A New Method for Inheriting Canonicity Test Failures in Close-by-One Type Algorithms

Simon Andrews[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. Close-by-One type algorithms are efficient algorithms for computing formal concepts. They use a mathematical canonicity test to avoid the repeated computation of the same concept, which is far more efficient than methods based on searching. Nevertheless, the canonicity test is still the most labour intensive part of Close-by-One algorithms and various means of avoiding the test have been devised, including the ability to inherit test failures at the next level of recursion. This paper presents a new method for inheriting canonicity test failures in Close-by-One type algorithms. The new method is simpler than the existing method and can be amalgamated with other algorithm features to further improve efficiency. The paper recaps an existing algorithm that does not feature test failure inheritance and an algorithm that features the existing method. The paper then presents the new method and a new algorithm that incorporates it. The three algorithms are implemented on a 'level playing field' with the same level of optimisation. Experiments are carried out on the implemented algorithms, using a representative range of data sets, to compare the number of inherited canonicity test failures and the computation times. It is shown that the new algorithm, incorporating the new method of inheriting canonicity test failures, gives the best performance.

Keywords: Formal Concept Analysis · FCA · FCA algorithms · Computing formal concepts · Canonicity test · Inheriting canonicity test failures· Close-by-One · FCbO · In-Close

1 Introduction

In the development o fast algorithms to compute formal concepts, the discovery of the so-called 'canonicity test', whereby the attributes in a concept could be examined to determine its newness in the computation, gave rise to the original Close-by-One (CbO) algorithm [8]. The canonicity test has proved to be fundamental in the efficient computation of formal concepts, being far more efficient than previous methods of determining the newness of a concept based on

searching, and was integral to the subsequent CbO algorithm presented in [6]. Nevertheless, the canonicity test is still the most labour intensive part of CbO-type algorithms and various means of avoiding or improving the test have been devised, giving rise to a number of advances in CbO-type algorithms including FCbO [7, 9], In-Close2 [3] and In-Close4 [4]. 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 many canonicity tests. In-Close4 made use of empty intersections between the current concept extent and attribute-extents in the computation to also avoid canonicity tests.

This paper describes a new method of inheriting failed canonicity tests that is simpler than the method used by FCbO. Furthermore, the method can be amalgamated with existing efficiency features to further improve performance.

The rest of this paper is structured as follows: The paper will use the algorithm In-Close4 [4] as the framework in which to incorporate the new inheritance method, so Section 2 is a recap of that algorithm. Section 3 is a recap of the FCbO algorithm, describing its method of inheriting failed canonicity tests. Section 4 describes the new method of inheriting failed canonicity tests and incorporates it into In-Close4, creating a new algorithm, In-Close5. It should be noted that In-Close1, In-Close2 and In-Close3 are previous versions of In-Close, as presented in [2]. Section 5 describes the implementation of In-Close4, FCbO and In-Close5 on a 'level playing field' using the same programming optimisations. Section 5 also shows how the new method of inheriting failed canonicity tests can be amalgamated with existing efficiency features to further improve performance. Section 6 presents a series of experiments and results, comparing the performance of In-Close4, FCbO and In-Close5. Finally, Section 7 provides some concluding remarks and ideas for further work.

2 Recap of the In-Close4 Algorithm

Below is a recap of the In-Close4 algorithm, as presented in [4]. In-Close4 combines the efficiency of a partial-closure canonicity test [2] with full inheritance of the parent intent. The full inheritance is achieved by adapting and incorporating the combined breadth-first and depth-first approach of FCbO [7, 9]. 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. Child intents only have to be finished off by adding attributes that are not in the parent intent. 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. In-Close4 also makes use of empty intersections when the current extent is intersected with the next attribute-extent (next column) in the formal context: empty intersections are inherited so

that they can be skipped at subsequent levels in the computation and, whenever an empty intersection occurs, the algorithm forgoes the canonicity test.

The In-Close4 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. The algorithm is also invoked with an empty set of attributes, $P = \emptyset$, in which to store subsequent empty intersections.

Note that forgoing the canonicity test after an empty intersection means that the algorithm is incomplete, in that it will not compute the concept $(Y, \emptyset)$. 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.

In-Close4

`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)
```

A line by line explanation of In-Close4 is as follows:

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 inherit all of their parent's attributes, these can be skipped. Also skip any attributes in P as these are inherited empty intersections - if the parent extent resulted in an empty intersection, so will its children since they are all subsets of the parent.

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

Line 4 - If the extent, C, is not empty...

Line 5 - If the extent, C, equals the extent of the concept whose intent is currently being closed, A, then...

Line 6 - ...add the current attribute, j, to the intent being closed, B.

Line 7 - Otherwise, test the canonicity using the partial-closure canonicity test [1]: $^{\uparrow}$ 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...

Line 9 - ...place the new extent, C, and the location where it was found, j, in a queue for later processing.

Line 10 - If the extent, C, is empty...

Line 11 - ... add the current attribute to P so that the empty intersection can be inherited.

Line 12 - Pass concept (A, B) to the notional procedure `ProcessConcept` to process it in some way (for example, storing it in a data base of concepts).

Line 13 - Store P in Q ready to pass the attributes resulting in empty intersections to the next level.

Line 14 - The queue is processed by obtaining from the queue each new extent and the location it was found.

Line 15 - 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 16 - Recursively 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 [7, 9] as presented in [2]. 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 failures is achieved by recording intents that are not canonical as N^js, where j is the current attribute, thus enabling subsequent levels to compare these failed intents against the current one and thus avoid the computation of a repeated concept without the need for the original canonicity test. FCbO is invoked with the initial concept $(A, B) = (X, X^{\uparrow})$, initial attribute $y = 0$ and a set of empty N^ys, $\{N^y = \emptyset \mid y \in Y\}$.

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.

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 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 A New Method of Inheriting Failed Canonicity Tests

The method of inheriting failed canonicity tests employed by FCbO requires the manipulation and storage of a two-dimensional array to represent intents that fail the canonicity test. A total of n intents are required, and, although the use of pointers in a optimised implementation avoids the need for copying intents, they still need to be computed and stored. This results in computational overheads so that, even though a significant number of canonicity test are avoided [9], algorithms such as In-Close4 are still able to outperform FCbO [2, 4].

However, it is possible to obtain the inheritance of failed canonicity tests with a simpler method. Firstly consider the criteria for failure in In-Close4: the test will fail if there exists an attribute in $C^{\uparrow_j}$ that is not in $B \cap Y_j$. In other words, when there is an attribute before j (but not in the current intent, B) who's attribute-extent contains the extent, C - in which case the extent, C,

will already have been computed. Now consider the starting attribute, y, for the current cycle (Line 1 of In-Close4). Let us say, in a failed canonicity test, that the smallest attribute in $C^{\uparrow_j}$ that is not in $B \cap Y_j$ is i. If $i \geq y$ then an extent, H, where $C \subseteq H$, will have been discovered in the current cycle at i (and be waiting in the current queue). And there may be other extents, discovered after i but before j that are also supersets of C and also in the queue. Thus, if $i \geq y$, the current attribute, j, will be required at the next level to be examined by the children in the queue: C may be canonical with respect to one of the children or j may be an attribute in the intent of a child and thus required to be added. However, if $i < y$, the concept with extent C and its children will have already been computed and processed. Thus no children in the current queue, or subsequent children, need examine j. In other words, if $i < y$ then j can be inherited as a canonicity test failure - all subsequent children can skip j in the cycle. All that is required is to maintain a set of such attributes that can be passed down to the next level in the algorithm.

The new algorithm, In-Close5, below, is In-Close4 with the new method of inheriting failed canonicity tests added. It is invoked in the same way as In-Close4 but with the addition of an initially empty set of attributes, $N = \emptyset$, in which to store canonicity test failures.

In-Close5

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

```
1  for j ← y upto n − 1 do
2      if j ∉ B and j ∉ P and j ∉ N 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                     if min(C^↑j) < y then
12                         N ← N ∪ {j}
13         else
14             P ← P ∪ {j}
15 ProcessConcept((A, B))
16 Q ← P
17 M ← N
18 while GetFromQueue(C, j) do
19     D ← B ∪ {j}
20     ComputeConceptsFrom((C, D), j + 1, Q, M)
```

The new lines in In-Close5 are as follows:

Line 2 - As well as skipping inherited attributes in the intent, $j \notin B$, and inherited empty intersections, $j \notin P$, the algorithm now also skips inherited canonicity test failures, $j \notin N$.

Line 11 - If the canonicity test (Line 8) is failed, a test is carried out comparing the smallest attribute in $C^{\uparrow_j}$ with y. If the attribute is smaller than y then...

Line 12 - ...j is added to the set of canonicity test failures, N.

Line 17 - Store N in M ready to pass the canonicity test failures to the next level.

5 Implementation

The three algorithms, In-Close4, FCbO and In-Close5, were implemented in ANCII C using the same data structures, data pre-processing and level of optimisation to create a 'level playing field' for comparing their performance. The key optimisations are described below.

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. Using bits to represent cells of the formal context also allows more of the context to be retained in cache memory.

Using a Local Boolean Copy of the Current Intent Typical implementations of CbO-type algorithms maintain a global data structure to store integer representations of concept intents (integers mapping to formal attributes) but, at the same time, also use a Boolean (bit-array) representation of the current intent to facilitate an efficient implementation of the test for inherited attributes, $j \notin B$.

Efficient Implementation of the Partial-Closure Canonicity Test in In-Close Algorithms 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, thus saving additional time. In FCbO, however, a full-closure, $C^{\uparrow}$ is *always* required because, if the test is passed, it provides the closure of the concept intent, or, if the test is failed, it provides the failed intent to be stored in M^j. In In-Close, new concept intents are closed at the next level, during the main cycle, whenever $C = A$ by $B \leftarrow B \cup \{j\}$ (Lines 5 and 6 of In-Close4, for example). Furthermore, given that the test $C = A$ is provided at no computational cost, as a by-product of the intersection in $C \leftarrow A \cap \{j\}^{\downarrow}$, the overheads of the closure

are close to zero. This also means that savings are made by In-Close algorithms when canonicity tests succeed. Here, the partial closure, $C^{\uparrow_j}$ is carried out up to j, compared to the full closure, $C^{\uparrow}$, in FCbO.

Amalgamation of Efficiency Features in In-Close5 In implementation, the set of inherited empty intersections, the set of inherited canonicity test failures and the local, Boolean, copy of the current intent can be amalgamated into a single bit-array, in effect reducing the test in Line 2 of In-Close5, $j \notin B$ And $j \notin P$ And $j \notin N$ to a single test, $j \notin Z$, where $Z = B \cup P \cup N$. Lines 6, 12 and 14 will all become $Z \leftarrow Z \cup \{j\}$, thus updating the same bit-array in the implementation (of course the update of the global set of intents in the implementation, required by Line 6, remains unchanged). Amalgamating the three sets of attributes also means there are overhead savings made from reduced parameter passing.

6 Evaluation of Performance

In this section, In-Close4, FCbO and In-Close5 are 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. In each case, the time taken to compute all formal concepts is measured along with the number of canonicity tests carried out.

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, due to background system processes, for example, 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 [5] 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-Close5 was fastest for the *Mushroom*, *Adult* and *Student* data sets, and equal fastest, with In-Close4, for the *Internet Ads* data set. In-Close5 used the fewest canonicity tests for the *Adult* and *Internet Ads* data sets and was not far behind FCbO for the *Mushroom* and *Student* data sets.

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 pre-defined number of unique

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

Data set	Mushroom	Adult	Internet Ads	Student
$\|G\| \times \|M\|$	$8,124 \times 126$	$32,561 \times 124$	$3,279 \times 1,565$	587×145
Density	17.36%	11.29%	0.97%	24.50%
#Concepts	233,116	1,388,469	16,570	22,760,243
FCbO	0.23	1.46	0.21	8.80
In-Close4	0.19	0.88	**0.07**	4.65
In-Close5	**0.18**	**0.85**	**0.07**	**4.31**

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

Data set	Mushroom	Adult	Internet Ads	Student
FCbO	**331,106**	2,029,933	363,568	**40,630,663**
In-Close4	429,974	1,707,707	91,029	53,162,649
In-Close5	332,449	**1,667,052**	**67,715**	41,048,752

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. For all three data sets, In-Close5 was quickest and performed the fewest canonicity tests.

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-Close4	0.77	5.60	6.56
In-Close5	**0.69**	**5.35**	**5.81**

Random Data Set Experiments. Three series of random data experiments were carried out, testing the effect of variation of the number of attributes, context density, and number of objects, respectively:

– *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.

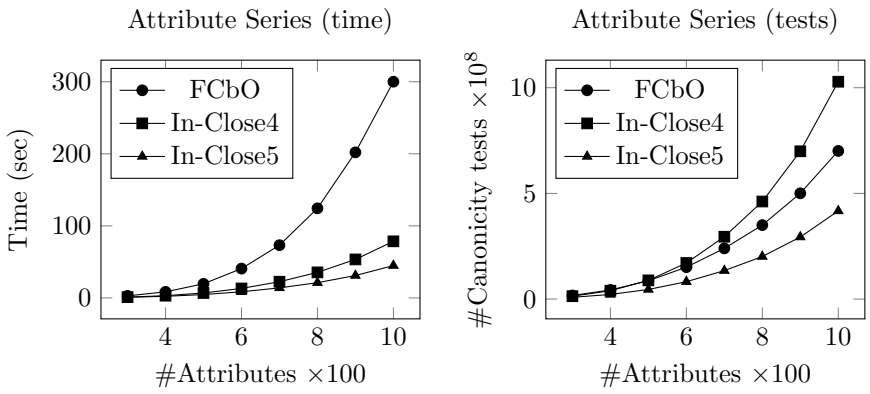
Table 4. Artificial data set results (canonicity tests).

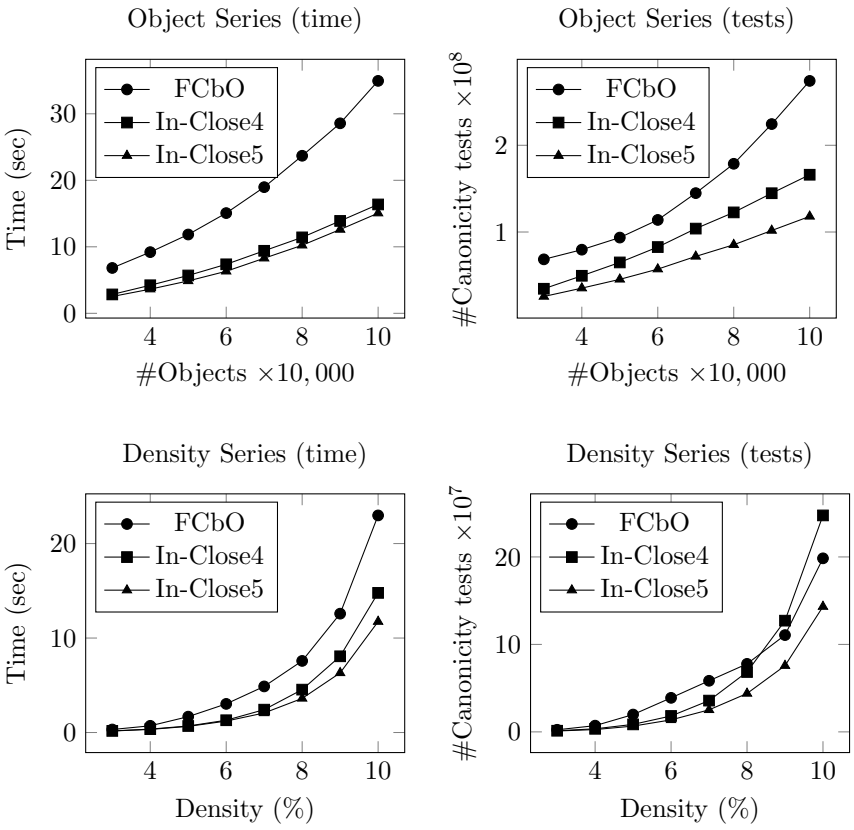
Data set	M7X10G120K	M10X30G120K	T10I4D100K
FCbO	4,640,906	167,814,522	75,281,105
In-Close4	2,360,015	29,686,007	21,262,544
In-Close5	**2,339,951**	**26,593,944**	**14,907,484**

– *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.

– *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 shown in the plots below. In all three series, In-Close5 performed the fewest canonicity tests and was fastest. It is interesting to note that In-Close4 often performed fewer canonicity tests than FCbO (particularly apparent in the *Object* series). One might therefore deduce that the *Object* series data sets gave rise to large numbers of empty intersections - perhaps not surprising as the number of objects is increased at a relatively low density in a randomised formal context.





7 Conclusions

In conclusion, the performance of In-Close5 clearly demonstrates the efficiency savings provided by the new method of inheriting canonicity test failures when its results are compared to those of In-Close4 (the same algorithm but without canonicity test failure inheritance). In-Close5 clearly outperforms FCbO, the algorithm that features the existing method of inheriting canonicity test failures. Although FCbO's method inherits more test failures than the new method, the simplicity of the new method warrants its attention as a useful contribution to the area. It was shown in In-Close3 [2] that incorporating FCbO's method gave little improvement of performance, due to the computational overheads of implementing it, whereas it is show here that the incorporation of the new method does improve performance significantly.

An implementation of In-Close5 is available, free and open source, at `https://sourceforge.net/projects/inclose/`.

References

1. 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. pp. 37–50. Springer (2014)
2. Andrews, S.: A best-of-breed approach for designing a fast algorithm for computing fixpoints of galois connections. Information Sciences **295**, 633–649 (2015)
3. 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). pp. 50–62. Springer (2011)
4. Andrews, S.: Making use of empty intersections to improve the performance of cbo-type algorithms. In: Proceedings of the 14th International Conference on Formal Concept Analysis. pp. 56–71. Springer (2017)
5. Frank, A., Asuncion, A.: UCI machine learning repository: http://archive.ics.uci.edu/ml (2010)
6. 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. pp. 71–82 (2008)
7. Krajca, P., Vychodil, V., Outrata, J.: Advances in algorithms based on CbO. In: Kryszkiewicz, M., Obiedkov, S. (eds.) CLA 2010. pp. 325–337. University of Sevilla (2010)
8. Kuznetsov, S.O.: Mathematical aspects of concept analysis. Mathematical Science **80**(2), 1654–1698 (1996)
9. Outrata, J., Vychodil, V.: Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data. Information Sciences **185**(1), 114–127 (Feb 2012). https://doi.org/10.1016/j.ins.2011.09.023