Advances in algorithms based on CbO

Petr Krajca, Jan Outrata, Vilem Vychodil[*]

Department of Computer Science, Palacky University, Olomouc, Czech Republic
Tř. 17. listopadu 12, 771 46 Olomouc, Czech Republic
krajcap@inf.upol.cz,{jan.outrata,vilem.vychodil}@upol.cz

Abstract. The paper presents a survey of recent advances in algorithms for computing all formal concepts in a given formal context which result as modifications or extensions of CbO. First, we present an extension of CbO, so called FCbO, and an improved canonicity test that significantly reduces the number of formal concepts which are computed multiple times. Second, we outline a parallel version of the proposed algorithm and discuss various scheduling strategies and their impact on the overall performance and scalability of the algorithm. Third, we discuss important data preprocessing issues and their influence on the algorithms. Namely, we focus on the role of attribute permutations and present experimental observations about the efficiency of the proposed algorithms with respect to the number of inversions in such permutations.

1 Introduction

The major issue of widely-used algorithms for computing formal concepts, including CbO [12–14], NextClosure [5, 6], or UpperNeighbor [16], is that some concepts are computed multiple times which brings significant overhead. This paper deals with various ways to reduce the overhead. Notice that recently an increasing attention has been paid to various modifications of CbO, see [1, 10, 17].

This paper presents a survey of recent advances in three interconnected areas. First, we present an algorithm called FCbO which achieves better performance than CbO by reducing the total number of formal concepts that are computed multiple times. The reduction is achieved by introducing an additional canonicity test which effectively prunes the CbO tree during the computation. Second, we elaborate on issues related to parallel execution of FCbO. We have already proposed a parallel variant of CbO, so-called PCbO [10]. In this paper, we propose an analogous parallelization of FCbO and we discuss various workload distribution strategies that may have impact on the overall performance of such parallelization. Third, we focus on data preprocessing—an important issue that is often underestimated. Namely, some algorithms for FCA (including those from the CbO family) achieve better performance if attributes are processed in particular order. In this paper, we present a preliminary study of the role of attribute permutations on the performance of CbO and the derived algorithms.

[*] Supported by grant no. P103/10/1056 of the Czech Science Foundation and by grant no. MSM 6198959214.

Notation Throughout the paper, $X = \{0, 1, \ldots, m\}$ and $Y = \{0, 1, \ldots, n\}$ are finite nonempty sets of objects and attributes, respectively, and $I \subseteq X \times Y$ is an incidence relation. The triplet $\langle X, Y, I\rangle$ is a formal context. The concept-forming operators induced by I will be denoted by ${}^{\uparrow_I} : 2^X \to 2^Y$ and ${}^{\downarrow_I} : 2^Y \to 2^X$, respectively, see [6] for details. We assume that reader has knowledge of basic algorithms for FCA.

2 FCbO: Fast Close-by-One with New Canonicity Test

In this section we briefly describe the new canonicity test and a new algorithm derived from CbO which uses this test. Recall that the original canonicity test used by CbO (and NextClosure) is always used *after* a new formal concept is computed. For $B \subseteq Y$ and $j \notin B$, one checks whether

$$B \cap Y_j = D \cap Y_j, \text{ where } D = (B \cup \{j\})^{\downarrow_I \uparrow_I} \tag{1}$$

and $Y_j = \{y \in Y \mid y < j\}$. FCbO employs an additional test that is performed *before* D is computed, eliminating thus the computation of ${}^{\downarrow_I \uparrow_I}$. Notice that (1) fails iff $B \oslash j \neq \emptyset$, where

$$B \oslash j = (D \setminus B) \cap Y_j = \big((B \cup \{j\})^{\downarrow_I \uparrow_I} \setminus B\big) \cap Y_j. \tag{2}$$

The new canonicity test exploits the fact that if (1) fails for given B and $j \notin B$, the monotony of ${}^{\downarrow_I \uparrow_I}$ yields that the test will also fail for each $B' \supseteq B$ such that $j \notin B'$ and $B \oslash j \nsubseteq B'$. The conclusion can be done without computing D. If $B \oslash j \subseteq B'$, we are still compelled to perform the original canonicity test. Thus, the new (additional) canonicity test is based on the following assertion:

Lemma 1 (See [17]). *Let $B \subseteq Y$, $j \notin B$, and $B \oslash j \neq \emptyset$. Then, for each $B' \supseteq B$ such that $j \notin B'$ and $B \oslash j \nsubseteq B'$, we have $B' \oslash j \neq \emptyset$.* □

FCbO can be seen as an extended version of CbO in that we propagate the information about sets (2) which take part in the new test. The information is propagated in the top-down direction. In order to apply the new test, we have to change the search strategy of the algorithm from the depth-first search (as it is in CbO) to a combined depth-first and breadth-first search. FCbO is represented by a recursive procedure FastGenerateFrom, see Algorithm 1, which accepts three arguments: a formal concept $\langle A, B\rangle$ (an initial formal concept), an attribute $y \in Y$ (first attribute to be processed), and a set $\{N_y \subseteq Y \mid y \in Y\}$ of subsets of attributes Y the purpose of which is to carry information about sets (2). Each invocation of FastGenerateFrom has its own local *queue* used to store information about computed concepts. Unlike CbO, if the canonicity tests succeed (line 7 and line 10), we do not call FastGenerateFrom recursively but we store information about the concept in the queue (line 11). After each attribute is processed, we perform the recursive calls, see lines 17–19. The new canonicity test is performed in line 7 based on information stored in N_j's. The original canonicity test is performed in line 10. If the test in line 10 fails, we update the contents of M_j, see line 13. Note that M_j's can be seen as local copies

Algorithm 1: Procedure FastGenerateFrom($\langle A, B\rangle, y, \{N_y \mid y \in Y\}$)

```
1  list $\langle A, B\rangle$        // concept $\langle A, B\rangle$ is processed, e.g., listed or stored
   // check halting condition of the current call
2  if $B = Y$ or $y > n$ then
3      return
4  end
   // process all attributes beginning with $y$
5  for $j$ from $y$ upto $n$ do
6      set $M_j$ to $N_j$                          // $M_j$ is a pointer to $N_j$
       // perform new canonicity test
7      if $j \notin B$ and $N_j \cap Y_j \subseteq B \cap Y_j$ then
           // compute new concept $\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow_I}, (B \cup \{j\})^{\downarrow_I\uparrow_I}\rangle$
8          set $C$ to $A \cap \{j\}^{\downarrow_I}$
9          set $D$ to $C^{\uparrow_I}$
           // perform original canonicity test
10         if $B \cap Y_j = D \cap Y_j$ then
               // store new concept for further processing
11             put $\langle\langle C, D\rangle, j\rangle$ to queue
12         else
               // update information about implied attributes
13             set $M_j$ to $D$                // $M_j$ becomes a pointer to $D$
14         end
15     end
16 end
   // perform recursive calls of FastGenerateFrom
17 while get $\langle\langle C, D\rangle, j\rangle$ from queue do
18     FastGenerateFrom($\langle C, D\rangle, j + 1, \{M_y \mid y \in Y\}$)
19 end
   // terminate current call
20 return
```

of N_j's which are used as the third argument for consecutive calls of FastGenerateFrom. Sets N_j are used instead of (2) because it is actually easier (and more efficient) to maintain a set of pointers to intents than to compute (and allocate memory for) sets (2) during the computation.

FCbO is correct: when invoked with $\langle \emptyset^{\downarrow_I}, \emptyset^{\downarrow_I\uparrow_I}\rangle$, $y = 0$, and $\{N_y = \emptyset \mid y \in Y\}$, Algorithm 1 lists all formal concepts in $\langle X, Y, I\rangle$ in the same order as CbO, each of them exactly once. Let us note that FCbO can be turned into a "Fast NextClosure" (i.e., an algorithm that lists concepts in the lexicographical order [5]) by either (i) using a *stack* instead of a *queue* or by (ii) modifying the loop in line 5 so that it goes "**from** n **downto** y". See [17] for further details on FCbO.

Example 1. Consider a context with $X = \{0, \ldots, 3\}$, $Y = \{0, \ldots, 5\}$, and $I = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 0,2\rangle, \langle 1,0\rangle, \langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle, \langle 2,0\rangle, \langle 2,1\rangle, \langle 2,4\rangle, \langle 3,1\rangle, \langle 3,2\rangle\}$. This formal context induces 12 formal concepts denoted $C_1, \ldots, C_{12}$. In case of both CbO and FCbO, the computation can be depicted by a tree. Moreover, a

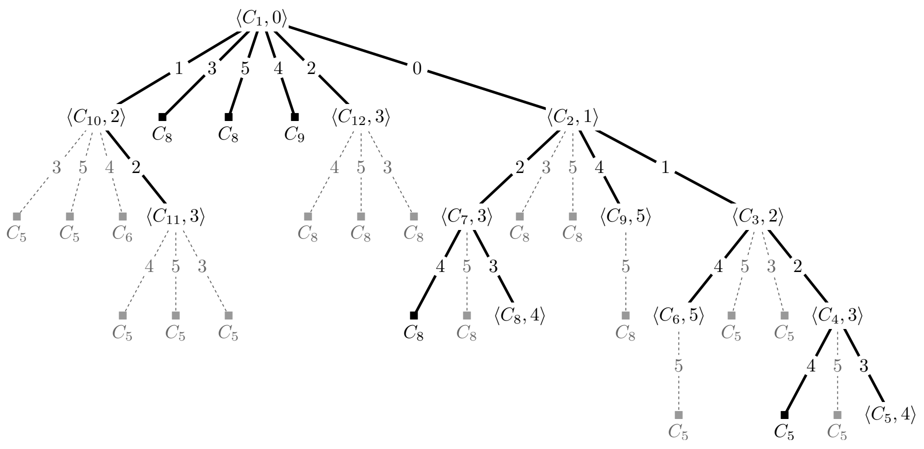

Fig. 1. Example of an FCbO tree—a pruned CbO tree.

	concepts	closures (CbO)	closures (FCbO)	ratio (CbO)	ratio (FCbO)
mushroom	238,710	4,006,498	426,563	5.9 %	55.9 %
anon. web	129,009	27,949,552	1,475,341	0.4 %	8.7 %
debian tags	38,977	12,045,680	679,911	0.3 %	5.7 %
tit-tac-toe	59,505	221,608	128,434	26.8 %	46.3 %

Table 1. Total numbers of closures computed by CbO and FCbO.

FCbO tree is a pruned version of the CbO tree, see Fig. 1. The black-square nodes represent concepts computed multiple times by FCbO and CbO whereas the grey-square nodes represent concepts computed multiple times by CbO and not computed by FCbO. Therefore, grey nodes and edges in Fig. 1 denote subtrees pruned using the new canonicity test. In this case, the number of concept computed multiple times is significantly reduced.

Experimental Evaluation We have evaluated FCbO and compared CbO and FCbO using various real data sets and artificial data sets. The impact of the new canonicity test is presented in Table 1 comparing the total numbers of closures computed by CbO and FCbO in selected benchmark data sets [2, 8]. The table includes numbers of concepts and ratios of the number of computed closures to the number of (distinct) formal concepts in the data, i.e., the frequency of successful canonicity tests. Apparently, FCbO has a higher rate of successful canonicity tests than CbO. Thus, in terms of the number of computed closures, FCbO is more efficient than CbO. Since the total number of computed closures directly influences the speed of the algorithm, FCbO is (usually) faster than CbO [17]. The reduction of total time needed for computing all formal concepts is apparent from Table 2. The table shows total time (in seconds) needed to

	mushroom	tic-tac-toe	debian tags	anon. web
size	$8,124 \times 119$	958×29	$14,315 \times 475$	$32,710 \times 295$
density	19 %	34 %	< 1 %	1 %
FCbO	0.23	0.02	0.10	0.15
CbO	4.34	0.06	5.31	27.14
NextClosure	685.00	1.86	1,432.25	8,236.85
UpperNeighbor	4,368.19	12.54	2,159.80	11,068.52
Berry's [3]	950.73	6.93	1,512.73	4,421.51

Table 2. Performace of algorithms (speed in seconds).

analyze the data sets. For the purpose of comparison the table contains also other well-known algorithms. The experiments were performed on an Apple MacPro computer equipped with two quad-core processors (Intel Xeon, 2.8 GHz) and 16 GB of RAM and all algorithms were implemented in ANSI C using bitarray representation [11]. Notice that in the worst case, FCbO collapses into CbO (e.g., in case of I being the inequality relation on $X = Y$). FCbO is a polynomial time-delay algorithm [7, 9] because the additional canonicity test has a linear time-delay overhead compared to CbO, see [17] for further details on FCbO and its performance.

3 PFCbO: Parallel FCbO and Workload Distribution

This section is devoted to parallelization issues of FCbO. Recall that in [10], we have described PCbO which results by a parallelization of CbO. FCbO can be turned into a parallel algorithm in much the same way as the original CbO can be turned into PCbO. Since the procedure of parallelization is fairly similar to that presented in [10], we focus mainly on issues that are not discussed in [10]. Namely, we compare several strategies to balance the workload distribution among independent processors and compare their efficiency.

Following the ideas from [10], a parallel variant of FCbO consists of three stages: First, we compute and process all concepts that are derivable in less than L steps. Second, we store all concepts derivable in exactly L steps in a new *queue*. Third, we distribute concepts from the *queue* among P independent processors and we let each of the processors compute the remaining concepts using FCbO. Typically, each processor r has its own queue denoted $queue_r$ containing concepts assigned to this processor. A parallel algorithm based on these ideas shall be called *Parallel Fast Close-byOne* (PFCbO).

Clearly, the practical efficiency of both PCbO and PFCbO depends on the choice of the strategy that distributes concepts among processors during the third step of the computation. The decision how to assign concepts to particular queue is generally difficult since we do not know the distribution of formal concepts in the search space of all formal concepts until we actually compute them all and reveal the structure of the call tree. As a consequence, the distribution of workload may be in some cases unbalanced. In [10], we have used a simple round-robin principle which turned out to be reasonably efficient. Nevertheless, there are other schemes of the workload distribution that can be considered:

(i) *round-robin*—concepts are distributed to queues attached to each processor, in the way that n-th concept is placed into a $queue_r$ where $r = (n \bmod P)+1$ and P is the number of processors. For instance, if we consider $P = 4$ and concepts $C_1, \ldots, C_{10}$, they are assigned to queues as follows:

$$queue_1 = \{C_1, C_5, C_9\}, \qquad queue_2 = \{C_2, C_6, C_{10}\},$$
$$queue_3 = \{C_3, C_7\}, \qquad queue_4 = \{C_4, C_8\}.$$

(ii) *zig-zag*—this strategy is similar to the previous strategy but it uses a different formula to determine the $queue_r$. The $queue_r$ is given by

$$r = \min\bigl(n \bmod z, z - (n \bmod z)\bigr) + 1 \tag{3}$$

where $z = 2 \times P + 1$ assuming that P is number of processors. For $P = 4$ and concepts $C_1, \ldots, C_{10}$ the distribution of concepts is

$$queue_1 = \{C_1, C_8, C_9\}, \qquad queue_2 = \{C_2, C_7, C_{10}\},$$
$$queue_3 = \{C_3, C_6\}, \qquad queue_4 = \{C_4, C_5\}.$$

(iii) *blocks*—this workload distribution scheme divides the queue of all concepts into chunks of approximately equal size and these "blocks of concepts" are redistributed into the queues of independent processors. In this case, the n-th concept is placed into $queue_r$, where

$$r = \left\lceil \frac{(n \times P)}{Q} \right\rceil \tag{4}$$

with P being the number of processors, Q being the number of all concepts, and $\lceil x \rceil$ being the usual ceiling function. For instance, in case of $C_1, \ldots, C_{10}$ (i.e., $Q = 10$) and four queues (i.e., $P = 4$), we get:

$$queue_1 = \{C_1, C_2\}, \qquad queue_2 = \{C_3, C_4, C_5\},$$
$$queue_3 = \{C_6, C_7\}, \qquad queue_4 = \{C_8, C_9, C_{10}\}.$$

(iv) *fair*—all concepts remain stored in one shared queue and each processor gets concepts from the queue one by one. The benefit of this scheme is that it allows to react on the revealing structure of the call tree. On the other hand, this method of distributing concepts requires synchronization among processors while accessing this queue. Note that in contrast to the above-described schemes, this scheme has no fixed structure and the workload is distributed non-deterministically.

(v) *random*—the workload is spread among processors randomly. We are considering this strategy to be referential and it is included for the purpose of comparison.

Experimental Evaluation In order to evaluate the strategies of workload distribution, we have tested our algorithm for each strategy using various data sets and various number of processors. Table 3 depicts the time needed to compute all formal concepts using particular strategy. Surprisingly, there are only small

	round-r.	*blocks*	*fair*	*zig-zag*	*random*
debian tags	0.0974	0.0988	0.0938	0.0984	0.0986
anon. web	0.1518	0.1590	0.1500	0.1528	0.1522
mushroom	0.1772	0.2158	0.1550	0.1788	0.1820
tic-tac-toe	0.0172	0.0198	0.0168	0.0174	0.0180
random ($5000 \times 100 \times 10$)	0.0806	0.1194	0.0796	0.0820	0.0876
random ($10000 \times 100 \times 15$)	1.1380	2.1326	0.8698	1.0974	1.1670

Table 3. Performace under various workload distributions (speed in seconds).

differences among the considered schemes of the workload distribution, i.e., the ordinary round-robin used in [10] is indeed adequate for the job. Nevertheless, the *fair* strategy seems to be the most efficient. One can see that the *round-robin* and *zig-zag* strategies provide performance slightly better than the random workload distribution. On the other hand, the *blocks* scheme of distribution provides performance even worse than the random distribution and seems to be inappropriate for PFCbO.

4 Data Preprocessing Issues

Algorithms for computing concepts can be classified in many ways, see, e.g. [15]. An important attribute of algorithms for FCA is whether their performance depends on the order of objects and attributes in the input data table. Therefore, an algorithm for computing formal concepts shall be called (*permutation*) *resistant* whenever all isomorphic copies of a formal context $\langle X, Y, I\rangle$ with $Y = \{0, 1, \ldots, n\}$ require the same number of elementary computation steps in order to compute all concepts. For our purposes, an elementary computation step will be represented by computation of a single fixpoint of the concept-forming operators ${}^{\uparrow_I}$ and ${}^{\downarrow_I}$.

One can easily see that, e.g., Lindig's UpperNeighbor algorithm [16] is resistant. On the other hand, CbO and FCbO are not resistant. Indeed, a different order of attributes in a data table can yield different CbO and FCbO trees that may have different numbers of nodes (notice that the loop in line 5 of Algorithm 1 processes attributes from left to right). Since CbO and FCbO are not resistant, a proper ordering of attributes before computation can further reduce the number of concepts that are computed multiple times, thus improving the efficiency. In this section, we investigate particular permutations of attributes and explore the impact of inversions on the number of computed closures.

In order to describe various formal contexts with respect to the structure of the data table, we introduce a notion of an ordered formal context and inversion:

Definition 1. *An ordered formal context is a formal context $\langle X, Y, I\rangle$ where $Y = \{0, \ldots, n\}$ and for all attributes*

$$|\{0\}^{\downarrow_I}| \leq |\{1\}^{\downarrow_I}| \leq \cdots \leq |\{n\}^{\downarrow_I}|. \quad (5)$$

A pair of attributes $\langle y_1, y_2\rangle \in Y \times Y$ such that $|\{y_1\}^{\downarrow_I}| \not\leq |\{y_2\}^{\downarrow_I}|$ shall be called an inversion.

Verbally, the attributes in an ordered formal context are sorted in the ascending order according to their support, i.e., the number of objects having these attributes. As a consequence of the previous definition, an ordered formal context contains no inversions.

From the point of view of formal concepts and concept lattices, the order of objects and attributes in which they appear in the data table is not essential. Therefore, one can reorder attributes in an arbitrary way. From the computational point of view, however, it may happen that certain orderings of attributes yield better results in conjunction with particular argorithms than other orderings. In case of our algorithms, the order has an important impact on the process of the execution of both CbO and FCbO since the canonicity test is driven by the order of attributes. The following assertions show that for an ordered formal context with pairwise distinct columns, the canonicity tests succeed for all attribute concepts. We first prove a technical claim:

Lemma 2. *Let $\langle X, Y, I\rangle$ be an ordered formal context with $Y = \{0, \ldots, n\}$. Then, for each $k, j \in Y$ such that $k < j$, we have $k \in \{j\}^{\downarrow_I\uparrow_I}$ iff $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$.*

Proof. Note that attributes from Y are integers and "$<$" denotes the usual strict linear order on the set of all integers. Suppose that $k \in \{j\}^{\downarrow_I\uparrow_I}$, i.e., $\{k\} \subseteq \{j\}^{\downarrow_I\uparrow_I}$. By the antitony of $^{\downarrow_I}$, we get $\{k\}^{\downarrow_I} \supseteq \{j\}^{\downarrow_I\uparrow_I\downarrow_I} = \{j\}^{\downarrow_I}$. Thus, it remains to show the converse inclusion. Since $\langle X, Y, I\rangle$ is ordered and $k < j$, we get $|\{k\}^{\downarrow_I}| \leq |\{j\}^{\downarrow_I}|$, see (5). Hence, $|\{k\}^{\downarrow_I}| \leq |\{j\}^{\downarrow_I}|$ and $\{k\}^{\downarrow_I} \supseteq \{j\}^{\downarrow_I}$ yield $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$. Conversely, if $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$ then obviously $k \in \{j\}^{\downarrow_I\uparrow_I}$, proving the claim. □

Applying Lemma 2, we get:

Theorem 1. *Let $\langle X, Y, I\rangle$ be an ordered formal context where $Y = \{0, \ldots, n\}$ and $\{a\}^{\downarrow_I} \neq \{b\}^{\downarrow_I}$ for any $a, b \in Y$. Then for each $j \in Y$ such that $j \not\in \emptyset^{\downarrow_I\uparrow_I}$,*

$$\emptyset^{\downarrow_I\uparrow_I} \cap Y_j = \{j\}^{\downarrow_I\uparrow_I} \cap Y_j, \qquad (6)$$

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

Proof. Take $j \in Y$ such that $j \not\in \emptyset^{\downarrow_I\uparrow_I}$. Observe that condition (6) holds true iff there is no attribute $k \in Y$ such that $k \not\in \emptyset^{\downarrow_I\uparrow_I}$, $k < j$, and $k \in \{j\}^{\downarrow_I\uparrow_I}$. Thus, consider any $k \in Y$ such that $k < j$. Since $\langle X, Y, I\rangle$ is ordered, our assumption $j \not\in \emptyset^{\downarrow_I\uparrow_I}$ yields $k \not\in \emptyset^{\downarrow_I\uparrow_I}$. By the assumption, $\{k\}^{\downarrow_I} \neq \{j\}^{\downarrow_I}$, i.e., Lemma 2 yields $k \not\in \{j\}^{\downarrow_I\uparrow_I}$, finishing the proof. □

Theorem 1 shows that for an ordered formal context with pairwise distinct columns, invocations of FastGenerateFrom in the first level of recursion always succeeds and generates concepts. Moreover, from the proof of Theorem 1 it follows that in any ordered formal context, the first derivation [10] does not exists for attribute j if there is an attribute k such that $k < j$ and $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$.

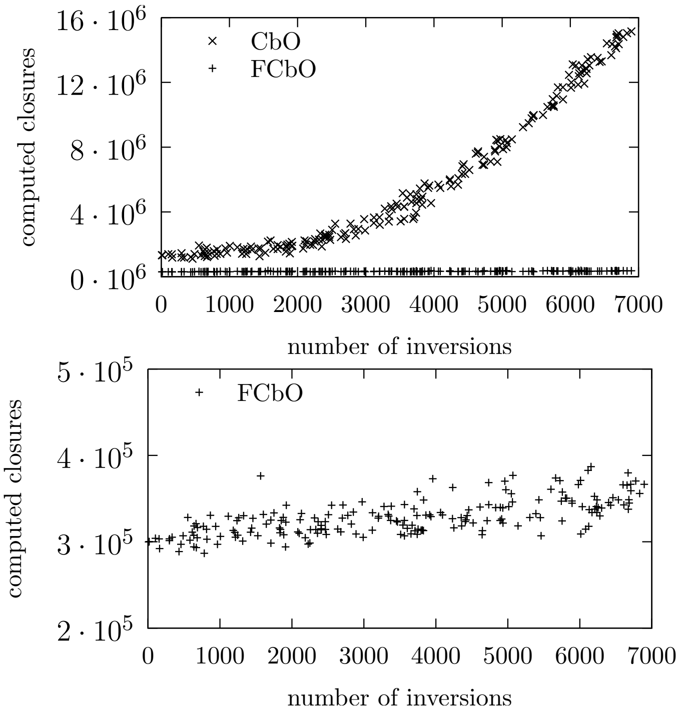

Fig. 2. Impact of inversion in the mushrooms data set

This has a practical consequence for the parallel variants of CbO and FCbO in case of ordered contexts, because it allows us to determine the number of concepts generated during the first stages of the algorithms. If the number of attributes is significantly larger than the number of processors, and this condition is usually fulfilled, it is sufficient to compute only the first derivations and then distribute the workload among all processors.

Furthermore, our empirical experiments have shown an interesting tendency that while processing ordered formal contexts, canonicity tests fail less frequently than in case of contexts containing inversions. In addition, the experiments have shown that with the increasing number of inversions in a data table, the average number of computed closures grows. For instance, Fig. 2 shows how the number of inversions in the *mushroom* data set affects the total number of computed closures. The first graph (at the top) depicts this dependency for CbO and FCbO. The second graph (at the bottom) provides a more detailed view for FCbO. Similar tendency can be observed for other benchmark data sets.

Remark 1. Let us note that the ordering of attributes introduced by (5) has already been used in [4] but the purpose of the ordering in [4] is different. In [4], the authors use this particular ordering of attributes in a parallel version of Ganter's NextClosure algorithm to achieve soundness of the algorithm (each

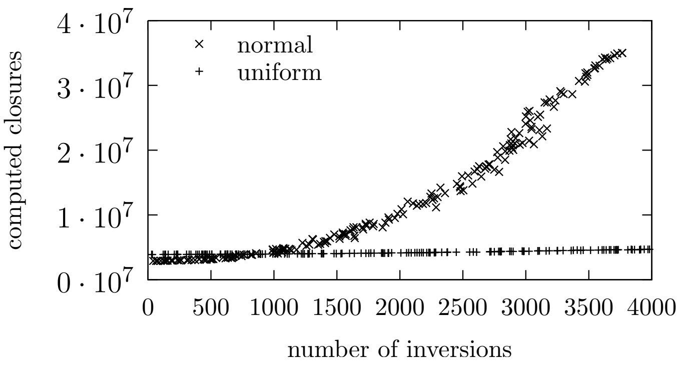

Fig. 3. Impact of inversions in two artificial data sets

	concepts	closures (unordered)	closures (ordered)	ratio (unordered)	ratio (ordered)
mushroom	238,710	426,563	299,201	55.9 %	79.7 %
anon. web	129,009	1,475,341	398,147	8.7 %	32.4 %
debian tags	38,977	679,911	298,641	5.7 %	13.0 %
tit-tac-toe	59,505	128,434	89,930	46.3 %	66.1 %

Table 4. Numbers of computed closures in case of (un)ordered attributes.

concept is then listed only once) while in our case, [4] is used for the sake of increased efficiency.

Remark 2. We have observed a general tendency that certain data sets are more affected by the above-discussed phenomenon than others. For instance, if 1's in a data table are approximately uniformly spread among attributes (i.e., each attribute has approximately the same support), the ordering of attributes (usually) does not have a considerable effect on decreasing the number of closures. Fig. 3 depicts how the increasing number of inversions affects the number of computed closures in two artificial data sets where 1's are distributed (i) approximately uniformly among the attributes and (ii) approximately normally among the attributes. Both data sets have the same parameters in that they consist of 1000 objects, 100 attributes, and contain 15 % of 1's, however, the distributions of 1's among the attributes are quite different. As one can see from Fig. 3, the impact of the number of inversions on the number of computed closures is more significant in case of normally distributed 1's among the attributes.

Experimental Evaluation From our observations it follows that it is desirable to incorporate a preprocessing step which transforms a formal context into a corresponding ordered formal context. In order to evaluate the benefits of this preprocessing step, we have used similar approach as in case of evaluation of

	mushroom	tic-tac-toe	debian tags	anon. web
PFCbO ($P = 1$)	0.23	0.02	0.10	0.15
PFCbO ($P = 2$)	0.14	0.01	0.07	0.11
PFCbO ($P = 4$)	0.09	0.01	0.06	0.09
PFCbO ($P = 8$)	0.06	0.01	0.06	0.08
PCbO ($P = 1$)	4.34	0.06	5.31	27.14
PCbO ($P = 2$)	2.39	0.03	3.59	14.77
PCbO ($P = 4$)	1.65	0.02	2.59	9.22
PCbO ($P = 8$)	0.99	0.01	1.85	5.60

Table 5. Performace with multiple processors (speed in seconds).

the new canonicity test. We have focused on the total numbers of closures and concepts computed by FCbO while processing various ordered and unordered data sets. The results are presented in Table 4 which also includes corresponding ratios.

Apparently, reordering of attributes reduces the number of computed closures, and thus, can reduce time of computation. Note that the Ganter's algorithm [5, 6] is in principle equivalent to CbO. As such, it is also not permutation resistant. Thus, the preprocessing step which reorders attributes can also increase its performance.

5 Overall Evaluation

So far, we have proposed and evaluated several improvements and refinements of the original CbO and PCbO algorithms, namely, new canonicity test, workload distribution schemes, and reordering attributes. However, we have evaluated the impact of each improvement separately. Therefore, we conclude this paper with the evaluation of PFCbO which includes all these improvements.

Table 5 table shows the total time (in seconds) needed to compute all formal concepts in the benchmark data sets using PCbO and PFCbO run on the Apple MacPro computer, equipped with eight processor cores. The parameter P indicates the number of used processors for particular experiment.

Fig. 4 demonstrates the scalability of PFCbO, i.e., the ability to decrease the time of computation by using more processors. In the depicted two experiments, we have used computer equipped with Sun UltraSPARC T1 processor having eight cores (each capable to process up to 4 threads simultaneously) and 8 GB of RAM. Fig. 4 (at the top) shows relative speed up for data sets having 10000 objects, 10 % density of 1's in the data table and various counts of attributes. Fig. 4 (at the bottom) shows relative speed up for data sets having 1000 objects, 100 attributes, and various densities of 1's in the data tables. The 1's in both data sets spread approximately normally among the attributes. Note that each graph contains a certain point from which the increasing number of processors does not allow to take advantage of more processors and performance of the algorithm may even decline due to the overhead related to the management of multiple threads of execution. However, this is a quite common behavior of parallel algorithms.

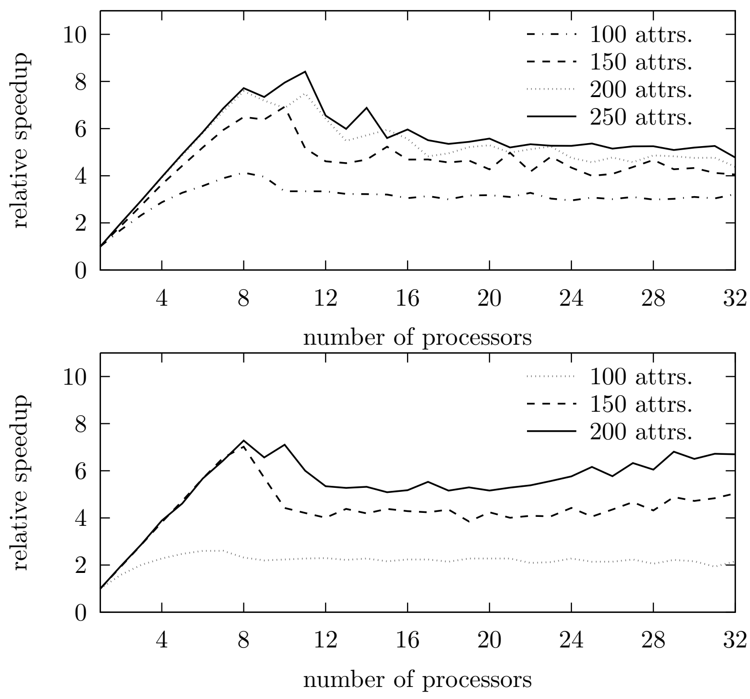

Fig. 4. Scalability of PFCbO

References

1. Andrews S.: In-Close, a Fast Algorithm for Computing Formal Concepts. In: Rudolph, Dau, Kuznetsov (Eds.): *Supplementary Proceedings of ICCS '09*, CEUR WS **483**(2009), 14 pages. `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-483/paper1.pdf`
2. Asuncion A., Newman D.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences, 2007.
3. Berry A., Bordat J.-P., Sigayret A.: A local approach to concept generation. *Annals of Mathematics and Artificial Intelligence*, **49**(2007), 117–136.
4. Fu H., Mephu Nguifo E.: A Parallel Algorithm to Generate Formal Concepts for Large Data. *ICFCA 2004, LNCS* **2961**, pp. 394–401.
5. Ganter B.: *Two basic algorithms in concept analysis.* (Technical Report FB4-Preprint No. 831). TH Darmstadt, 1984.
6. Ganter B., Wille R.: *Formal concept analysis. Mathematical foundations.* Berlin: Springer, 1999.

7. Goldberg L. A.: *Efficient Algorithms for Listing Combinatorial Structures.* Cambridge University Press, 1993.
8. Hettich S., Bay S. D.: The UCI KDD Archive University of California, Irvine, School of Information and Computer Sciences, 1999.
9. Johnson D. S, Yannakakis M., Papadimitriou C. H.: On generating all maximal independent sets. *Information Processing Letters* **27**(3)(1988), 119–123.
10. Krajca P., Outrata J., Vychodil V.: Parallel Recursive Algorithm for FCA. In: Belohlavek, Kuznetsov (Eds.): *Proc. CLA 2008*, CEUR WS **433**(2008), 71–82. `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-433/paper6.pdf`
11. Krajca P., Vychodil V.: Comparison of data structures for computing formal concepts. In: *Proc. MDAI 2009, LNAI* **5861**(2009), 114–125.
12. Kuznetsov S.: Interpretation on graphs and complexity characteristics of a search for specific patterns. *Automatic Documentation and Mathematical Linguistics*, **24**(1)(1989), 37–45.
13. Kuznetsov S.: A fast algorithm for computing all intersections of objects in a finite semi-lattice (Быстрый алгоритм построения всех пересечений объектов из конечной полурешетки, in Russian). *Automatic Documentation and Mathematical Linguistics*, **27**(5)(1993), 11–21.
14. Kuznetsov S.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. *PKDD 1999*, pp. 384–391.
15. Kuznetsov S., Obiedkov S.: Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Int.*, **14**(2002), 189–216.
16. Lindig C.: Fast concept analysis. *Working with Conceptual Structures– –Contributions to ICCS 2000,* pp. 152–161, 2000. Aachen: Shaker Verlag.
17. Outrata J., Vychodil V.: Fast algorithm for computing fixpoints of galois connections induced by object-attribute relational data (in preparation).