Making Use of Empty Intersections to Improve the Performance of CbO-Type Algorithms

ANDREWS, Simon <http://orcid.org/0000-0003-2094-7456>

Published version

ANDREWS, Simon (2017). Making Use of Empty Intersections to Improve the Performance of CbO-Type Algorithms. In: Formal Concept Analysis : 14th International Conference, ICFCA 2017, Rennes, France, June 13-16, 2017, Proceedings. Lecture notes in artificial intelligence (10308). Springer, 56-71.

Making use of empty intersections to improve the performance of CbO-type algorithms

Simon Andrews (ORCID 0000-0003-2094-7456)

Conceptual Structures Research Group
Communication and Computing Research Centre
Department of Computing
Faculty of Arts, Computing, Engineering and Sciences
Sheffield Hallam University, Sheffield, UK
s.andrews@shu.ac.uk

Abstract. This paper describes how improvements in the performance of Close-by-One type algorithms can be achieved by making use of empty intersections in the computation of formal concepts. During the computation, if the intersection between the current concept extent and the next attribute-extent is empty, this fact can be simply inherited by subsequent children of the current concept. Thus subsequent intersections with the same attribute-extent can be skipped. Because these intersections require the testing of each object in the current extent, significant time savings can be made by avoiding them. The paper also shows how further time savings can be made by forgoing the traditional canonicity test for new extents, if the intersection is empty. Finally, the paper describes how, because of typical optimizations made in the implementation of CbO-type algorithms, even more time can be saved by amalgamating inherited attributes with inherited empty intersections into a single, simple test.

Keywords: Formal Concept Analysis · FCA · FCA algorithms · Computing formal concepts · Canonicity test · Close-by-One · CbO · In-Close

1 Introduction

There have been many advances in Formal Concept Analysis that address complexity, focus the analysis and concentrate on producing manageable and meaningful results. Such approaches include ice-berg lattices [1], nesting lattices [2], creating sub-contexts [3], fault-tolerance [4, 5], expandable concept trees [6], rough concepts [7] and approximation [8]. However, there will always be the need for faster performance of the fundamental operations in FCA, such as the computation of formal concepts. Clearly, we need to avoid the production of concept lattices that are indecipherable, 'bird's nests', containing thousands of nodes and FCA results that are over-complex and meaningless. Nevertheless, with FCA being applied ever more to the analysis of large sets of data, performance is an important factor. Even if the set of concepts is small, dynamic applications need

fast computational engines to cope with fast changes in the underlying data. Or it may be the case that although the set of computed concepts is large, applications may use this as a data-set of concepts on which to carry out queries and subsequent analysis. Two recent major European projects provide real-world examples involving large and dynamic data bases: The ATHENA project - using social media in crisis management [9, 10] and the ePOOLICE project - scanning the electronic environment to detect organised crime [11]. Both projects involved the development of software systems where FCA was implemented as a component. It was found that the most practical means of applying FCA was to compute very large sets of concepts on the fly, and then query these sets for concepts of interest, rather than create pre-processing components to form sub-contexts from the original data set. Both projects involved creating large sets of structured data from the text-processing of Internet-based unstructured data, dynamically and on the fly. Having a fast concept mining engine to convert this data quickly into formal concepts before querying and visualization was essential to the operation of the systems.

In the development of such fast algorithms, the discovery of the so-called 'canonicity test', whereby the attributes in a concept could be examined to determine its newness in the computation [12], has proved to be fundamental. This test has given rise to a number of algorithms based on the original Close-by-One (CbO) algorithm [12] including the CbO algorithm presented in [13], FCbO [14, 15] and In-Close2 [16]. FCbO introduced a combined 'breadth and depth' approach to computation that allowed child concepts to fully inherit their parent's attributes. In-Close2 then added a modified, 'partial-closure', canonicity test to reduce the computation required in the test. FCbO also introduced a technique whereby failed canonicity tests could be inherited, thereby avoiding repeated tests.

This paper describes new advances in the performance of CbO-type algorithms by making use of empty intersections. During the computation, if the intersection between the current concept extent and the next attribute-extent is empty, this fact can be simply inherited by subsequent children of the current concept. Because all extents D that are children of a current extent C are sub-sets of C, it is simple to show that intersections of child extents with an attribute-extent $\{j\}^{\downarrow}$ will be empty if the intersection between the same attribute-extent and the parent extent was empty:

$if\ D \subseteq C\ and\ C \cap \{j\}^{\downarrow} = \emptyset\ then\ D \cap \{j\}^{\downarrow} = \emptyset$

Thus subsequent intersections between child extents and the same attribute-extent can be skipped. Because these intersections require the testing of each object in the current extent, significant time savings can be made by avoiding them.

The rest of this paper is structured as follows: The paper will use the algorithm In-Close2 [16] as its starting point, so section 2 is a recap of that algorithm. Section 3 is a recap of the FCbO algorithm. As a fast CbO-type algorithm for computing formal concepts it is a good benchmark for the In-Close variants. Section 4 describes a new In-Close variant, In-Close4a, that incorporates the

inheritance of empty intersections as described above. It should be noted that In-Close1, In-Close2 and In-Close3 are the existing versions of In-Close, as presented in [17]. Thus the new versions presented here are named In-Close4a and In-Close4b. In-Close3 is a version that incorporated FCbO's feature of inheriting failed canonicity tests, but, because of the added complexity and overheads incurred, actually performed less well overall than In-Close2. Thus In-Close2, rather than In-Close3, is used here as the starting point for the new versions. Section 5 describes a refined version of In-Close4a, called In-Close4b, that forgoes the traditional canonicity test for new extents, if the intersection between the current extent and the next attribute-extent is empty. Section 6 describes some key optimizations made in the implementation of CbO-type algorithms, and how further time savings can be made by amalgamating inherited attributes with inherited empty intersections into a single, simple test. Section 7 presents a series of experiments and results, comparing the performance of In-Close2, In-Close4a, In-Close4b and FCbO. Finally, section 8 provides some concluding remarks and ideas for further work.

2 Recap of the In-Close2 algorithm

Below is a recap of the In-Close2 algorithm, as presented in [17]. In-Close2 was been 'bred' from In-Close and FCbO to combine the efficiencies of a partial-closure canonicity test (as compared to a full closure test) with full inheritance of the parent intent. The full inheritance is achieved by adapting the combined breadth-first and depth-first approach of FCbO [14, 15]. The main cycle is completed before passing to the next level, so that all the attributes of a parent intent can be passed down to the next level rather than just some of them. Like In-Close, child intents only have to be 'finished off' by adding attributes when $A = C$, but now additional attributes after j are also inherited and can be skipped. During the main cycle, whilst the current intent is being closed, new extents that pass the canonicity test are stored in a queue, similar to the queue in FCbO, to be processed after the main cycle has completed.

The In-Close2 algorithm is invoked with an initial pair $(A, B) = (X, \emptyset)$, where A is a set of objects (extent) and B is a set of attributes (intent) and X is the set of all objects in the formal context, and initial attribute $y = 0$. Y is the set of all attributes in the formal context and Y_j is the set of all attributes up to (but not including) j.

Line 1 - Iterate across the formal context, from a starting attribute y up to attribute $n - 1$, where n is the number of attributes in the context.

Line 2 - Skip attributes already in B. Because intents now inherit all of their parent's attributes, these can be skipped.

Line 3 - Form an extent, C, by intersecting the current extent, A, with the next column of objects in the context.

Line 4 - If the extent formed, C, equals the extent, A, of the concept whose intent is currently being closed, then...

Line 5 - ...add the current attribute, j, to the intent being closed, B.

In-Close2

`ComputeConceptsFrom`$((A, B), y)$

```
1  for j ← y upto n − 1 do
2      if j ∉ B then
3          C ← A ∩ {j}↓
4          if C = A then
5              B ← B ∪ {j}
6          else
7              if B ∩ Y_j = C↑j then
8                  PutInQueue(C, j)
9  ProcessConcept((A, B))
10 while GetFromQueue(C, j) do
11     D ← B ∪ {j}
12     ComputeConceptsFrom((C, D), j + 1)
```

Line 7 - Otherwise, test the canonicity using the partial-closure canonicity test [18]: $^{\uparrow}$ is the standard closure operator in FCA and $^{\uparrow_j}$ is a modification meaning "close up to, but not including attribute j".

Line 8 - If the canonicity test is passed, place the new extent, C, and the location where it was found, j, in a queue for later processing.

Line 9 - Pass concept (A, B) to the notional procedure `ProcessConcept` to process it in some way (for example, storing it in a set of concepts).

Lines 10 - The queue is processed by obtaining each new extent and associated location from the queue.

Line 11 - Each new partial intent, D, inherits all the attributes from its completed parent intent, B, along with the attribute, j, where its extent was found.

Line 12 - Call `ComputeConceptsFrom` to compute child concepts from $j + 1$ and to complete the intent D.

3 Recap of the FCbO Algorithm

Below is a recap of the FCbO algorithm [14, 15] as presented in [17]. FCbO introduced the feature of inherited canonicity test failures to improve the performance of CbO-type algorithms, along with the combined breadth/depth first approach to enable full inheritance of parent intents. The inheritance of test failure is achieved by recording intents that are not canonical as N^js, where j is the current attribute, thus enabling subsequent levels to test these failed intents against the current one and thus avoid the computation of a redundant extent and intent. FCbO is invoked with the initial concept $(A, B) = (X, X^{\uparrow})$, initial attribute $y = 0$ and a set of empty Ns, $\{N^y = \emptyset \mid y \in Y\}$.

FCbO

`ComputeConceptsFrom`$((A,B), y, \{N^y \mid y \in Y\})$

```
1  ProcessConcept((A, B))
2  for j ← y upto n − 1 do
3      M^j ← N^j
4      if j ∉ B and N^j ∩ Y_j ⊆ B ∩ Y_j then
5          C ← A ∩ {j}^↓
6          D ← C^↑
7          if B ∩ Y_j = D ∩ Y_j then
8              PutInQueue ((C, D), j)
9          else
10             M^j ← D
11 while GetFromQueue((C, D), j) do
12     ComputeConceptsFrom((C, D), j + 1, {M^y | y ∈ Y})
```

Line 1 - Pass concept (A, B) to the notional procedure `ProcessConcept` to process it in some way (for example, storing it in a set of concepts).

Line 2 - Iterate across the context, from starting attribute y up to attribute $n-1$.

Line 3 - M^j is set to the latest intent that failed the canonicity test at attribute j, N^j.

Line 4 - Skip attributes in B and those that have an inherited record of failure.

Line 5 - Otherwise, form an extent, C, by intersecting the current extent, A, with the next column of objects in the context.

Line 6 - Close the extent to form an intent, D.

Line 7 - Perform the canonicity test.

Line 8 - If the concept is a new one, store it in a queue along with the attribute it was computed at.

Line 10 - Otherwise set the record of failure for attribute j, M^j, to the intent that failed the canonicity test.

Line 11 - Get each stored concept from the queue...

Line 12 - ...and pass it to the next level, along with the stored starting attribute for the next level and the failed intents from this level.

4 New algorithm, In-Close4a: skipping attributes with inherited empty intersections

In-Close4a adds a new feature to In-Close2 by enabling subsequent levels to skip attributes that have previously resulted in an empty intersection with a parent extent. If a parent extent intersected with an attribute-extent results

in an empty set, then any subset (child) of the parent extent intersected with the same attribute-extent will also result in an empty set. Thus, if a record is kept of the parent empty intersections, subsequent children can skip the attributes concerned. Because the intersection between the current extent and the attribute-extent must iterate all of the objects in the current extent, a significant number of operations can potentially be avoided if intersections can be skipped.

The new algorithm, In-Close4a, incorporating this feature is given below. In-Close4a is invoked with an initial pair $(A, B) = (X, \emptyset)$, an initial attribute $y = 0$ and an empty set of attributes, $P = \emptyset$.

In-Close4a

`ComputeConceptsFrom`$((A, B), y, P)$

```
1  for j ← y upto n − 1 do
2      if j ∉ B and j ∉ P then
3          C ← A ∩ {j}↓
4          if C = A then
5              B ← B ∪ {j}
6          else
7              if C = ∅ then
8                  P ← P ∪ {j}
9              if B ∩ Y_j = C^↑j then
10                 PutInQueue(C, j)
11 ProcessConcept((A, B))
12 Q ← P
13 while GetFromQueue(C, j) do
14     D ← B ∪ {j}
15     ComputeConceptsFrom((C, D), j + 1, Q)
```

The changes in the algorithm from In-Close2 are as follows:

Line 2 - Skip attributes already in B and also skip attributes in P, as we know they will result in empty intersections.

Line 7 - If the new extent, C, is empty...

Line 8 - ... add the current attribute to P.

Line 12 - Store P in Q ready to pass the attributes resulting in empty intersections to the next level.

Note that the algorithm will compute the concept $(\emptyset, Y)$, if it exists. If $C = \emptyset$, then the canonicity test is still carried out, and, if it is the first occurrence of the empty set of objects, it will pass the canonicity test and the concept will be completed at the next level.

5 New algorithm, In-Close4b: forgoing the canonicity test after empty intersections

The concept $(\emptyset, Y)$ can be viewed as a special case and will exist if there are no objects that have all the attributes in the formal context. If it exists it will be computed by CbO-type algorithms at the first occurrence of $C = \emptyset$, with subsequent empty extents failing the canonicity test (although FCbO will avoid some of these tests via inherited test-failure). However, as an alternative approach, In-Close4b forgoes the canonicity test when $C = \emptyset$, in effect making it a case of automatic failure. Thus, as a refinement of In-Close4b, the new algorithm not only skips inherited empty intersections but also avoids having to carry out a canonicity test whenever an empty intersection occurs. It is quite likely for the empty intersection to occur very frequently in the computation, so there is thus the potential to save a significant number of operations.

Of course, the resulting algorithm is incomplete, in that it will not compute the concept $(\emptyset, Y)$. However, it is a simple task to add it afterwards, if it exists: If $Y^{\downarrow} = \emptyset$ then add $(\emptyset, Y)$ to the set of computed concepts.

The new algorithm, In-Close4b, incorporating this feature is given below. In-Close4b is invoked with an initial pair $(A, B) = (X, \emptyset)$, an initial attribute $y = 0$ and an empty set of attributes, $P = \emptyset$.

In-Close4b

`ComputeConceptsFrom`$((A, B), y, P)$

```
1  for j ← y upto n − 1 do
2      if j ∉ B and j ∉ P then
3          C ← A ∩ {j}↓
4          if C ≠ ∅ then
5              if C = A then
6                  B ← B ∪ {j}
7              else
8                  if B ∩ Y_j = C^↑j then
9                      PutInQueue(C, j)
10         else
11             P ← P ∪ {j}
12 ProcessConcept((A, B))
13 Q ← P
14 while GetFromQueue(C, j) do
15     D ← B ∪ {j}
16     ComputeConceptsFrom((C, D), j + 1, Q)
```

It should be noted that forgoing the canonicity test following an empty intersection was actually present in the very first incarnation of the In-Close algo-

rithm [19] (although not in later versions). However, it was not realised as a novel time-saving device at the time and thus was not explicitly described as such. Its effects were not explicitly measured or compared, nor were its implications considered, as far as incompleteness is concerned. Consequently, In-Close4b could be regarded as a new 'best of breed' algorithm, combing time-saving features of In-Close, In-Close2, In-Close4a and FCbO.

An implementation of In-Close4b is available, open-source and free to download from Sourceforge[1].

6 Optimisation

In this section, three key optimisations will be briefly described. The first two are recapping existing major optimisations: the use of bit-arrays and the implementation of the partial-closure canonicity test, and the third explains how the inheritance of attributes and the inheritance of empty intersections can be amalgamated into a single test. The experiments in Section 7 use optimised implementations, so it is worth understanding the key optimisations involved.

Recap of the use of bit-arrays. Implementations of CbO-type algorithms, such as In-Close and FCbO, typically use a bit-array to represent the formal context. This allows operations on the formal context, such as closure operations, to be implemented using bit-wise operators in the manner of fine-grained parallel processing. In a typical 64-bit architecture, this means that 64 cells of the formal context can be operated on simultaneously.

Recap of the implementation of the partial-closure canonicity test. In practice, it is not necessary to always close the new extent up to the current attribute. It is only necessary to find the *first instance* where $B \cap Y_j$ and $C^{\uparrow_j}$ do not agree. Thus failure is typically detected before j is reached, making even more time savings than the partial-closure would. By comparison, in other algorithms, such as FCbO, a *full-closure* is *always* required because, not only is it the the basis of the canonicity test, it also provides the closure of new concepts. In In-Close, with the partial-closure canonicity test, new concepts are closed incrementally whenever $C = A$ by $B \leftarrow B \cup \{j\}$ (lines 4 and 5 of In-Close4a, for example). Furthermore, given that the test $C = A$ is provided 'for free', as a by-product of the intersection in $C \leftarrow A \cap \{j\}^{\downarrow}$, the overheads of this incremental closure are insignificant.

Amalgamation of the inheritance of attributes and the inheritance of empty intersections. In typical implementations of fast CbO-type algorithms, the intent, B, of a concept is stored twice: once as a local version, in Boolean (bit) form, with each bit representing the presence or absence of an attribute, and once

[1] https://sourceforge.net/projects/inclose/

as integers (the index numbers of the attributes) in a tree data structure. The latter is space-saving and thus used for storage and output, whilst the former is used as a fast method of determining $j \notin B$ to skip inherited attributes. Thus, because the Boolean form is not required for subsequent storage and output, it can be used to record the inherited attribute *and* the empty intersections - both are tested, recorded and inherited in the same bit-array. The addition of the inheritance of empty intersections does not, therefore, require an additional array for P and incurs very few overheads in the implementation.

7 Evaluation of performance

In this section, In-Close2, In-Close4a, In-Close4b and FCbO will be evaluated by comparing their performance over a varied range of data sets. The experiments are divided into three groups: 1) real data sets, 2) artificial data sets, and 3) randomised data sets. Following the performance comparison, the algorithms are investigated to compare the number of canonicity tests carried out in each case, as this is the most labour intensive operation involved. Indeed, the intention of features such as skipping attributes and inheriting canonicity failures are specifically designed to reduce the number of canonicity test required.

Optimised implementations of the algorithms were created using C++ and the experiments were conducted on a standard 64-bit Intel architecture, using a PC with an Intel Core i7-2600 3.40GHz CPU and 8GB of RAM. To cater for any inconsistency of system performance, each experiment was conducted multiple times and the average time taken for each.

Real data set experiments. Four real data sets were used in the experiments: *Mushroom*, *Adult* and *Internet Ads*, taken from the UCI Machine Learning Repository [20] and *Student*, an anonymised data set from an internal student experience survey carried out at Sheffield Hallam University, UK. The data sets were selected to represent a broad range of features, in terms of size and density, and the UCI ones, in particular, are well known and used in FCA work.

The results of the experiments are given in Table 1 (timings) and Table 2 (canonicity tests). In all timing cases (apart from two ties) In-Close4b was fastest, followed by In-Close4a then In-Close2, and FCbO was the slowest.

The smallest number of canonicity tests was achieved by FCbO for *Mushroom* and *Student* and by In-Close4b for *Adult* and *Internet Ads*. It is clear, from comparing In-Close2 results with those of In-Close4a and 4b that making use of empty intersections has a large effect in reducing the number of canonicity tests required. However, it is also clear that the inherited canonicity failures in FCbO also produce a large reduction. Comparing the timings with the number of canonicity tests carried out (particularly between In-Close2 and FCbO) gives clear evidence of the significance of the partial-closure canonicity test in reducing computation time. Of course, it should be noted that making use of empty intersections prevents canonicity test involving the closure of the empty set of objects. Such closures are obviously less labour intensive than closing non-empty

sets of object, but nevertheless a significant number of operations are still necessary, and the timing results clearly show that making use of empty intersections shows a small but significant speed-up for each of the data sets.

Table 1. Real data set results (timings in seconds).

Data set	Mushroom	Adult	Internet Ads	Student
$\|G\| \times \|M\|$	$8,124 \times 125$	$32,561 \times 124$	$3,279 \times 1,565$	587×145
Density	17.36%	11.29%	0.97%	24.50%
#Concepts	226,921	1,388,469	16,570	$22,760,243$
FCbO	0.21	1.46	0.31	8.80
In-Close2	0.21	1.05	0.12	5.06
In-Close4a	0.19	0.95	0.08	**4.65**
In-Close4b	**0.17**	**0.88**	**0.07**	**4.65**

Table 2. Real data set results (canonicity tests).

Data set	Mushroom	Adult	Internet Ads	Student
In-Close2	1,164,829	2,614,810	1,867,689	55,038,419
In-Close4a	447,356	1,916,356	682,635	53,859,754
FCbO	**325,986**	2,029,933	363,568	**40,630,663**
In-Close4b	405,584	**1,707,707**	**91,029**	53,162,649

Artificial data set experiments. Artificial data sets were used that, although randomised, the randomisation was constrained by properties of real data sets, such as many-valued attributes having a fixed number of possible values. Three data sets, *M7X10G120K*, *M10X30G120K* and *T10I4D100K*, were used to provide a range of features in terms of size and density.

The timing results of the artificial data set experiments are given in Table 3 and the comparison of the number of canonicity tests carried out is given in Table 4. The timing results reiterate those obtained in the real data set experiments: In-Close4b fastest, then In-Close4a, then In-Close2, then FCbO. In terms of the number of canonicity tests carried out, In-Close4b had the fewest, then FCbO, then In-Close4a, then In-Close2.

Random data set experiments. Three series of random data experiments were carried out, testing the effect of changes in the number of attributes, context density, and number of objects:

Table 3. Artificial data set results (timings in seconds).

Data set	M7X10G120K	M10X30G120K	T10I4D100K
$\|G\| \times \|M\|$	$120,000 \times 70$	$120,000 \times 300$	$100,000 \times 1,000$
Density	10.00%	3.33%	1.01%
#Concepts	1,166,326	4,570,493	2,347,376
FCbO	1.35	15.45	23.83
In-Close2	0.98	8.37	9.10
In-Close4a	0.81	7.45	7.79
In-Close4b	**0.77**	**5.60**	**6.56**

Table 4. Artificial data set results (canonicity tests).

Data set	M7X10G120K	M10X30G120K	T10I4D100K
In-Close2	14,621,925	330,546.826	185,296,387
In-Close4a	5,548,392	191,601,668	112,154,256
FCbO	4,640,906	167,814,522	75,281,105
In-Close4b	**2,360,015**	**29,686,007**	**21,262,544**

- Figure 1: *Attributes series* - with 5% density and 5,000 objects, the number of attributes was varied between 300 and 1,000. The number of concepts varied from approximately 1,000,000 to 22,000,000.
- Figure 3: *Objects series* - with 5% density and 200 attributes, the number of objects was varied between 30,000 and 100,000. The number of concepts varied from approximately 4,000,000 to 22,000,000.
- Figure 2: *Density series* - with 200 attributes and 10,000 objects, the density of 1s in the context was varied between 3 and 10%. The number of concepts varied from approximately 200,000 to 19,000,000.

The results of the random data set timings are given in Figure 1 (attribute series), Figure 2 (density series) and 3. A comparison of the number of canonicity tests carried out for the random data sets in each algorithm is given in Figure 4 (attribute series), Figure 5 (density series) and 6.

Again, the results confirm those from the previous experiments, with In-Close4b showing the best performance, then In-Close4a, then In-Close2, then FCbO. In terms of the number of canonicity tests carried out, overall FCbO showed the greatest reduction, although In-Close4b was a close second and, for lower density data sets in the density series, In-Close4b had greatest reduction.

8 Conclusions

In conclusion, the experimental results clearly show that making use of empty intersections, by skipping inherited empty intersections (In-Close4a) and forgoing the canonicity test following an empty intersection (In-Close4b), provides a

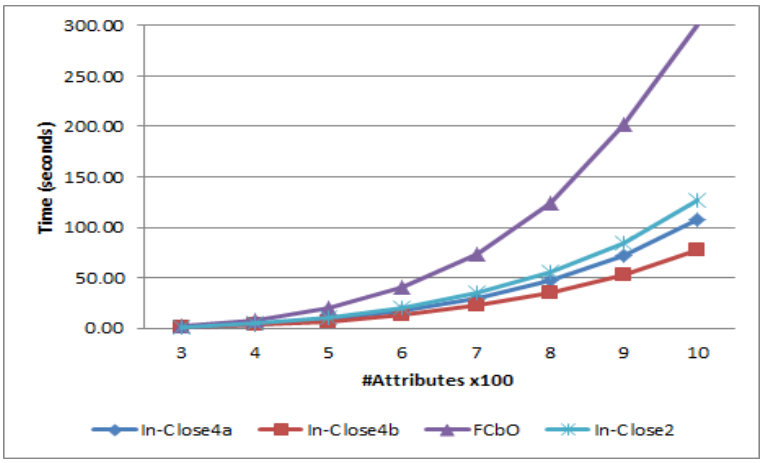

Fig. 1. Comparison of performance with varying number of attributes. 5% density, 5,000 objects. #Concepts range from approx. 1,000,000-22,000,000

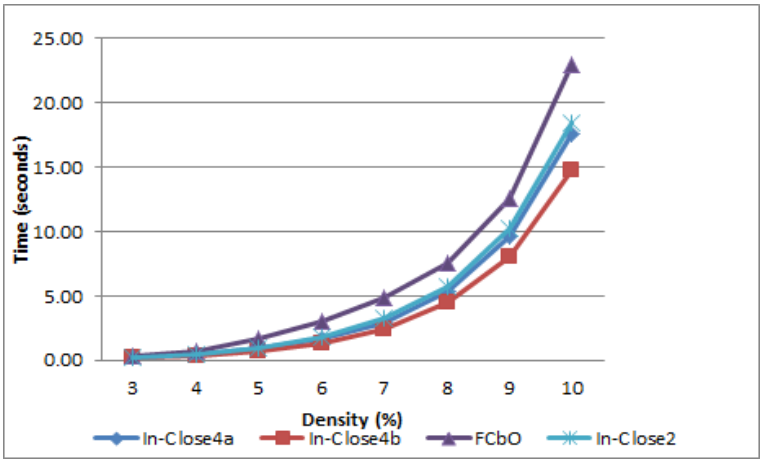

Fig. 2. Comparison of performance with varying context density. 200 attributes, 10,000 objects. #Concepts range from approx. 4,000,000-22,000,000

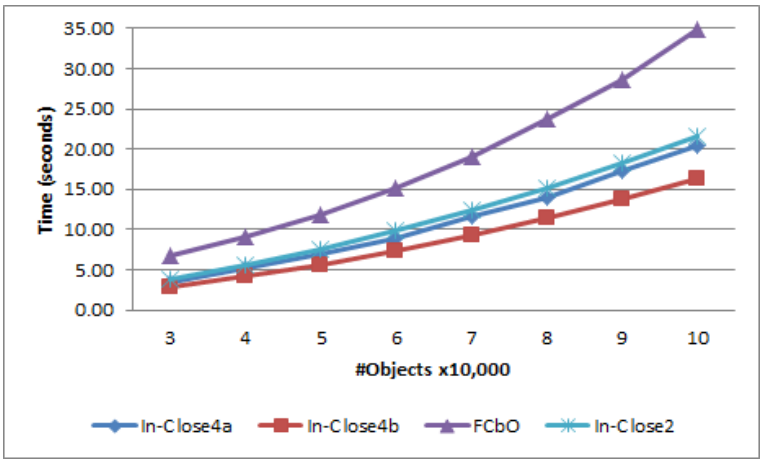

Fig. 3. Comparison of performance with varying number of objects. 5% density, 200 attributes. #Concepts range from approx. 200,000-19,000,000

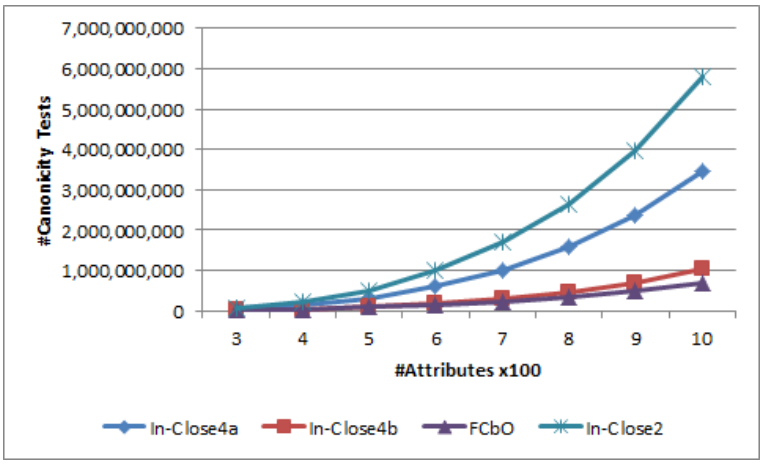

Fig. 4. Comparison of the number of canonicity tests carried out with varying number of attributes. 5% density, 5,000 objects. #Concepts range from approx. 1,000,000-22,000,000

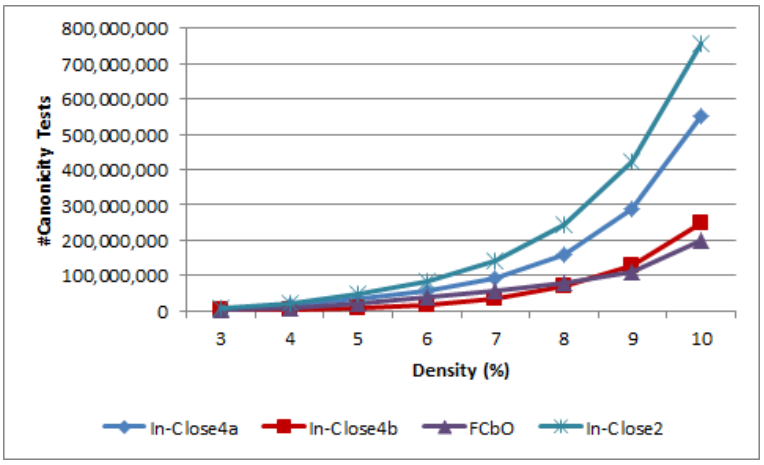

Fig. 5. Comparison of the number of canonicity tests carried out with varying context density. 200 attributes, 10,000 objects. #Concepts range from approx. 4,000,000-22,000,000

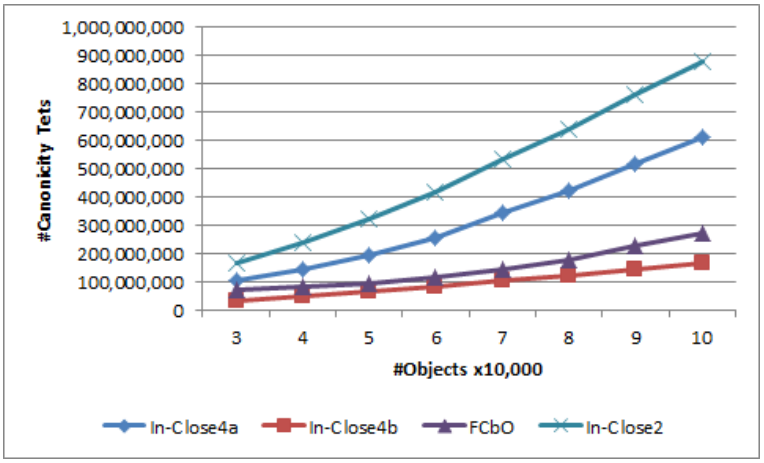

Fig. 6. Comparison of the number of canonicity tests carried out with varying number of objects. 5% density, 200 attributes. #Concepts range from approx. 200,000-19,000,000

small but significant improvement to the performance of CbO-type algorithms. Their implementation is simple, costing almost no overhead and easily optimised to amalgamate inherited empty intersections with inherited attributes.

In terms of further work, the canonicity test results for FCbO are interesting. FCbO shows an equal or better reduction in canonicity tests carried out, compared to the other algorithms, through its inheritance of canonicity test failures, but this is outweighed by the costly 'full-closure' canonicty test it employs (as also evidence in previous papers [16–18]). Although previous work [17] has already produced an algorithm (In-Close3) that incorporates FCbO's inheritance of canonicity test failures, it failed to provide any significant improvements in performance compared to In-Close2, due to the complexity and consequent overheads in computation that it required. Nevertheless, the large reduction in canonicity tests carried out in FCbO makes it tempting to re-visit the inheritance of test failures - either to improve and optimise further its implementation, or to investigate the possibility of designing a simpler method of test failure inheritance. If test failures can be inherited without the penalty of significant overheads in computation, further improvements in the performance of CbO-type algorithms will be possible.

References

1. Stumme, G., Taouil, R., Bastide, Y., Lakhal, L.: Conceptual clustering with iceberg concept lattices. Proc. GI-Fachgruppentreffen Maschinelles Lernen (FGML01) (2001)
2. Valtchev, P., Grosser, D., Roume, C., Hacene, M.R.: Galicia: an open platform for lattices. In: A. de Moor B. Ganter, editor, Using Conceptual Structures: Contributions to 11th Intl. Conference on Conceptual Structures. (2003) 241–254
3. Andrews, S., Orphanides, C.: Knowledge discovery through creating formal contexts, IEEE Computer Society (2010) 455–460
4. Pensa, R.G., Boulicaut, J.F.: Towards fault-tolerant formal concept analysis. In Bandini, S., Manzoni, S., eds.: Advances in Artificial Intelligence, 9th Congress of the Italian Association for Artificial Intelligence. Volume 3673 of Lecture Notes in Computer Science., Springer-Verlag Berlin Heidelberg (2005)
5. Dau, F.: An implementation for fault tolerance and experimental results. In: CUBIST Workshop. (2013) 21–30
6. Andrews, S., Hirsch, L.: A tool for creating and visualising formal concept trees. CEUR Workshop Proceedings: Proceedings of the Fifth Conceptual Structures Tools & Interoperability Workshop (CSTIW 2016) **1637** (2016) 1–9
7. Liu, M., Shao, M., Zhang, W., Wu, C.: Reduction method for concept lattices based on rough set theory and its application. Computers & Mathematics with Applications **53** (2007) 1390–1410
8. Gaume, B., Navarro, E., Prade, H.: Clustering bipartite graphs in terms of approximate formal concepts and sub-contexts. International Journal of Computational Intelligence Systems **6** (2013) 1125–1142
9. ATHENA: The European ATHENA Project - use of new smart devices and social media in crisis situations: http://www.projectathena.eu/ (Accessed: September 2016)

10. Andrews, S., Yates, S., Akhgar, B., Fortune, D.: Strategic Intelligence Management: National Security Imperatives and Information and Communication Technologies. In: The ATHENA Project: Using Formal Concept Analysis to Facilitate the Actions of Responders in a Crisis Situation. Elsevier: Butterworth-Heinemann (2013) 167–180
11. Andrews, S., Brewster, B., Day, T.: Organised crime and social media: Detecting and corroborating weak signals of human trafficking online. In: International Conference on Conceptual Structures, Springer (2016) 137–150
12. Kuznetsov, S.O.: Mathematical aspects of concept analysis. Mathematical Science **80** (1996) 1654–1698
13. Krajca, P., Outrata, J., Vychodil, V.: Parallel recursive algorithm for FCA. In Belohavlek, R., Kuznetsov, S., eds.: Proceedings of Concept Lattices and their Applications. (2008)
14. Krajca, P., Vychodil, V., Outrata, J.: Advances in algorithms based on CbO. In Kryszkiewicz, M., Obiedkov, S., eds.: CLA 2010, University of Sevilla (2010) 325–337
15. Outrata, J., Vychodil, V.: Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data. Information Sciences **185** (2012) 114–127
16. Andrews, S.: In-close2, a high performance formal concept miner. In Andrews, S., Polovina, S., Hill, R., Akhgar, B., eds.: Conceptual Structures for Discovering Knowledge - Proceedings of the 19th International Conference on Conceptual Structures (ICCS), Springer (2011) 50–62
17. Andrews, S.: A best-of-breed approach for designing a fast algorithm for computing fixpoints of galois connections. Information Sciences **295** (2015) 633–649
18. Andrews, S.: A partial-closure canonicity test to increase the efficiency of cbo-type algorithms. In: Proceedings of the 21st International Conference on Conceptual Structures, Springer (2014) 37–50
19. Andrews, S.: In-close, a fast algorithm for computing formal concepts. In Rudolph, S., Dau, F., Kuznetsov, S.O., eds.: ICCS 2009. Volume 483 of CEUR WS., http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-483/ (2009)
20. Frank, A., Asuncion, A.: UCI machine learning repository: http://archive.ics.uci.edu/ml (2010)