Revisiting Algorithms for Fuzzy Concept Lattices*

Domingo López-Rodríguez[0000−0002−0172−1585], Ángel Mora[0000−0003−4548−8030], and Manuel Ojeda-Hernández[0000−0003−4785−6802]

Departamento de Matemática Aplicada, Universidad de Málaga, 29071 Málaga, Spain
{dominlopez, amora, manuojeda}@uma.es

Abstract. A central notion in Formal Concept Analysis is the concept lattice. This lattice allows describing a hierarchical biclustering between objects and attributes of a formal context, whose hierarchy is defined by an order that expresses the specialisation-generalisation relationship between concepts. It is a fundamental way of representing the knowledge implicit in the context. Therefore, in practice, due to its theoretical complexity, it is necessary to define computationally efficient algorithms for its calculation. In the literature, several algorithms, using different approaches, have been proposed for the computation of the lattice in the classical framework, where the presence of an attribute in an object is modelled as a binary value, indicating that the attribute is either present or absent. However, it is possible to extend this framework to take into account the different degrees to which an attribute could be present in an object. Through this extension, it is possible to model fuzzy situations where the attribute is not 100% present in an object, giving flexibility to the model. In this paper, we review some of the best known algorithms for the calculation of the concept lattice in the binary version, and we extend them for the calculation of the fuzzy concept lattice, presenting the most significant differences with respect to the original binary versions. In addition, we will present examples of the execution of these new versions of the algorithms.

Keywords: Formal concept analysis · Concept Lattice · Graded attributes · Algorithms.

1 Introduction

In recent years, Formal Concept Analysis (FCA) is reaching a high degree of maturity. Only a few decades have passed since the first works of the pioneers, R. Wille and B. Ganter [21, 11]. The number of publications in the area is increasing, as well as the direct applications of FCA in emergent and hot topics as Social Network Analysis [20], Recommender Systems [9, 7], Medical Diagnosis [22, 8],

* This work has been partially supported by the Ministerio de Ciencia, Innovación y Universidades [FPU19/01467, TIN2017-89023-P, PGC2018-095869-B-I00] and the Junta de Andalucía [UMA2018-FEDERJA-001].

E-learning Systems [19, 17] and others. For the non-expert reader, there are many possible references but we highlight [10] because of the pedagogical vision of the authors.

Within the classical FCA framework, the knowledge extracted from a binary table of data (formal context) is essentially represented as two complementary entities: the concept lattice and a basis of context-valid implications. An interesting research line is the one that studies algorithms to compute both knowledge entities. The intrinsic exponential nature of the process of building the lattice and the basis of the implications encourages this line to be very active from the very beginning. This paper focuses on this line.

In this work, we emphasise the importance of the concept lattice: it represents an exhaustive analysis of the closed sets according to the well-known derivation operators forming a Galois connection. That allows us to establish a hierarchical biclustering of the relationships between objects and attributes.

Let us recall that the number of concepts can be exponential in the size of the input context and the problem of determining this number is #P-complete [14]. It is therefore necessary to develop algorithms that intelligently exploit the structure of the lattice itself and the properties of the closure system for the efficient computation of the concept set. To solve this problem, a wide variety of proposals have been made. The most naive one consists of exhaustively enumerating all possible subsets and then checking which ones are closed. More advanced methods use pruning strategies to avoid such complete enumeration. For example, the NextClosure algorithm [11] uses the lectic order of intents. Others are based on determining the set of all extents by performing a recursive search accompanied by canonicity tests that allow a more efficient exploration. Those based on the CbO strategy [13], such as FastCbO (FCbO) [12, 18], or InClose [1] stand out.

The classical FCA framework is extended by considering non-binary relationships between the objects and attributes, i.e. $\mathbb{L}$-fuzzy relations. The first approaches were due to Burusco and Fuentes [6] and Bělohlávek [3]. Fuzzy FCA (FFCA) considers $\mathbb{L}$-contexts $\langle G, M, I\rangle$ where G is a set of objects, M is a set of attributes an I is a fuzzy relation between G and M. The interpretation of $I(x, y)$ is the truth degree to which object x has attribute y.

For FFCA, few algorithms have been developed by adapting the classical ones to the generalised framework. We highlight [2, 4] as the first approaches to the problem in this fuzzy setting and emphasise [5], where the authors compute the set of fixpoints of a given fuzzy closure operator, relating fuzzy and classical closure operators. One might expect that with *native fuzzy* approaches the resulting algorithms could be more efficient. Thus, it is interesting to adapt classical approaches to the fuzzy setting and to promote the research on *ad hoc* fuzzy algorithms. The purpose of this work is to generalise other known and efficient algorithms to FFCA. We analyse their efficiency and see what the most relevant differences with respect to the binary versions are.

The rest of the work is organised as follows. In Section 2, we present the background and the most prominent algorithms in classical FCA and in Section 3, we show how we have extended the InClose family of algorithms to the fuzzy setting. We provide an example of the execution of these new versions in Section 4 and present our conclusions and the future lines of work in Section 5.

2 Preliminaries and related works

In this section we give a brief summary of fuzzy set theory. Throughout the paper, the reader is expected to be familiar with Formal Concept Analysis notions.

Let $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$ be a complete residuated lattice, that is, $(L, \wedge, \vee)$ is a complete lattice with 0 and 1 being the least and the greatest elements of L, respectively, $(L, \otimes, 1)$ satisfies $\otimes$ is commutative, associative, and 1 is neutral with respect to $\otimes$, and $\otimes$ and $\rightarrow$ satisfy the so-called *adjointness property*: for all $a, b, c \in L$, we have that $a \otimes b \leq c$ iff $a \leq b \rightarrow c$.

An $\mathbb{L}$-*set* is a mapping $X \colon U \rightarrow L$ from the universe set U to the truth values set L, where $X(u)$ means the degree to which u belongs to X. The set of $\mathbb{L}$-sets on the universe U is denoted by L^U. If U is finite, it is common to denote fuzzy sets as $X = \{{}^{l_1}\!/\!_{u_1}, {}^{l_2}\!/\!_{u_2}, \dots, {}^{l_n}\!/\!_{u_n}\}$, where each element u_i belongs to X in degree l_i. If $l_i = 0$, the element u_i might be omitted and if $l_i = 1$ we denote it by $\{u_i\}$. Operations with $\mathbb{L}$-sets are defined element-wise. For instance, $A \otimes B \in L^U$ is defined as $(A \otimes B)(u) = A(u) \otimes B(u)$ for all $u \in U$.

Now we turn our focus to the main features of the classical algorithms that are considered the main background of this work. All of them start from a binary formal context. For the pseudocode and more details, we refer the reader to the sources cited in the text.

Although former attempts to compute the maximal rectangles of a binary relation exist [16], the first proper approach to compute all the formal concepts in the FCA framework was the NextClosure algorithm [11]. The goal is to compute all the intents of the formal concepts. Thus, the search space is formed by all possible attributes subsets. The NextClosure approach differs from that of the brute force since it introduces a pruning strategy when exploring the search space, based on the introduction of an order (called *lectic* order) between the sets of attributes.

NextClosure begins with the computation of the closure of the empty set. After this closure computation, the algorithm recursively enumerates the next closed sets in the lectic order. It is well-known that the complexity is exponential in the worst case, but it is interesting to know that the algorithm has a polynomial delay, $\mathcal{O}(|G||M|^2)$ searching all the intents or $\mathcal{O}(|G|^2|M|)$ searching all the extents. This fact establishes an upper bound to compute one concept.

Krajča et al. [12] presented a survey on algorithms for computing formal concepts. The authors said "the major issue of widely-used algorithms for computing formal concepts is that some concepts are computed multiple times which brings significant overhead". Close-by-one (CbO) [13] proposed an interesting line to reduce this overhead and, at present, the faster algorithms follow the framework posed by CbO. Kuznetsov's algorithm builds a tree where nodes represent computed closures, and edges connecting two closures A, B (into two nodes) are labelled by attributes $\{y\}$ connecting them when B is the closure of $A \cup \{y\}$.

With respect to CbO, Fast Close-by-One (FCbO) [12, 18] achieves a reduction of the number of concepts which are computed multiple times, including an additional canonicity test. Currently, FCbO is one of the main methods to compute formal concepts having parallelizable versions and several new improvements.

In this paper, due to its simplicity of implementation, we focus on another highly efficient algorithm named InClose [1]. It uses incremental closure and matrix searching as key features to achieve a faster method. InClose avoids to repeat the computation of closures and only computes a closure per concept in an incremental way. We propose fuzzy extensions for the different versions of the In-Close algorithm in the literature. We implement and compare the results of the algorithms in the following sections.

3 Extension of the algorithms to the fuzzy setting

The topic of discussion in this section will be the extension of the classical FCA algorithms to a fuzzy setting. Let $\mathbb{L} = (L, 0, 1, \wedge, \vee, \otimes, \to)$ be a complete residuated lattice. Since all our sets of objects and attributes are finite, without loss of generality we will consider L to be finite as well. This is because the set $\{I(x, y)\}_{x\in G, y\in M}$ is finite and we can consider $\mathbb{L}'$ to be the smallest complete lattice such that $\{I(x, y)\}_{x\in G, y\in M} \subseteq \mathbb{L}' \subseteq \mathbb{L}$. The complete lattice $\mathbb{L}'$ exists and is finite, the proof is not the goal of this paper so it is omitted.

In classical algorithms such as InClose2, introduced by Andrews [1], the code runs for each attribute $m \in M$. The difference in the fuzzy case is that we have to consider the degrees of each attribute, hence the run of the algorithm will take the first attribute m_1 and run through all the different truth values in L. This tour on the possible degrees could go either forward or backwards through the elements of L. The choice taken so far is to run backwards on L in order to avoid redundant cycles. For instance, there is no point in adding to the set B all the degrees of an attribute one by one $\{{}^{0.1}/_{m_1}, {}^{0.2}/_{m_1}, \ldots, {}^{1}/_{m_1}\}$ since running backwards the code would add $\{{}^{1}/_{m_1}\} = \{m_1\}$ directly, thus saving computation time.

The `FuzzyInClose2` algorithm is the fuzzy extension to Andrews' InClose2. In Algorithm 1, the procedure is initialised. For this, only the formal context is needed, then the auxiliary function `InClose2_ChildConcepts` is called with the initial extent G, intent $\varnothing$, attribute index 0 and concept list $\mathbb{C} = \varnothing$.

Algorithm 1: FuzzyInClose2($\mathbb{K}$)

```
  Input: $\mathbb{K} = (G, M, I)$: A formal context with grades in
         $L = \{0 < l_1 < \ldots < l_n = 1\}$
  Output: $\underline{\mathbb{B}}(\mathbb{K})$: The concept lattice of $\mathbb{K}$.
1 $\mathbb{C} := \varnothing$
2 InClose2_ChildConcepts($G$, $\varnothing$, 0, $\mathbb{C}$)
3 return $\underline{\mathbb{B}}(\mathbb{K}) = \mathbb{C}$
```

The auxiliary function used above is the one represented in Algorithm 2 below. This is the one that actually extends the InClose2 algorithm to the fuzzy framework and performs the recursive search. Notice how in line 3 we make k range from n to 1, this is notation for running through the elements of L in a decreasing order.

Algorithm 2: InClose2_ChildConcepts(A, B, y, $\mathbb{C}$)

```
Input: $A$: An extent; $B$: The intent corresponding to $A$, that will be
       completed in this execution; $y$: index of the attribute where to start
       the exploration of this branch; $\mathbb{C}$: the global variable where to
       accumulate the computed concepts.
1  $Q := \varnothing$
2  for $j \in \{y+1, \dots, |M|\}$ do
3      for $k \in \{n, \dots, 1\}$ do
4          $g := l_k$
5          if $B \cap \{m_j\} \subsetneq \{{}^{g}/_{m_j}\}$ then
6              $C := A \cap \{{}^{g}/_{m_j}\}^{\downarrow}$
7              if $C^{\uparrow} \cap \{m_j\} = \{{}^{g}/_{m_j}\}$ then
8                  if $C = A$ then
9                      $B := B \cup \{{}^{g}/_{m_j}\}$
10                 else
11                     if $B \cap M_j = C^{\uparrow_j}$ then
12                         $Q := Q \cup \{(C, j, k)\}$
13 $\mathbb{C} := \mathbb{C} \cup \{(A, B)\}$
14 for $(C, j, k) \in Q$ do
15     $D := B \cup \{{}^{l_k}/_{m_j}\}$
16     InClose2_ChildConcepts($C$, $D$, $j$, $\mathbb{C}$)
```

Remark 1. The notation of the pseudocode may look cumbersome. It is done set-theory style in order to maximize the amount of information given to the reader. However, from the coding point of view, the computation of the algorithms is much simpler due to the use of fuzzy-set notation. For instance, in Algorithm 4, in line 5, when it reads $B \cap \{m_j\} \subsetneq \{{}^{g}/_{m_j}\}$ in fuzzy-set notation this is just $B(m_j) < g$.

Notice that prior to line 1 it is required that A is an extent. The algorithm starting with $A = G$ ensures this since G is always an extent and all ramifications of the algorithm are built as intersections of G with attribute-extents, that always give other extents. It is also required that B is the intent corresponding to A accumulated over the course of the execution branch that has led to this node. Note that B will be completed in the current level of recursion. Hence, $B \subseteq A^{\uparrow}$ is needed. This is ensured in the first iteration of the algorithm since $B = \varnothing$, through all the ramifications this still holds, although it cannot be seen that trivially.

Line 1: Initialize a list Q as empty.

Line 2: Iterate across the formal context, from a starting attribute $y+1$ up to attribute n, where n is the number of attributes in the context.

Line 3: Iterate across the truth values in L backwards and...

Line 4: ... call the current truth value g.

Line 5: Skip attributes already in B, since the truth values are gone through in decreasing order, $B \cap \{m_j\} \neq \varnothing$ implies m_j belongs to B in a higher degree than g, so this can be skipped too.

Line 6: Form an extent, C, by intersecting the current extent, A, with the next column of objects in the context with degree g.

Line 7: Skip all g such that $C^{\uparrow} \cap \{m_j\} \neq \{{}^{g}/_{m_j}\}$. This is a partial canonicity test because in this iteration we are focused only on g.

Line 8: ... if $C = A$, then...

Line 9: ... add $\{{}^{g}/_{m_j}\}$ to the set B...

Lines 10 and 11: ... else, for the canonicity test, if the partial intent of C up to j is exactly B restricted to the first $j-1$ attributes...

Line 12: ... store $(C, j, k) \in Q$...

Line 13: Store the concept (A, B) in the list $\mathbb{C}$...

Line 14: For each extent C, attribute m_j and truth value l_k stored in Q...

Line 15: Create the intent $D = B \cup \{{}^{l_k}/_{m_j}\}$.

Line 16: Call `InClose2_ChildConcepts` to compute the child concepts of C starting from the attribute m_j and complete the intent D.

It is interesting to remark that in line 7 the calculation of $C^{\uparrow} \cap \{m_j\}$ does not require the computation of the intent since it is simply

$$(C^{\uparrow} \cap \{m_j\})(m_j) = C^{\uparrow}(m_j) = \bigwedge_{o \in G} \left(C(o) \to I(o, m_j)\right).$$

Similarly, the computation of the intent $C^{\uparrow_j}$ is not needed since it follows from the previous formula: the implementation would only present a loop over the attributes $i < j$, computing $C^{\uparrow} \cap \{m_i\}$ using the formula above, and stopping early when $(C^{\uparrow} \cap \{m_i\})(m_i) = C^{\uparrow}(m_i) \neq B(m_i) = B \cap \{m_i\}$.

InClose2 has been proved to be correct and fairly quick timewise, even though it does several redundant computations. For instance, assume that $A \cap \{{}^{l_k}/_{m_j}\}^{\downarrow}$ is empty. Then, for all child concepts from this extent on, it follows trivially that $C \cap \{{}^{l_k}/_{m_j}\}^{\downarrow} = \varnothing$, hence this iteration may be skipped with no loss of information. This is taken into account in a series of algorithms called InClose4, introduced by Andrews [1] as well. Actually, there are two variations of InClose4 which are called InClose4a and InClose4b. In these algorithms a list P is added to the parameters in order to track the empty intersections in all the child extent iterations. This way, several calculations are skipped and thus, runtime is shortened.

Next, we give some details of the differences between `FuzzyInClose2` and `FuzzyInClose4a`. Due to space reasons, only the pseudocode of `FuzzyInClose4b` is displayed, since it is the fastest and the less computation demanding.

The main method `FuzzyInClose4a` only differs from the InClose2 version in that the auxiliary function, now called `InClose4a_ChildConcepts`, requires an extra argument, P, that keeps track of the empty intersections occurring in a branch of the execution, and that is initially empty.

There are subtle changes in lines 5, 11 and 12 of `InClose2a_ChildConcepts` that improve it to `InClose4a_ChildConcepts`.

Line 5 Skip all attributes that are already in B in degree g or higher or in P in degree g or lower.

Line 5 in InClose4a: we can omit all attributes that are already in B in degree g or higher or in P in degree g or lower, by performing the check $B \cap \{m_j\} \subsetneq \{{}^{g}/_{m_j}\}$ ***and*** $(P \cap \{m_j\} = \varnothing$ ***or*** $\{{}^{g}/_{m_j}\} \subsetneq P \cap \{m_j\})$.

After line 10 Before performing the canonicity test $B \cap M_j = C^{\uparrow_j}$ in line 11 (Algorithm 2), the fuzzy version of InClose4a will check if the new extent is empty, thus updating P, the record of empty intersections, accordingly: **if** $C = \varnothing$ **then** $P := \{{}^{g}/_{m_j}\} \cup (P \smallsetminus \{m_j\})$. The update of P is designed to keep track of the *minimal* degree g for which the computation of the extent C provides the empty set. Thus, the condition in line 5 will be optimal and reject all the cases that inherit the empty intersection.

Notice that the execution of `FuzzyInClose4a`($\mathbb{K}$) computes all the extents of the given context from G to $\varnothing$, in case $\varnothing$ is indeed an extent.

Another algorithm that blends the spirit of `FuzzyInClose2` and the avoiding empty intersections of `FuzzyInClose4a` is the so-called `FuzzyInClose4b` algorithm.

Algorithm 3: FuzzyInClose4b($\mathbb{K}$)

```
Input: $\mathbb{K} = (G, M, I)$: A formal context with grades in
       $L = \{0 < l_1 < \ldots < l_n = 1\}$
Output: $\underline{\mathbb{B}}(\mathbb{K})$: The concept lattice of $\mathbb{K}$.
1 $\mathbb{C} := \varnothing$
2 InClose4b_ChildConcepts($G$, $\varnothing$, 0, $\varnothing$, $\mathbb{C}$)
3 if $M^{\downarrow} = \varnothing$ then
4     $\mathbb{C} := \mathbb{C} \cup \{(\varnothing, M)\}$
5 return $\underline{\mathbb{B}}(\mathbb{K}) = \mathbb{C}$
```

Introduced in the classical FCA by Andrews [1], InClose4b keeps the idea of skipping empty intersections to speed up the code but checks this condition earlier in the process. As we can see in Algorithm 4, lines 8 and 9, the filter of the extent C being empty is applied before than in `FuzzyInClose4a`, where this is done in lines 11 and 12. This makes it faster to discard empty intersections, but it comes with a price. Originally, there were extents not computed by this algorithm. Fortunately, this occurs only once and it is the extent $\varnothing$, in the case it is indeed an extent. This is a small price to pay and it is solved in Algorithm 3, lines 3 and 4, where a direct check on $(\varnothing, M)$ being a formal concept is made and, if it is, it is stored in $\mathbb{C}$.

Algorithm 4: InClose4b_ChildConcepts(A, B, y, P, $\mathbb{C}$)

```
Input: $A$: An extent; $B$: The intent corresponding to $A$, that will be
       completed in this execution; $y$: index of the attribute where to start
       the exploration of this branch; $P$: record for empty intersections; $\mathbb{C}$:
       the global variable where to accumulate the computed concepts.
1  $Q := \varnothing$
2  for $j \in \{y+1, \dots, |M|\}$ do
3      for $k \in \{n, \dots, 1\}$ do
4          $g := l_k$
5          if $B \cap \{m_j\} \subsetneq \{{}^{g}/_{m_j}\}$ and $(P \cap \{m_j\} = \varnothing$ or $\{{}^{g}/_{m_j}\} \subsetneq P \cap \{m_j\})$
            then
6              $C := A \cap \{{}^{g}/_{m_j}\}^{\downarrow}$
7              if $C^{\uparrow} \cap \{m_j\} = \{{}^{g}/_{m_j}\}$ then
8                  if $C = \varnothing$ then
9                      $P := (P \setminus \{m_j\}) \cup \{{}^{g}/_{m_j}\}$
10                 else
11                     if $C = A$ then
12                         $B := B \cup \{{}^{g}/_{m_j}\}$
13                     else
14                         if $B \cap M_j = C^{\uparrow_j}$ then
15                             $Q := Q \cup \{(C, j, k)\}$
16 $\mathbb{C} := \mathbb{C} \cup \{(A, B)\}$
17 for $(C, j, k) \in Q$ do
18     $D := B \cup \{{}^{l_k}/_{m_j}\}$
19     InClose4b_ChildConcepts($C$, $D$, $j$, $P$, $\mathbb{C}$)
```

4 An example

In this section, we present an example of the implementation of the fuzzy versions `FuzzyInClose2`, `FuzzyInClose4a` and `FuzzyInClose4b`. Due to space restrictions, we cannot present the complete trace of the execution of these algorithms. However, we have chosen to present this trace in graphical form, so it is possible to check the different steps followed by each of these algorithms. For this work, we have implemented the algorithms using the functionalities provided by the `fcaR` package [15] in the R programming language and used those implementations for the example in this section.

In this example, we consider the formal context of Table 1, where the set of grades for the attributes $M = \{a, b, c, d, e\}$ is $L = \{0, {}^{1}/_{2}, 1\}$. It is not a large context in order to make the execution graphs legible.

The first of the tested versions is `FuzzyInClose2`. In Figure 1, we can check the order in which the extent intersections are computed in this algorithm. Starting with $\varnothing$, the first level of recursion (intersection of G with all attribute extents) is carried out from left to right, in the attribute order defined in the previous

Table 1. Simple formal context for the example.

	a	b	c	d	e
o1	0	0	½	0	1
o2	1	½	½	0	½
o3	½	½	0	½	½

section. We have used a colour code to indicate the possible situations that can occur when checking a given node: a red box indicates that the canonicity test has failed at that point; a light orange box indicates that the partial canonicity test is not passed; and a grey box appears when parent and child nodes have the same extent, so the intent of the parent is updated. Once the whole level has been checked, we start to develop the branches in the next phase of the recursion, going through the nodes from left to right. So, under the node labelled $\{a\}$, we start to develop its branch, following exactly the same procedure as before: we have to go through all the attributes/degrees starting from the attribute $\{b\}$, and then develop the leftmost branch that is not developed at that moment. Repeating the process, we arrive at 5 levels of recursion for the calculation of all concepts.

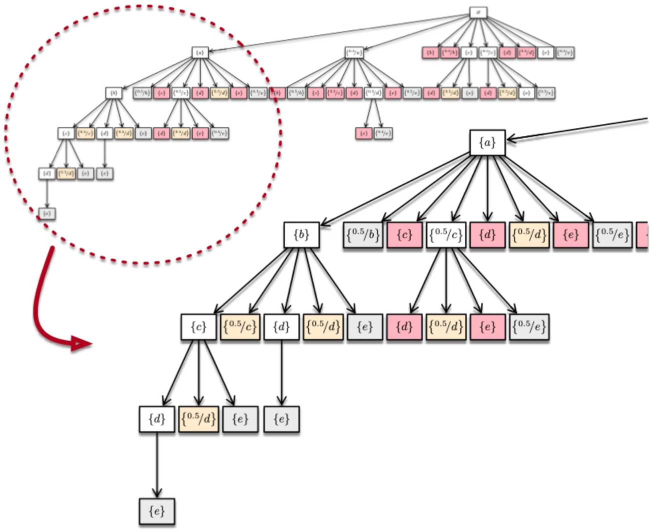

Fig. 1. Order of computations performed by the fuzzy version of InClose2.

The execution graphs for `FuzzyInClose4a` and `FuzzyInClose4b`, which take into account the occurrence of empty intersections at each level of recursion of the algorithm, are presented in Figure 2 below. These empty intersections are inherited downwards in the branches arising from that level of recursion, and allow to reduce the number of operations to be performed, as well as the depth of the recursion. The same colour code is used as in Figure 1, adding the blue boxes to represent iterations where an empty extent was found.

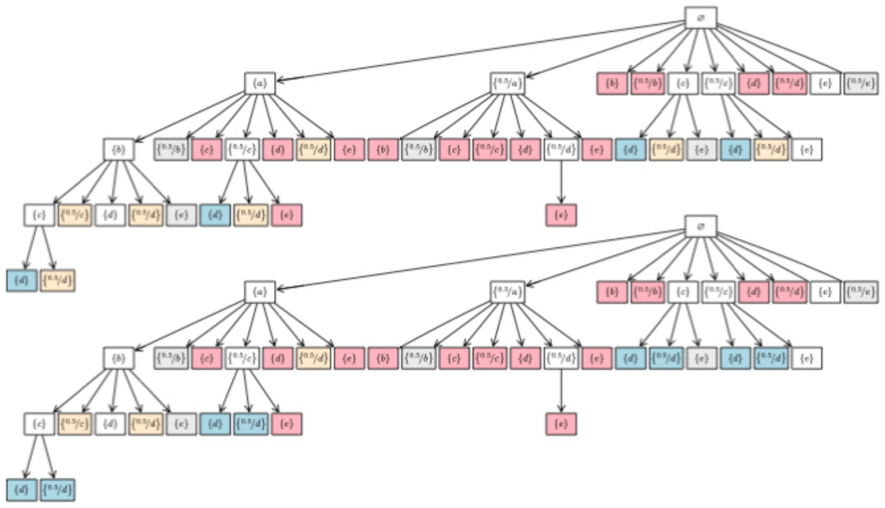

Fig. 2. Order of computations performed by the fuzzy version of InClose4a (up) and InClose4b (down).

As can be seen, there is no great difference, at least in this example, between the two proposals, although there is a clear reduction in the computational load with respect to our baseline of comparison, the `FuzzyInClose2` method. We can see that, instead of 5 levels of recursion, in the `FuzzyInClose4` family, only 4 appear. Due to the different characteristics of both methods, it is to be expected that in larger contexts differences will appear and that, probably, they will be in favour of the `FuzzyInClose4b` variant, since it makes a greater pruning in the exploration of the graph of intersections of extents.

Finally, we present in Table 2 a quantitative comparison of the number of operations carried out by each algorithm. We will only count computationally expensive operations: partial canonicity tests, i.e., checks of $C^{\uparrow} \cap \{m_j\} = \{{}^{g}/_{m_j}\}$; complete canonicity tests, as in the binary case, on the equality $B \cap M_j = C^{\uparrow_j}$; and the number of attribute intents computed and the number of intersections of extents. In this table we show the number of such operations for each algorithm.

Table 2. Number of computations performed by each of the algorithms for the dataset in Table 1.

Algorithm	Partial tests	Full tests	#Intents	#Extents
FuzzyInClose2	49	29	97	50
FuzzyInClose4a	41	29	89	42
FuzzyInClose4b	33	25	74	42

In the table, different facts can be observed. On the one hand, looking at the #Extents column, we can see how versions 4a and 4b of the algorithm build smaller execution graphs, since the number of intersections coincides with the number of nodes explored in the execution of the graph. On the other hand, although the execution graph is the same for versions 4a and 4b, the different strategy for exploiting empty intersections produces a clear reduction in the number of operations to be carried out, since fewer tests, both partial and complete, are checked. Thus, the exploration carried out by `FuzzyInClose4b` is more efficient and, for larger contexts, could be the fastest of the 3 algorithms.

5 Conclusions and future work

The concept lattice is a fundamental way of representing the knowledge implicit in the context. In the literature, several algorithms, using different approaches, have been proposed for the computation of the lattice in the classical framework. In this paper some of the extensions of these algorithms to the fuzzy framework have been introduced. These algorithms are of the family of InClose. After a brief explanation of the code, an example was shown to illustrate the trace of the different approaches. Lastly, the algorithms are compared taking into account runtime, number of test calculations and number of extents computed.

As a future work, we devise to study several optimisations to the InClose4 family, taking advantage of the structure of the degrees in L. The aim of the possible optimisations will be to reduce the number of computations both of intents and extents, hence reducing the computational cost of the algorithm.

Furthermore, we aim to explore generalisations of other algorithms, such as the FastCbO family or the NextNeighbour or NextPriorityConcept, for the fuzzy setting, along with different optimisations that could alleviate the greater computational cost when compared to the binary case. We want to perform a thorough comparison between the different versions of the algorithms in terms of execution time and the number of computations required to compute the fuzzy concept lattice. We expect to incorporate the implementation of all algorithms for lattice computation in the fuzzy framework into the above-mentioned `fcaR` package.

Other research line for future works will study the extension of the provided algorithms to compute the canonical basis of implications in this fuzzy setting. It has already been proved that, in the binary case, CbO-like algorithms can be modified to provide such a basis. We aim to extend the fuzzy versions of the algorithms to compute the basis.

References

1. Andrews, S.: Making use of empty intersections to improve the performance of CbO-type algorithms. In: In.Conference on Formal Concept Analysis. pp. 56–71. Springer (2017)

2. Belohlavek, R.: Algorithms for fuzzy concept lattices. In: Int. Conf. on Recent Advances in Soft Computing. pp. 200–205 (2002)
3. Belohlavek, R.: Fuzzy relational systems: foundations and principles, vol. 20. Springer Science & Business Media (2002)
4. Belohlavek, R., De Baets, B., Outrata, J., Vychodil, V.: Lindig's algorithm for concept lattices over graded attributes. LNCS **4617**, 156–167 (2007)
5. Belohlavek, R., Konecny, J.: Fixpoints of fuzzy closure operators via ordinary algorithms. In: 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). pp. 1–6 (2017)
6. Burusco-Juandeaburre, A., Fuentes-González, R.: The study of the L-fuzzy concept lattice. Mathware and Soft Computing **1**(3), 209–218 (1994)
7. Chemmalar Selvi, G., Lakshmi Priya, G.G.: Rating prediction method for item-based collaborative filtering recommender systems using formal concept analysis. EAI Endorsed Transactions on Energy Web **8**(33) (2021)
8. Cordero, P., Enciso, M., López, D., Mora, A.: A conversational recommender system for diagnosis using fuzzy rules. Expert Systems with Applications **154** (2020)
9. Cordero, P., Enciso, M., Mora, Á., Ojeda-Aciego, M., Rossi, C.: A formal concept analysis approach to cooperative conversational recommendation. Int.Journal of Computational Intelligence Systems **13**(1) (2020)
10. Ganter, B., Obiedkov, S.: Conceptual Exploration. Springer Berlin Heidelberg (2016)
11. Ganter, B.: Two basic algorithms in concept analysis (1984), FB4-Preprint 831. Darmstadt, Germany: Technische Hochschule Darmstadt
12. Krajca, P., Outrata, J., Vychodil, V.: Advances in algorithms based on CbO. In: CLA. vol. 672, pp. 325–337. Citeseer (2010)
13. Kuznetsov, S.: A fast algorithm for computing all intersections of objects in a finite semi-lattice. Automatic Documentation and Mathematical Linguistics **27**, 11–21 (1993)
14. Kuznetsov, S.O.: Interpretation on graphs and complexity characteristics of a search for specific patterns. Automatic Documentation and Mathematical Linguistics **24**(1), 37–45 (1989)
15. López-Rodríguez, D., Mora, A., Domínguez, J., Villalón, A., Johnson, I.: fcaR: Formal Concept Analysis (2020), https://cran.r-project.org/package=fcaR
16. Norris, E.M.: An algorithm for computing the maximal rectangles in a binary relation. Revue Roumaine de Mathématiques Pures et Appliquées **23**(2), 243–250 (1978)
17. Ojeda-Hernández, M., Pérez-Gámez, F., Mora Bonilla, Á., López-Rodríguez, D.: Using logic to determine key items in math education. In: 15th International Conference e-Learning, EL 2021. pp. 62–69 (2021)
18. Outrata, J.: A lattice-free concept lattice update algorithm based on* CbO. In: CLA. vol. 2013, pp. 261–274. Citeseer (2013)
19. Priss, U.: A preliminary semiotic-conceptual analysis of a learning management system. In: Procedia Computer Science. vol. 176 (2020)
20. Salman, H.E.: Feature-based insight for forks in social coding platforms. Information and Software Technology **140**, 106679 (2021)
21. Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In: Rival, I. (ed.) Ordered Sets. pp. 445–470. Springer Netherlands, Dordrecht (1982)
22. Zheng, F., Cui, L.: A lexical-based formal concept analysis method to identify missing concepts in the NCI thesaurus. In: Proceedings - 2020 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2020 (2020)