A lattice-free concept lattice update algorithm based on $*$CbO[⋆]

Jan Outrata

Dept. Computer Science, Palacky University, Olomouc
17. listopadu 12, CZ-77146 Olomouc, Czech Republic
jan.outrata@upol.cz

Abstract. Updating a concept lattice when introducing new objects to input data can be done by any of the so-called incremental algorithms for computing concept lattice of the data. The algorithms use and update the lattice while introducing new objects one by one. The present concept lattice of input data without the new objects is thus required before the update. 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 CbO algorithm for computing the set of all formal concepts, or its modifications like FCbO, PCbO or PFCbO, to compute new and modified formal concepts only and the changes of the lattice order relation when input data changes. We describe the algorithm and present an experimental evaluation of its performance and a comparison with AddIntent incremental algorithm for computing concept lattice.

1 Introduction

In applications of Formal Concept Analysis (FCA) [3, 16] 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 and to compute the new, changed, concept lattice we would like to compute the changes only and "update" the present lattice instead of recomputing the whole lattice from new, changed, input data.

The particular change of object-attribute relational data processed in FCA, namely the introduction of new objects (described by particular values of attributes), can be handled by the so-called incremental algorithms for computing concept lattice, Norris's [13], Object Intersections [2] or, more recent, AddIntent [12], for instance, or the incremental lattice update algorithms presented in [2]. The algorithms build/update the lattice incrementally by adding objects of input data, one by one, to present concept lattice computed so far from already processed objects, starting from the first object and empty concept lattice.

⋆ Support by the ESF project No. CZ.1.07/2.3.00/20.0059, the project co-financed by the European Social Fund and the state budget of the Czech Republic, is acknowledged.

The lattice is required for the computation and the present lattice of input data before the introduction of new objects would be required for the update of the lattice when adding the objects. However, that lattice can be large, the application may not store it (it can be even impossible due to space requirements) or it can be stored at different (presentation) place than the computation place, or there can be other drawbacks and difficulties of keeping the whole lattice. Moreover, 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 (depending on the particular algorithm) extensive modifications of the incremental algorithm.

In the following we propose 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 objects only, not requiring the concept lattice of present objects. The algorithm results as a modification of Kuznetsov's Close-by-One (CbO) algorithm [8–10] for computing the set of all formal concepts or any of its recent derivatives including FCbO [14], PCbO [7] or PFCbO [6]. When introducing new objects to input data the modified algorithm computes and outputs the resulting new and modified formal concepts only and the respective changes in the lattice order relation (pairs of formal concepts to be created or deleted). Note that (1) new formal concept here is a concept in new data with (some of the) introduced objects in its object set (extent) and attribute set (intent) not equal to intent of any concept in the data before the update and modified formal concept is a concept in new data with the same intent as some existing concept in data before update and enlarging its extent by (some of the) introduced objects, other formal concepts in new data are called old; (2) since the concept lattice (before update) is not required by the algorithm the computed changes in the lattice order relation are output only and have to be applied where the (part of the) lattice is stored and (3) introducing new objects to input data cannot result in removal of formal concepts from the concept lattice [4, 15]. Moreover, considering instead the new objects to be some of the objects of input data to be deleted from the data, the present objects to be the set of objects of input data after the deletion and the computed formal concepts to be resulting concepts to remove or modify together with the "inverse" changes of the lattice order relation, we have easily handled also the change of input data consisting of deleting existing objects. The change by altering objects can be handled by first deleting the objects and then by introducing the altered objects, interpreting the output formal concepts accordingly. Finally, changes of input data by introducing new and deleting or altering existing attributes are completely analogical and can be handled (better than by altering objects) by an algorithm re-described with objects and attributes switched or by the present algorithm with objects and attributes switched in input data (transposed data) and in 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 AddIntent algorithm [12], which is considered one of the up-to-date most efficient incremental algorithms for computing concept lattice.

2 The algorithm

We describe the algorithm as a modification of Fast Close-by-One (FCbO) algorithm [14], as it happens to be one of the up-to-date most efficient (sequential/serial) derivatives of CbO [5]. In essence, the modification equally applies also to other CbO derivatives and CbO itself. A deep knowledge of FCbO is not required in the description, however, basic knowledge is beneficial. 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 descriptions below we assume the reader is familiar with basics of FCA, see [2, 3] for reference. Input object-attribute data (formal context) is 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 [3] are denoted by ${}^{\uparrow_I} : 2^X \mapsto 2^Y$ and ${}^{\downarrow_I} : 2^Y \mapsto 2^X$, $\mathcal{B}(X, Y, I)$ denotes the set of all formal concepts in I and $\leq$ the partial order relation on $\mathcal{B}(X, Y, I)$ forming together a concept lattice.

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. Let us suppose we are introducing to input data $\langle X, Y, I\rangle$ (finite nonempty set of) new objects $X_N = \{m+1, \ldots, m'\}$ not present in X ($X_N \cap X = \emptyset$), 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 usual scenario of introducing new objects to input data 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 input data with new objects added by the triplet $\langle X', Y', I'\rangle$, where $X' = X \cup X_N = \{0, \ldots, m'\}, Y' = Y \cup Y_N = \{0, \ldots, n'\}$, $n' = k$ if $k > n$ and $n' = n$ otherwise, and $I' \subseteq X' \times Y'$ such that $I' \cap (X \times Y) = I$, $I' \cap (X_N \times Y_N) = N$ and $I' \cap (X \times (Y_N \setminus Y)) = I' \cap (X_N \times (Y \setminus Y_N)) = \emptyset$. Hence I' is the union of I and N both extended to X' and Y'.

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 $\mathcal{B}(X', Y', I')$, with the exception of formal concept $\langle Y'^{\downarrow_{I'}}, Y'\rangle$. 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 only attributes Y_N.

Now, for any $B' = B^{\downarrow_{I'}\uparrow_{I'}} \subseteq Y_N$, if $B' = B'' = B^{\downarrow_I\uparrow_I}$ then the formal concept $\langle A', B'\rangle \in \mathcal{B}(X', Y', I')$ with the same intent $B' = B''$ as formal concept $\langle A'', B''\rangle \in \mathcal{B}(X, Y, I)$ is a modified formal concept enlarging the extent of

Algorithm 1: Procedure UpdateFastGenerateFrom($\langle A, B\rangle, y, \{N_y \mid y \in Y\}$), cf. [14]

```
   // list $\langle A, B\rangle$, e.g., print it on screen or store it
1  if $(A \cap X)^{\uparrow_{I'}} = B$ then
2      if $(A \cap X) \subset A$ then
3          list $\langle A, B\rangle$ as modified;
4      end
5  else
6      list $\langle A, B\rangle$ as new;
7  end
8  if $B = Y'$ or $y > n'$ then
9      return
10 end
11 for $j$ from $y$ upto $n'$ do
12     set $M_j$ to $N_j$;
       // go through attributes from $Y_N$ only
13     if $j \notin B$ and $j \in Y_N$ and $N_j \cap Y_j \subseteq B \cap Y_j$ then
14         set $C$ to $A \cap \{j\}^{\downarrow_{I'}}$;
15         set $D$ to $C^{\uparrow_{I'}}$;
16         if $B \cap Y_j = D \cap Y_j$ then
17             put $\langle\langle C, D\rangle, j\rangle$ to $queue$;
18         else
19             set $M_j$ to $D$;
20         end
21     end
22 end
23 while get $\langle\langle C, D\rangle, j\rangle$ from $queue$ do
24     UpdateFastGenerateFrom($\langle C, D\rangle, j + 1, \{M_y \mid y \in Y\}$);
25 end
26 return
```

$\langle A'', B''\rangle$ by objects $A' \setminus A'' \subseteq X_N$ if $A' \supset A''$ (otherwise $\langle A', B'\rangle = \langle A'', B''\rangle$ is old). Otherwise, if $B' \subset B'' = B^{\downarrow_I \uparrow_I}$ (since the $\supset$ inclusion cannot happen since $X' \supset X$) then the formal concept $\langle A', B'\rangle \in \mathcal{B}(X', Y', I')$ is a new formal concept and the formal concept $\langle A'', B''\rangle \in \mathcal{B}(X, Y, I)$ is called its generator [4, 15]. In both cases, $A' \cap X = A''$.

We use the above presented ideas to modify FCbO algorithm described in [14] 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 $\langle X', Y', I'\rangle$ 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 [14] so we describe only the modifications introduced in Update-FastGenerateFrom.

The procedure accepts as its arguments a formal concept $\langle A, B\rangle$ (an initial formal concep t), an attribute $y \in Y_N$ (first attribute to be processed) and a set $\{N_y \subseteq Y \,|\, y \in Y\}$ of subsets of attributes Y (see [14] for the meaning of the set), and uses local variables *queue* as a temporary storage for computed formal concepts and M_y $(y \in Y)$ as sets of attributes which are used in pl ace of the third argument for further invocations of the procedure. After invocation, the procedure recursively descends through the space of new and modified formal concepts of $\langle X', Y', I'\rangle$ resulted by adding new objects X_N described by attributes Y_N to $\langle X, Y, I\rangle$.

When invoked with $\langle A, B\rangle$ and $y \in Y_N$ (first attribute to be processed) and $\{N_y \,|\, y \in Y\}$, UpdateFastGenerateFrom first checks if $(A \cap X)^{\uparrow_{I'}}$ equals B (line 1), in which case, if further $(A \cap X) \subset A$ (line 2), $\langle A, B\rangle$ is a modified formal concept (see above) and it is listed as such. In the other case, $\langle A, B\rangle$ is a new formal concept. Then the procedure behaves exactly as the original procedure FastGenerateFrom, see [14] for full description, with the exception that it goes through attributes $j \in Y_N$ only (line 13). Let us recall that the set $Y_j \subseteq Y'$ is defined by

$$Y_j = \{y \in Y' \,|\, y < j\}.$$

In order to list all new and modified formal concepts of $\langle X', Y', I'\rangle$ which are not formal concepts of $\langle X, Y, I\rangle$, each of them exactly once, we invoke UpdateFastGenerateFrom with $\langle \emptyset^{\downarrow_{I'}}, \emptyset^{\downarrow_{I'}\uparrow_{I'}}\rangle$, y being the first attribute in Y_N and $\{N_y = \emptyset \,|\, y \in Y\}$ as its initial arguments. The correctness of Algorithm 1 follows from the correctness of FCbO algorithm [14] and the above description of its modification.

Example 1. We illustrate Algorithm 1 on the following example. Consider an input data given in an 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	×		×	×		×

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

$$C_1^* = \langle\{0,1,2,3,4\}, \emptyset\rangle, \qquad C_6^* = \langle\{1,2,3\}, \{1,5\}\rangle,$$
$$C_2^* = \langle\{0,2,4\}, \{0\}\rangle, \qquad C_{11} = \langle\{1,3\}, \{1,3,5\}\rangle,$$
$$C_5^* = \langle\{0,4\}, \{0,2,3\}\rangle, \qquad C_8^* = \langle\{0,1,3,4\}, \{3\}\rangle,$$
$$C_9 = \langle\{4\}, \{0,2,3,5\}\rangle, \qquad C_{12} = \langle\{1,3,4\}, \{3,5\}\rangle,$$
$$C_{10} = \langle\{2,4\}, \{0,5\}\rangle, \qquad C_{13} = \langle\{1,2,3,4\}, \{5\}\rangle.$$

Now 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 5 new formal concepts C_9 to C_{13} and 5 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 input data $\langle X'', Y'', I'' \rangle$ incrementally, 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 $i \in X'' = \{0, \ldots, n''\}$, setting $\langle X, Y, I \rangle := \langle \{0, \ldots, i-1\}, \bigcup_{j=0}^{i-1} \{j\}^{\uparrow_{I''}}, I \cap (\{0, \ldots, i-1\} \times \bigcup_{j=0}^{i-1} \{j\}^{\uparrow_{I''}}) \rangle$, $X_N := \{i\}$, $Y_N := \{i\}^{\uparrow_{I''}}$ 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 (though efficient) computations of formal concepts listed as modified by procedure UpdateFastGenerateFrom, see the performance evaluation in Section 3. The concepts are, moreover, filtered out from the listing but, however, necessary to compute in order to decide whether a concept is new or modified. Incremental algorithms iterate over (so far) incrementally computed and stored concept lattice to decide and compute new formal concepts which is more efficient. On the other hand, as discussed in introduction Section 1, the concept lattice is required for the computation and must be stored by the incremental algorithms, while Algorithm 1 requires input data only/instead 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. Note that the modified formal concepts cannot raise a change in the relation since attribute set (intent) of a modified concept remains unchanged with introduction of new objects to input data, as discussed above.

In order to determine the changes in concept lattice order relation by the modified FCbO algorithm introduced in previous Section 2.1, we first have to determine the concept lattice order relation alone since the FCbO algorithm, as well as the modification, computes the set of formal concepts of input data only. So, in the following, we describe an extension of Fast Close-by-One (FCbO) algorithm [14] to determine the concept lattice order relation $\leq$ on the set of all formal concepts $\mathcal{B}(X, Y, I)$ of input data $\langle X, Y, I \rangle$ computed by the original algorithm, i.e. making 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 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 or deleted in the relation at the place where it is stored.

We now describe how to extend FCbO algorithm, as described in [14], to determine the concept lattice order cover relation on the set of computed formal concepts, i.e. to compute concept lattice. We can use the pseudocode in Algorithm 1 if we look apart from the modifications to FCbO introduced there (the check whether $\langle A, B\rangle$ is new or modified formal concept between lines 1 and 6 and going through attributes from Y_N only at line 12), in which case we obtain the original procedure FastGenerateFrom from [14]. We will need a little knowledge of FCbO (or CbO) now. The algorithm, see the pseudocode of procedure UpdateFastGenerateFrom, computes a formal concept $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle$ (lines 14 and 15) from a formal concept $\langle A, B\rangle$ for all attributes $j \in Y$ such that $y \leq j \leq n$ which are not present in B (lines 12 and 13). Certainly $\langle C, D\rangle \leq \langle A, B\rangle$. If $\langle C, D\rangle$ passes the canonicity test (line 16) and is listed we list the pair $\langle\langle C, D\rangle, \langle A, B\rangle\rangle$ as to be created in the cover relation in spite of that the formal concepts in it need not fulfill the definition of cover relation. We do it since a test for the fulfilment would be too expensive compared to that in the case of nonfulfilment we will list the pair later again as to be deleted from the relation.

Next, since further formal concepts $\langle C \cap \{j'\}^{\downarrow}, (C \cap \{j'\}^{\downarrow})^{\uparrow}\rangle$ are computed from $\langle C, D\rangle$ in the next recursive invocation of (Update)FastGenerateFrom for attributes $j' \geq j + 1$ ($j + 1$ is passed in y argument in the invocation, line 20), in order to list pairs $\langle\langle E, F\rangle, \langle C, D\rangle\rangle$ to be created in the cover relation, we determine formal concepts $\langle E, F\rangle = \langle C \cap \{i\}^{\downarrow}, (C \cap \{i\}^{\downarrow})^{\uparrow}\rangle$ for all attributes $j - 1 \geq i \geq 0$ (which are not present in D). Due to the order in which formal concepts are computed by FCbO, formal concepts $\langle E, F\rangle$ were already computed (and listed) in some of the previous stages of the algorithm before the computation of $\langle C, D\rangle$. However, since formal concepts are not stored in FCbO nor in our extension, we have to compute the concepts $\langle E, F\rangle$ repeatedly. Again, formal concepts $\langle E, F\rangle, \langle C, D\rangle$ in the pairs need not fulfill the definition of cover relation. Here, however, we can use a cheap test known from Lindig's NextNeighbor algorithm [11], but limited to attributes $0, \ldots, j - 1 \in Y$. The test is based on the fact that a formal concept $\langle C, D\rangle \neq \langle Y^{\downarrow}, Y\rangle$ covers a formal concept $\langle E, F\rangle = \langle C \cap \{i\}^{\downarrow}, (C \cap \{i\}^{\downarrow})^{\uparrow}\rangle, i \notin D$ iff $(C \cap \{k\}^{\downarrow})^{\uparrow} = F$ for all attributes $k \in F \setminus D$ (cf. Theorem 1 in [11]). If the test passes, we list the pair $\langle\langle E, F\rangle, \langle C, D\rangle\rangle$ as to be created in the cover relation. We do it even for formal concepts in the pair which pass the test and do not fulfill the definition of cover relation due to the limitation of the test to attributes $0, \ldots, j - 1 \in Y$ for the same reason as above for pairs $\langle\langle C, D\rangle, \langle A, B\rangle\rangle$. To resolve these cases we simply, together with listing the pair $\langle\langle E, F\rangle, \langle C, D\rangle\rangle$, list as to be deleted from the relation the pair $\langle\langle E, F\rangle, \langle A, B\rangle\rangle$ which does not fulfill the definition of the relation and could have been listed as to be created in the cover relation in this or the previous stage (previous recursive invocation of procedure (UPDATE)FASTGENERATEFROM) of the extended algorithm. The justification of this and that all such pairs will be listed as to be deleted from the cover relation is postponed to the full version of the paper.

The modified procedure LATTICEFASTGENERATEFROM, which implements the above presented ideas to procedure FASTGENERATEFROM to extend FCbO algorithm to determine the concept lattice order cover relation on the set of formal concepts computed by FCbO, i.e. to compute concept lattice, is depicted in Algorithm 2. As with the procedure UPDATEFASTGENERATEFROM in Section 2.1, we describe only the modifications introduced to FASTGENERATEFROM.

The original FCbO algorithm remains in essence intact, including its arguments and local variables, all the modifications to original procedure FASTGENERATEFROM are in the computed formal concept $\langle C, D\rangle$ processing part before the recursive call of the procedure (line 28), between lines 16 and 29. First note that the listing of $\langle C, D\rangle$ moved from the beginning of the procedure (see Algorithm 1) here, resulting effectively in the need to list the formal concept passed in the initial invocation of the procedure before the invocation. Note also that this move does not change the order of listed formal concepts. Then, after listing the pair $\langle\langle C, D\rangle, \langle A, B\rangle\rangle$ as to be created in the cover relation, between lines 18 and 27, formal concepts $\langle E, F\rangle = \langle C \cap \{i\}^{\downarrow}, (C \cap \{i\}^{\downarrow})^{\uparrow}\rangle, i \notin D$ covered by $\langle C, D\rangle$ are re-computed (lines 20 and 21) and listed in pairs $\langle\langle E, F\rangle, \langle C, D\rangle\rangle$ as to be created in the cover relation, for attributes $j - 1 \geq i \geq 0$. The test of covering (line 23) is performed using the *min* local variable in a slightly modified form borrowed from the original description of Lindig's NextNeighbor algorithm in [11]. With listing of $\langle\langle E, F\rangle, \langle C, D\rangle\rangle$, the pair $\langle\langle E, F\rangle, \langle A, B\rangle\rangle$ is listed as to be deleted from the cover relation as discussed above.

In order to output concept lattice of $\langle X, Y, I\rangle$, that means to list all formal concepts of $\langle X, Y, I\rangle$, each of them exactly once, together with all pairs of formal concepts to create or delete (if the pair exists) in order to obtain the cover relation of concept lattice of $\langle X, Y, I\rangle$, each pair of formal concepts in the cover relation listed exactly once, we first list $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$ and then invoke 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 [14] and the above description of its extension.

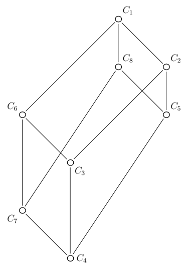
Determination of changes in (the cover relation of) concept lattice of $\langle X, Y, I\rangle$ needed to be done in reaction to the introduction of new objects to input data is then a matter of merging the Algorithms 1 and 2. In addition to that we list, for each new formal concept $\langle C, D\rangle$ computed from formal concept $\langle A, B\rangle$ (or $\langle C, D\rangle = \langle \emptyset^{\downarrow_{I'}}, \emptyset^{\downarrow_{I'}\uparrow_{I'}}\rangle$) and its generator $\langle E, F\rangle = \langle C \cap X, (C \cap X)^{\uparrow_{I'}}\rangle$, with listing of $\langle C, D\rangle$ also pairs $\langle\langle F^{\downarrow_{I'}}, F\rangle, \langle C, D\rangle\rangle$ as to be created in the cover relation and $\langle\langle F^{\downarrow_{I'}}, F\rangle, \langle A, B\rangle\rangle$ as to be deleted from the cover relation. The pairs will then not be listed in the extension introduced in procedure LATTICEFASTGENERATEFROM. Due to space limitations of the paper, the merged algorithm is postponed to the full version of the paper.

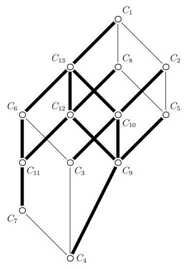
Algorithm 2: Procedure LatticeFastGenerateFrom($\langle A, B\rangle, y, \{N_y \mid y \in Y\}$), cf. [14]

```
if $B = Y$ or $y > n$ then
    return
end
for $j$ from $y$ upto $n$ do
    set $M_j$ to $N_j$;
    if $j \notin B$ and $N_j \cap Y_j \subseteq B \cap Y_j$ then
        set $C$ to $A \cap \{j\}^{\downarrow}$;
        set $D$ to $C^{\uparrow}$;
        if $B \cap Y_j = D \cap Y_j$ then
            put $\langle\langle C, D\rangle, j\rangle$ to $queue$;
        else
            set $M_j$ to $D$;
        end
    end
end
while get $\langle\langle C, D\rangle, j\rangle$ from $queue$ do
    // list $\langle C, D\rangle$, e.g., print it on screen or store it
    // list $\langle\langle C, D\rangle, \langle A, B\rangle\rangle$ as to be created in the cover relation,
       e.g., print $C$ and $A$ (or $D$ and $B$) on screen or store them
    set $min$ to $Y_j$;
    for $i$ from $j - 1$ downto $0$ do
        if $i \notin D$ then
            set $E$ to $C \cap \{i\}^{\downarrow}$;
            set $F$ to $E^{\uparrow}$;
            set $min$ to $min \setminus \{i\}$;
            if $D \cap Y_j \cap min = F \cap Y_j \cap min$ then
                // list $\langle\langle E, F\rangle, \langle C, D\rangle\rangle$ as to be created in the cover
                   relation
                // list $\langle\langle E, F\rangle, \langle A, B\rangle\rangle$ as to be deleted from the cover
                   relation
                set $min$ to $min \cup \{i\}$;
            end
        end
    end
    LatticeFastGenerateFrom($\langle C, D\rangle, j + 1, \{M_y \mid y \in Y\}$);
end
return
```

Example 2. We illustrate Algorithm 2 and the merged algorithm 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, listed in the order in which they are listed by procedure LatticeFastGenerateFrom, below the lattice.

Next (to the right) to the lattice there is then the concept lattice and below the listing of the 8 formal concepts the listing of the 5 new formal concepts C_9 to C_{13} and changes in the cover relation after the introduction of the two new objects to the data. The changes are depicted in bold face in the lattice and the listing of concepts and pairs of concepts in the changes are again listed down-right in order in which they would be listed by a procedure describing the merged algorithm.





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

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

3 Experimental evaluation

The asymptotic worst-case time complexity of Algorithm 1 remains the same as of 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 Algorithm 2 is in the worst case $|Y|$ factor higher but on average it performs a constant factor slower than FCbO (and ramification of the worst-case time complexity of FCbO itself remains a challenging open problem [14]).

We have run several experiments to evaluate the performance of Algorithms 1 and 2 and also the merged algorithm. For listing all formal concepts or concept lattice of input data we also compared the algorithms with AddIntent incremental algorithm [12] for computing concept lattice. In the comparison, we also run Algorithm 1 and the merged algorithm in the way processing input data incrementally as mentioned in Remark 1, for the sake of presenting more fair comparison with true incremental algorithm, though such 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 [14], while the implementation of AddIntent algorithm was borrowed from one of the authors of [12].

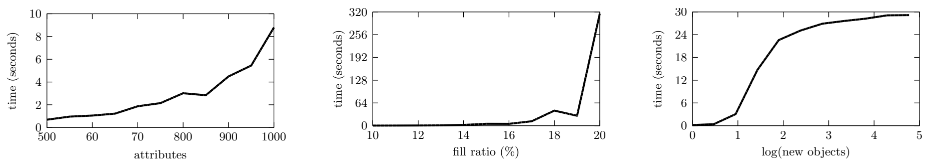


Fig. 1. Running time of Algorithm 1 on random data dependent on number of attributes (on the left, introducing a single new object), on fill ratio (percentage of ×s, in the middle, introducing a single new object) and on number of new objects (on the right).

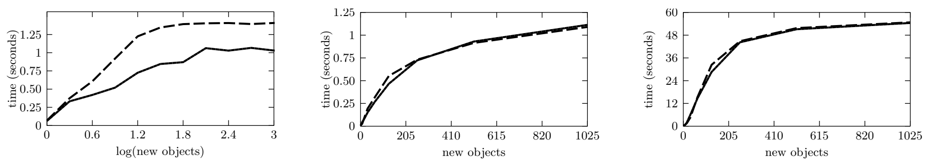


Fig. 2. Running time of Algorithm 1 dependent on number of new (last) objects for mushroom (on the left), anonymous-msweb (in the middle) and T10I4D100K datasets (on the right), solid line—orig. object ordering, dashed line—random object ordering.

The experiments were run on otherwise idle 32-bit i386 hardware (2× Intel Xeon 3.2 GHz, 3 GB RAM) and we were interested in the performance of all algorithms measured by running time. We have run the algorithms on synthetic randomly generated data with various size and percentage of ×s in the table (fill ratio, with normal distribution) as well as with three selected real benchmark datasets from the UCI Machine Learning Repository [1].

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 100000 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 100000 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. The illustration for the benchmark datasets, 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 (right). In the graph the solid line is for data with objects ordered as in the original dataset 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,

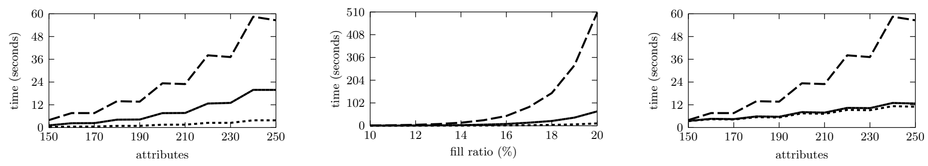


Fig. 3. Running time dependent on number of attributes (on the left and right) and on fill ratio (percentage of ×s, in the middle), solid line—Algorithm 2 (left, middle)/merged algorithm (right), dashed line—AddIntent, dotted line—FCbO (left, middle)/Algorithm 1 (right).

Table 1. Running time (in seconds) of determining the cover relation of concept lattice and numbers of formal concepts for selected datasets.

dataset	size	fill ratio	#concepts	orig. object ordering			random object ordering		
				Alg. 2	AddInt.	FCbO	Alg. 2	AddInt.	FCbO
mushroom	8124 × 119	19.167 %	238710	10.010	9.760	0.830	11.025	8.061	0.919
anon. web	32711 × 296	1.019 %	129009	2.860	6.540	1.326	2.841	6.521	1.306
T10I4D100K	100000 × 1000	1.010 %	2347376	453.060	342.966	55.523	452.893	343.056	55.490

left and middle) that this operation is extremely fast compared to computing all formal concepts which took 7061 seconds for 500 attributes and 355 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 (Figure 1, right, and Figure 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 datasets with FCbO times in Table 1). Note also that running times for mushroom dataset 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 performance for random data is illustrated in Figure 3. The graph on the left shows the dependency of required time on the number of attributes, of data with 10000 objects and fixed table fill ratio 5 %, the graph in the middle shows the dependency on the fill ratio of tables with 1000 objects and fixed 100 attributes. The graphs show also the comparison with AddIntent algorithm 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 datasets is presented in Table 1. Again, the times are given for data with objects ordered as in the original dataset and for data with randomly ordered objects (the time is an average from three orderings), and we also put information on size, fill ratio and the number of all formal concepts for each dataset.

From the graphs we can see that Algorithm 2 considerably outperforms AddIntent algorithm on synthetic random data (in particular for higher fill ratio of data table), while for real benchmark datasets this must not always be the case (T10I4D100K, closely mushroom). This, however, heavily depends on the concept lattice structure (size of the cover relation) of the datasets. We can also

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

dataset	#concepts in total	orig. object ordering			random object ordering		
		merged alg.	AddInt.	Alg. 1	merged alg.	AddInt.	Alg. 1
mushroom	9577435	130.746	9.760	128.993	284.983	8.061	284.130
anon. web	992259	41.386	6.540	41.320	41.342	6.521	41.077
T10I4D100K	14240918	2389.180	342.966	2384.160	2388.793	343.056	2378.703

see from the comparison with 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. Also, ordering of objects in the benchmark datasets makes almost no difference in the running times.

Finally, the comparison of performance of Algorithm 1 and the merged algorithm 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 graph shows the performance for the data with 10000 objects and fixed table fill ratio 5 % from the preceding evaluation of Algorithm 2 (the time is an average from three random orderings of objects); the solid line is for the merged algorithm, the dashed line is for AddIntent and the dotted line is for Algorithm 1. 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 datasets).

The results are really interesting here. For the real benchmark datasets AddIntent algorithm significantly outperforms our algorithms, see Table 2. This was expected since, as hinted by Remark 1, the algorithms are not designed and ment to be used as incremental algorithms (processing objects one by one). What is, however, very surprising is that on synthetic random data both Algorithm 1 and the merged algorithm considerably outperform AddIntent and, moreover, computing concept lattice (determining cover relation) takes just a little more time than computing the set of formal concepts only. It seems that the usage of the algorithms as incremental algorithms, after all, deserves more attention!

4 Conclusion

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. 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 computed before the update as the so-called incremental (update) algorithms do. The algorithms result as modification or extension of 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 by the first algorithm significantly faster than re-computing all formal concepts or whole concept lattice and that the second algorithm is performance comparable to incremental algorithms for computing concept lattice.

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

References

1. Bache K., Lichman M.: *UCI Machine Learning Repository.* University of California, Irvine, CA, School of Information and Computer Sciences, 2013. `http://archive.ics.uci.edu/ml`
2. Carpineto C., Romano G.: *Concept data analysis. Theory and applications.* J. Wiley, 2004.
3. Ganter B., Wille R.: *Formal concept analysis. Mathematical foundations.* Berlin: Springer, 1999.
4. Godin R., Missaoui R., Alaoui H.: Incremental Concept Formation Algorithms Based on Galois Lattices. *Computation Intelligence* **11**(1995), 246-267.
5. Kirchberg M., Leonardi E., Tan Y. S., Link S., K L Ko R., Lee B. S.: Formal Concept Discovery in Semantic Web Data. In: *Proc. ICFCA 2012, LNAI*, **7278**(2012), 164–179.
6. Krajca P., Outrata J., Vychodil V.: Advances in algorithms based on CbO. In: *Proc. CLA 2010*, 325–337.
7. Krajca P., Outrata J., Vychodil V.: Parallel algorithm for computing fixpoints of galois connections. *Annals of Mathematics and Artificial Intelligence* **59**(2)(2010), 257–272.
8. Kuznetsov S. O.: Interpretation on graphs and complexity characteristics of a search for specific patterns. *Automatic Documentation and Mathematical Linguistics*, **24**(1)(1989), 37–45.
9. Kuznetsov S. O.: A fast algorithm for computing all intersections of objects in a finite semi-lattice. *Automatic Documentation and Mathematical Linguistics*, **27**(5)(1993), 11–21.
10. Kuznetsov S. O.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. *PKDD 1999*, 384–391.
11. Lindig C.: Fast concept analysis. *Working with Conceptual Structures–Contributions to ICCS 2000*, 2000, 152–161.
12. van der Merwe D., Obiedkov S. A., Kourie D. G.: AddIntent: A New Incremental Algorithm for Constructing Concept Lattices. In: *Proc. ICFCA 2004, LNAI* **2961**(2004), 205–206.
13. Norris E. M.: An Algorithm for Computing the Maximal Rectangles in a Binary Relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, **23**(2)(1978), 243–250.
14. Outrata J., Vychodil V.: Fast Algorithm for Computing Fixpoints of Galois Connections Induced by Object-Attribute Relational Data. *Information Sciences* **185**(1)(2012), 114-127.
15. Valtchev P., Missaoui R.: Building Concept (Galois) Lattices from Parts: Generalizing the Incremental Methods. *LNAI* **2120**(2001), 290-303.
16. Wille R.: Restructuring lattice theory: an approach based on hierarchies of concepts. *Ordered Sets*, 1982, 445–470, 1982.