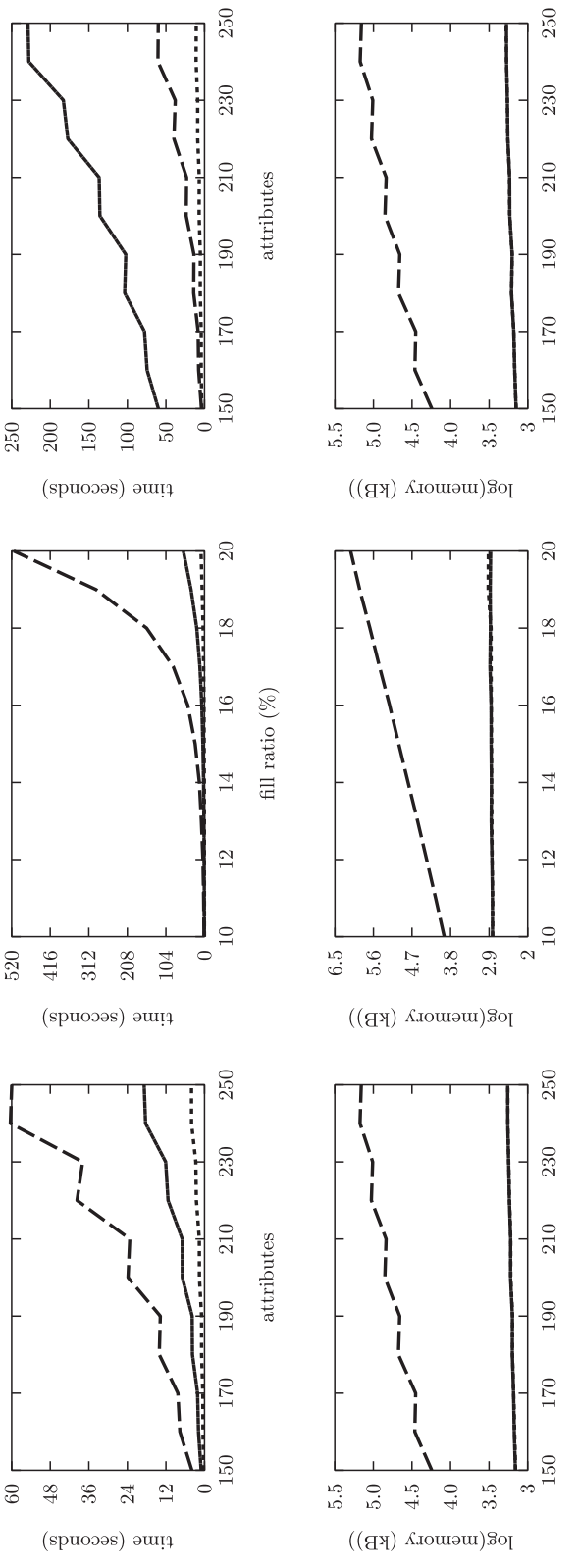


Figure 3. Running time (top) and memory usage (bottom) dependent on number of attributes (on the left and right) and on fill ratio (percentage of $\times$s, in the middle), solid line – Algorithm 2 (left, middle)/Algorithm 3 (right), dashed line – AddIntent, dotted line – FCbO (left, middle)/Algorithm 1 (right).

is in logarithmic scale (base 10). The graphs show also the comparison with AddIntent and the original FCbO algorithm; the solid line is for Algorithm 2, the dashed line is for AddIntent and the dotted line is for FCbO. The performance for the benchmark data-sets is presented in Table 1. Again, the times and memory usages are given for data with objects ordered as in the original data-set and for data with randomly ordered objects (the time and memory usage is an average from three orderings), and we also put information on size, fill ratio and the number of all formal concepts for each data-set.

From the graphs we can see that Algorithm 2 considerably outperforms AddIntent, in terms of time, on synthetic random data (in particular for higher fill ratio of data table), while for real benchmark data-sets, in the table, this must not always be the case (T10I4D100K, c20d10k, t25i10d10k, closely mushroom). This, however, heavily depends on the concept lattice structure (size of the cover relation) of the data-sets. We can also see from the comparison with the FCbO algorithm, as to the rest expected, that determining the cover relation takes considerably more time than computing just the set of the formal concepts. The comparison of performance in terms of memory usage also resulted as expected. Since Algorithm 2 does not store in memory the computed concept lattice (since the algorithm does not use it), and its memory usage is very low and almost constant (and almost the same as the memory usage of FCbO!) with both increasing number of attributes and data table fill ratio. Contrary to that, AddIntent stores the lattice in memory (since the algorithm needs it), and its memory usage increases very fast (exponentially) with the lattice size. See also that ordering of objects in the benchmark data-sets makes almost no difference in the performance of all algorithms (with the exception of the c20d10k data-set).

Finally, a comparison of performance of Algorithms 1 and 3 with AddIntent, of computing the set of all formal concepts or concept lattice when processing input data incrementally as mentioned in Remark 1, is presented in Figure 3 (right) and Table 2. The graphs show the time (top) and memory usage (bottom) performance for the data with 10,000 objects and fixed table fill ratio 5% from the preceding evaluation of Algorithm 2 (again the time and memory usage is an average from three random orderings of objects); the solid line is for Algorithm 3, the dashed line is for AddIntent and the dotted line is for Algorithm 3. In the table we also put, for our algorithms, for the sake of comparison, the numbers of formal concepts computed in total (for original ordering of objects in data-sets).

The results are somewhat interesting here. First, from both the graph of time performance for synthetic random data, Figure 3 (top right) and the table for benchmark data-sets, Table 2, we can see that AddIntent significantly outperforms Algorithm 3. This is not interesting and it was expected since, as mentioned by Remark 1, our algorithms are not designed and meant to be used as incremental algorithms (processing objects one by one). What is interesting, however, is that Algorithm 1, not determining the lattice cover relation like AddIntent but the set of formal concepts only, outperforms AddIntent for synthetic random data and for some benchmark data-sets (namely vote and t25i10d10k). The inspection of for what types of data this is the case is a subject for future research.

Table 1. Running time (in seconds) and memory usage (in kB) of determining the cover relation of concept lattice and numbers of formal concepts for selected data-sets.

Data-set	mushroom	anonymous-msweb	T10I4D100K	vote	c20d10k	t25i10d10k
Size	8124 × 119	32,711 × 296	100,000 × 1000	435 × 50	10,000 × 366	9976 × 1000
Fill ratio	19.167%	1.019%	1.010%	33.333%	5.181%	2.475%
#Concepts	238,710	129,009	2,347,376	264,035	375,378	3,853,929
Orig. object ordering						
Time						
Algorithm 2	9.750	2.840	400.170	2.030	72.480	583.240
AddIntent	9.880	6.590	352.350	7.810	23.640	477.240
FCbO	0.840	1.380	57.300	0.220	3.010	64.680
Memory						
Algorithm 2	3380	3928	19,980	688	4120	5072
AddIntent	140,008	22,844	348,380	66,080	252,876	532,916
FCbO	3352	3904	19,972	664	4088	5068
Random object ordering						
Time						
Algorithm 2	10.680	2.806	401.443	2.056	89.330	590.820
AddIntent	8.310	6.566	352.723	7.626	25.250	462.133
FCbO	0.920	1.393	57.703	0.223	3.683	65.550
Memory						
Algorithm 2	3384	3926	19,976	692	4121	5070
AddIntent	140,010	22,845	348,381	66,080	252,870	532,902
FCbO	3350	3901	19,974	664	4088	5068

Table 2. Running time (in seconds) and memory usage (in kB) of computing all formal concepts or concept lattice when processing input data incrementally and numbers of formal concepts computed in total for selected data-sets.

Data-set	mushroom	anonymous-msweb	T10I4D100K	vote	c20d10k	t25i10d10k
#Concepts in total	9,577,435	992,259	14,240,918	2,401,825	30,566,467	11,293,865
Orig. object ordering						
Time						
Algorithm 3	2233.450	1706.050	205,650.420	28.370	39,423.270	3889.760
AddIntent	9.880	6.590	352.350	7.810	23.640	477.240
Algorithm 1	129.250	38.190	2416.740	2.400	1119.700	83.870
Memory						
Algorithm 3	2588	3724	17,992	744	3108	3656
AddIntent	140,008	22,844	348,380	66,080	252,876	532916
Algorithm 1	2536	3728	17,988	728	3100	3504
Random object ordering						
Time						
Algorithm 3	5050.653	1831.500	205,603.293	33.190	43,810.816	4035.390
AddIntent	8.310	6.566	352.723	7.626	25.250	462.133
Algorithm 1	282.370	38.830	2415.783	2.780	1184.076	84.763
Memory						
Algorithm 3	2824	3610	17,988	744	3418	3694
AddIntent	140,010	22,845	348,381	66,080	252,870	532,902
Algorithm 1	2748	3548	17,988	724	3349	3594

4. Conclusions

We have introduced algorithms for updating the set of all formal concepts of object-attribute relational data when the data change (new objects or attributes are introduced or existing are deleted or altered) by computing only new and modified formal concepts and for determining the concept lattice order relation, i.e. for computing concept lattice of the data. Together the algorithms form an algorithm for updating concept lattice of object-attribute data from the data only, not requiring the possibly large concept lattice for the computation as the so-called incremental concept lattice algorithms do. The algorithms result as a modification or extension of the FCbO algorithm for computing the set of all formal concepts of data and the modifications can be equally applied to any other recent algorithms (PCbO, PFCbO) derived from Kuznetsov's CbO algorithm. The experimental evaluation of performance of the algorithms have shown that the update is performed significantly faster than recomputing all formal concepts or whole concept lattice over again and that the computation of concept lattice is performance comparable to existing incremental algorithms for computing concept lattice. However, unlike those algorithms our algorithms do not have memory requirements proportional to the size of computed concept lattice.

Future research will be focused on further experimental and real-world application evaluation of the algorithms and performance comparison with other incremental (update) algorithms for computing concept lattice.

Disclosure statement

No potential conflict of interest was reported by the author.

Funding

This work was supported by the European Social Fund and the state budget of the Czech Republic, under ESF Project No. CZ.1.07/2.3.00/20.0059.

Notes on contributor

Jan Outrata is an assistant professor at the Department of Computer Science, Faculty of Science, Palacký University Olomouc (Czech Republic). He obtained his MSc in Computer Science in 2003, and PhD in Mathematics in 2006, both from Palacký University Olomouc. His research interests include formal concept analysis and relational data analysis, classification and fuzzy relational systems. He has authored or co-authored over 30 papers in these areas in conference proceedings and journals including IEEE Trans. Fuzzy Systems, J. Computer and System Sciences, Int. J. General Systems, and Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. He is a member of the Steering Committee of the Int. Conf. on Concept Lattices and Their Applications (CLA).

References

Bache, K, and M Lichman. 2013. *UCI Machine Learning Repository*. Irvine, CA: School of Information and Computer Sciences. University of California. http://archive.ics.uci.edu/ml.

Carpineto, C., and G. Romano. 2004. *Concept Data Analysis. Theory and Applications*. Chichester: J. Wiley.
Ganter, B. 1984. Two Basic Algorithms in Concept Analysis. Technical Report FB4-Preprint No. 831. TH Darmstadt.
Ganter, B., and R. Wille. 1999. *Formal Concept Analysis. Mathematical Foundations*. Berlin: Springer.
Godin, R., R. Missaoui, and H. Alaoui. 1995. "Incremental Concept Formation Algorithms Based on Galois Lattices." *Computation Intelligence* 11: 246–267.
Kaytoue, M., F. Marcuola, A. Napoli, L. Szathmary, and J. Villerd. 2010. "The Coron System." In *Proceedings of ICFCA 2010 – Supplementary Proceedings*, 55–58. http://coron.loria.fr.
Kirchberg, M., E. Leonardi, Y. S. Tan, S. Link, R. K L Ko, and B. S. Lee. 2012. "Formal Concept Discovery in Semantic Web Data." In *Proceedings of ICFCA 2012, LNAI*. Leuven. Vol. 7278, 164–179. Berlin: Springer.
Krajca, P., J. Outrata, and V. Vychodil. 2010. "Advances in Algorithms Based on CbO." In *Proceedings of CLA 2010*. Sevilla. Vol. 2010, 325–337.
Krajca, P., J. Outrata, and V. Vychodil. 2010b. "Parallel Algorithm for Computing Fixpoints of Galois Connections." *Annals of Mathematics and Artificial Intelligence* 59 (2): 257–272.
Kuznetsov, S. O. 1989. "Interpretation on Graphs and Complexity Characteristics of a Search for Specific Patterns." *Automatic Documentation and Mathematical Linguistics* 24 (1): 37–45.
Kuznetsov, S. O. 1993. "A Fast Algorithm for Computing All Intersections of Objects in a Finite Semi-lattice." *Automatic Documentation and Mathematical Linguistics* 27 (5): 11–21.
Kuznetsov, S. O. 1999. "Learning of Simple Conceptual Graphs from Positive and Negative Examples." In *Proceedings of PKDD 1999*. 384–391. Heidelberg: Springer.
Lindig, C. 2000. "Fast Concept Analysis." In *Working with Conceptual Structures – Contributions to ICCS 2000*. Darmstadt. Vol. 2000, 152–161.
Norris, E. M. 1978. "An Algorithm for Computing the Maximal Rectangles in a Binary Relation." *Revue Roumaine de Mathématiques Pures et Appliquées* 23 (2): 243–250.
Outrata, J., and V. Vychodil. 2012. "Fast Algorithm for Computing Fixpoints of Galois Connections Induced by Object-attribute Relational Data." *Information Sciences* 185 (1): 114–127.
Valtchev, P., M. R. Hacene, and R. Missaoui. 2003. "A Generic Scheme for the Design of Efficient On-line Algorithms for Lattices." In *Proceedings of ICCS 2003, LNCS*. Dresden. Vol. 2746, 282–295. Berlin: Springer.
Valtchev, P., and R. Missaoui. 2001. "Building Concept (Galois) Lattices from Parts: Generalizing the Incremental Methods." In *Proceedings of ICCS 2001, LNAI*. Stanford. Vol. 2120, 290–303. Berlin: Springer.
Valtchev, P., R. Missaoui, and R. Godin. 2008. "A Framework for Incremental Generation of Closed Itemsets." *Discrete Applied Mathematics* 156 (6): 924–949.
van der Merwe, D., S. A. Obiedkov, and D. G. Kourie. 2004. "AddIntent: A New Incremental Algorithm for Constructing Concept Lattices." In *Proceedings of ICFCA 2004, LNAI*. Sydney. Vol. 2961, 205–206. Berlin: Springer.
Wille, R. 1982. "Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts." *Ordered Sets* 83: 445–470.