Fuzzy Sets and Systems 520 (2025) 109574

Contents lists available at ScienceDirect

Fuzzy Sets and Systems

journal homepage: www.elsevier.com/locate/fss

Close-by-One-like algorithms in the fuzzy setting: Theory and experimentation

Domingo López-Rodríguez [a], Manuel Ojeda-Hernández [b,*], Ángel Mora [a], Carlos Bejines [a]

[a] Departamento de Matemática Aplicada, Universidad de Málaga, 29071 Málaga, Spain
[b] Department of Computing Science, Umeå University, Umeå, Sweden

ARTICLE INFO

Keywords:
Formal concept analysis
Close-by-One
Algorithms
Comparison
Performance
Fuzzy

ABSTRACT

In Fuzzy Formal Concept Analysis (FFCA), concept lattices are computed by scaling the problem and applying ordinary FCA algorithms. In this paper, the CbO family of algorithms is extended to work natively in the fuzzy setting, they are proved to be correct and output the whole set of formal concepts, which makes them mathematically equivalent to the scaling approach.
However, experimental results demonstrate the performance improvement of these methods compared to scaling. The paper also discusses a new fuzzy strategy based on blacklisting redundant truth values to enhance the performance of algorithms by taking advantage of the structure of the residuated lattice.

1. Introduction

Formal Concept Analysis (FCA) was born in the eighties [47] and in these few decades has evolved into a mature field, not only in the theoretical framework but also in applications. For readers unfamiliar with the field, numerous references are available, but noteworthy among them is [22], esteemed for its pedagogical approach. Nowadays, FCA has settled as a data analysis tool [8,10,25]. It has found utility in a number of leading-edge fields, such as Social Network Analysis [14,43], Recommender Systems [12,15], Medical Diagnosis [13,48] or E-learning Systems [38,41], just to name a few. As an application-useful data analysis tool, FCA has gained interest for a variety of reasons, ranging from its ability to represent meaningful relationships between objects and attributes to uncovering underlying patterns and hidden structures in datasets.

In this day and age, the focus is on tractable, explainable and trustworthy methods [34]. Unlike other techniques used in machine learning or artificial intelligence, FCA is distinguished by its explainability [18,44,46]: intuitive visual representations, knowledge extracted using formal techniques that allow to trace where a certain relationship between data in an implication has arisen or to prove with a clear semantic interpretation the formal concepts generated. Therefore, in FCA, it is simple and easy to explain how to establish the relationship between objects and attributes in the data capture phase and the equally intuitive and easily interpretable knowledge extracted. Patterns, relationships between attributes, clusters of objects or attributes, trends and anomalies appear naturally without the need for prior assumptions about their structure.

* Corresponding author.
E-mail addresses: dominlopez@uma.es (D. López-Rodríguez), manueloh@cs.umu.se, manuojeda@uma.es (M. Ojeda-Hernández), amora@uma.es (Á. Mora), cbejines@uma.es (C. Bejines).

https://doi.org/10.1016/j.fss.2025.109574
Received 24 January 2025; Received in revised form 10 June 2025; Accepted 21 August 2025

Available online 1 September 2025

Table 1
An example of fuzzy formal context.

	$m1$	$m2$	$m3$
$g1$	0.5	0	1
$g2$	0.75	0.75	0
$g3$	0	0.25	0.75
$g4$	0	1	0
$g5$	0	0	1

The knowledge distilled from a binary data table (formal context) manifests in two primary aspects: the discernible concepts and a collection of attribute-implications [23]. Moreover, the concepts are not just isolated entities (forming, indeed, a complete clustering); they are intricately structured and hierarchically organised, lending a deeper understanding to the relationships inherent within the data.

Given the exponential nature of the process of computing the concept lattice [29], research into new methods that substantially reduce computational times leads to the possibility of applying FCA to real problems with large datasets. There is a whole line of research whose goal is to obtain faster, more optimised algorithms to compute the concept lattice. We refer the reader to [27] due to the detailed walk-through on the different CbO-based approaches up to date.

In many cases, relationships between objects and attributes cannot be fully disclosed, that is, having an attribute is not an absolute value, but there is some flexibility. Due to this type of situations, FCA with fuzzy attributes appeared, originally introduced by Burusco and Fuentes-González [11] and later by Pollandt [40] and Bělohlávek [6], the research on Fuzzy FCA (FFCA) is now standard.

The exponential nature of computing the formal concepts of a formal context in the crisp setting is worsened in the fuzzy paradigm. Therefore, new methods and algorithms need to be developed to make computing with FFCA feasible in applications. One of the first approaches in this line was published by Bělohlávek [5], extending Ganter's NextClosure [21] algorithm to the fuzzy framework. On the other hand, Bělohlávek and Konečny [9] proved that via scaling the fuzzy formal context, the crisp algorithms could be used to compute the set of all the fuzzy formal concepts, this approach was called that of wrapping algorithms.

In this paper, we tackle the problem of extending the most known CbO-based algorithms to a native fuzzy approach. The choice of this particular family of algorithms is due to the state-of-the-art speed they achieve as well as the intelligent use of the algebraic properties of the Galois connection, which is still present in the fuzzy framework. Specifically, the CbO, FastCbO and the InClose family of algorithms will be extended to a native fuzzy approach by running through the set of truth values in the algorithms and adapting the canonicity tests accordingly. The structure of the paper is as follows, in Section 2 some preliminaries are shown in order to make the paper self-contained. In Section 3, the extensions of these algorithms to the fuzzy setting is presented. At the end of the section, a blacklisting strategy which is native to the fuzzy framework is presented. This idea takes advantage of the lattice structure of the set of truth values. Next, Section 4 shows the experiments done to compare the algorithms in a variety of real and synthetic problems. Lastly, we provide some conclusions and possible future lines of work.

2. Preliminaries

In this section, we shall introduce the fundamentals of formal concept analysis in the fuzzy environment, as well as established methodologies documented in existing literature for computing the concept set in the fuzzy setting.

2.1. Fuzzy formal concept analysis

Firstly, let us provide a precise definition of what we mean by fuzzy formal concept analysis. For a written out section on this, we refer the reader to [6].

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 $a \otimes b \leq c$ if and only if $a \leq b \rightarrow c$.

An $\mathbb{L}$-*set* is a mapping $X : U \rightarrow L$ 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 $X = \{{}^{l_1}/_{u_1}, {}^{l_2}/_{u_2}, \ldots, {}^{l_n}/_{u_n}\}$, 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 $u \in U$.

A fuzzy formal context $\mathbb{K}$ is a tuple (G, M, I) where G and M are non-empty sets, the so-called sets of objects and attributes, respectively, and $I : G \times M \rightarrow L$ is the so-called incidence relation. It is usually represented as a table, as in Table 1. There are two distinguished mappings that relate objects and attributes, commonly called concept-forming operators or derivation operators, which are defined as follows for each $X \in L^G, Y \in L^M$

$$X^{\uparrow}(m) = \bigwedge_{g \in G} (X(g) \rightarrow I(g, m)) \quad \text{and} \quad Y^{\downarrow}(g) = \bigwedge_{m \in M} (Y(m) \rightarrow I(g, m)).$$

The mapping $^{\uparrow}$ is also commonly called the intent operator and the $^{\downarrow}$ the extent operator.

A pair $\langle A, B \rangle \in L^G \times L^M$ is said to be a fuzzy formal concept if $A^{\uparrow} = B$ and $B^{\downarrow} = A$. The first component A is commonly called the extent of the concept $\langle A, B \rangle$ and the second component B is commonly called the intent of the concept $\langle A, B \rangle$. It is standard in

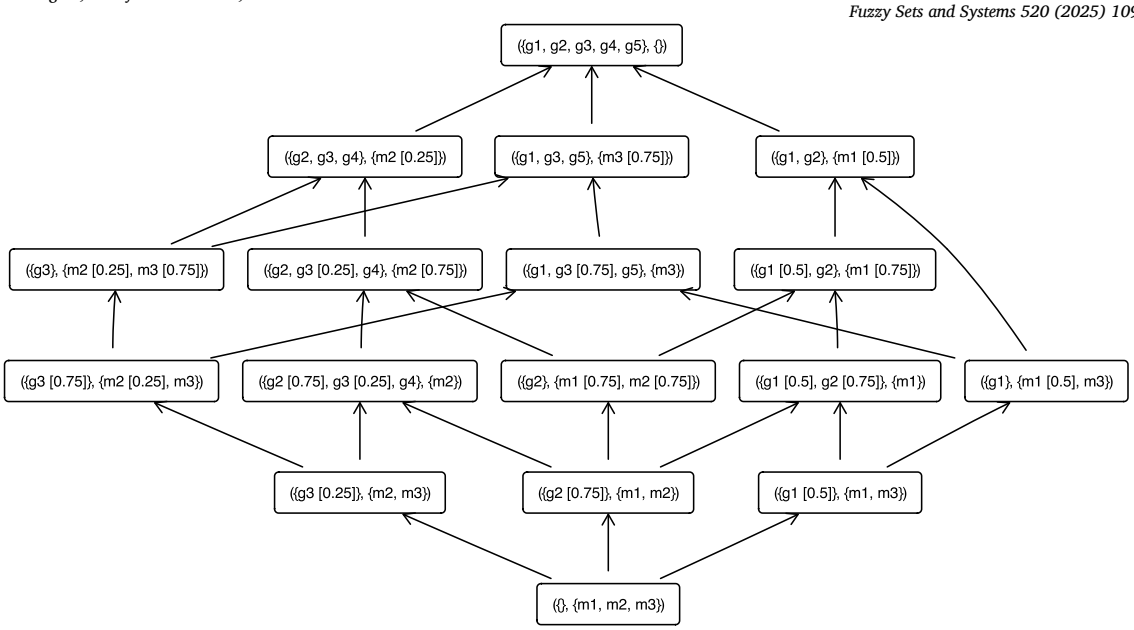

Fig. 1. Concept lattice of the formal context in Table 1.

FCA to identify a formal concept with its extent or its intent, treating them as the same entity. The set of all fuzzy formal concepts can be endowed with an order which turns out to be a complete lattice, the so-called fuzzy concept lattice (represented in Fig. 1 for the example context).

2.2. Algorithms for the fuzzy concept lattice

Within the framework of $\mathbb{L}$-fuzzy Formal Concept Analysis, prior studies have delineated two distinct strategies for computing the concepts of an $\mathbb{L}$-formal context.

- Proposals that scale fuzzy problems to enable crisp algorithms to handle them, rather than attempting to solve them directly. Although these proposals are interesting from a theoretical standpoint, they may not fully utilise the inherent value structure and all available information, potentially leading to inefficiencies.
  The main representative of this line of work is [9], where the idea was to perform a scaling operation (by modifying the derivation operators) in such a way that the resulting formal context could be analysed using classical (binary) algorithms. The steps proposed in this work can be summarised as follows:
  1. Use the *implication* operator $\to$ of the residuated lattice to transform the initial fuzzy context $\mathbb{K} = (G, M, I)$ into the context (G, P, J), where $P = M \times L$ and the new relation J is given by $J(g, \langle m, l\rangle) = l \to I(g, m)$. It should be noted that J also takes fuzzy values, since, particularly $J(g, \langle m, 1\rangle) = I(g, m)$.
  2. Construct the lattice $\mathbb{B}(G, P, J)$ by using an algorithm for determining the fixed points of a closure operator. To achieve this, it is sufficient to utilise an appropriate algorithm such as `FastCbO` or `InClose` on the newly created formal context, with the exception that the globalisation *hedge* will be incorporated in the derivation operators: $b^* = 1$ if and only if $b = 1$, otherwise $b^* = 0$ for any given attribute b.
  3. With the lattice thus constructed, it is sufficient to de-scale the intents computed so far to obtain the lattice of the initial context. Note that the lattices $\mathbb{B}(G, P, J)$ and $\mathbb{B}(G, M, I)$ are isomorphic, so no further transformation is required.

  In [9], the authors proposed the strategy above by applying the `FastCbO` and Lindig's `NextNeighbour` [33] algorithms.
- Proposals that function natively with fuzzy values are theoretically and conceptually related to algorithms that operate with crisp values. This ensures their correctness. Additionally, as they handle uncertain values naturally and properly, their performance is often superior.
  The two main proponents of this concept are the `NextClosure` algorithm [5] (an extension of the original by Ganter [22]) and Lindig's `NextNeighbour` [33], whose extension to attributes with degrees is presented in Bělohlávek's work [7]. In both cases, the search for closed sets is pruned to enhance efficiency.
  The `NextClosure` algorithm traverses the closed attribute sets in lexicographic order: after computing a given intent, the algorithm successively explores the attribute sets that follow the intent in lexicographic order, calculating their corresponding closures. Subject to certain technical constraints, the next intent is selected as the minimum closed set among these sets according to lexicographic order. Starting from the intent $\varnothing^{\downarrow\uparrow}$, all the intents are computed, and the corresponding extents are computed by using the $^{\downarrow}$ operator.

On the other hand, Lindig's algorithm operates under the premise of finding not only the collection of concepts but also the hierarchical relationship between them. Starting from a given concept, it modifies its intent by adding new attributes (one at a time), or increasing the truth value of attributes already present in the intent (again, one at a time), thus finding all immediate subconcepts of the original. If initiated with the concept $(G, \varnothing^{\downarrow\uparrow})$, it computes all concepts. Through exhaustive checking of the calculated concepts up to a given iteration, it additionally provides the conceptual hierarchy, automatically determining the upper and lower neighbours of each concept.

2.3. The CbO strategy

Due to its significance and relevance for the present manuscript, we will explain more detailedly the Close-by-One strategy [30]. Thus, the reader will be able to follow and understand more easily the following sections. The reader should recall that all the CbO-based algorithms are designed for the crisp case, and it is the purpose of this paper to present the extension of this strategy to the $\mathbb{L}$-fuzzy FCA framework.

The CbO family of algorithms can be considered the state-of-the-art in efficiency and speed of algorithms in the FCA setting because it takes advantage of the algebraic properties of the Galois connection. Since the extension of FCA to the fuzzy framework still makes use of a (fuzzy) Galois connection [6], we consider that extending these algorithms a natural choice.

As stated above, the CbO strategy takes advantage of the closure system structure, and proposes to operate directly on the set of extents, since, by that premise, the intersection of two extents is another extent. Thus, starting from a sufficiently big extent, in practice the set of all objects G, we can go through all extents by intersecting it with attribute-extents. This process can continue recursively until we have all the extents.

The Close-by-One (CbO) algorithm, introduced by Kuznetsov [30], is a prominent incremental algorithm for computing the set of all formal concepts of a given formal context (G, M, I). The main advantage that CbO offers with respect to a naive approach would be that of traversing extents only, rather than touring all fuzzy subsets of attributes and computing their closures via repeated composition of concept-forming operators.

The algorithm (composed by a main and an auxiliary function, shown in Algorithms 1 and 2) builds a search tree whose root is $\langle G, G^{\uparrow}\rangle$, representing the top formal concept. Here, G is the set of all objects and $G^{\uparrow}$ is the intent of all objects (i.e., the set of attributes common to all objects). New extents and their corresponding closed intents are computed at each node $\langle A, B\rangle$ in this tree, where A is always an extent and B is its closed intent. The algorithm systematically explores the concept lattice by considering attributes one by one in a predefined total order (e.g., $m_1 < m_2 < \cdots < m_{|M|}$).

From a node $\langle A, B\rangle$, for each attribute $j \in M$ that succeeds the last attribute considered for extending B along the current path, a new potential branch is explored. The algorithm first computes a candidate new extent $C := A \cap \{j\}^{\downarrow}$. This step efficiently determines the objects that possess both the attributes of the current extent A and the new attribute j. Subsequently, a candidate new intent $B_{candidate} = (B \cup \{j\})^{\downarrow\uparrow}$ is formed; this represents the closure of the current intent extended by attribute j.

A crucial canonicity test is then performed on $B_{candidate}$. This test checks if $B_{candidate}$ contains any attribute j' (short for $m_{j'}$) such that $j' \notin B$ and $j' < j$ (according to the predefined total order on attributes M). If such an attribute j' exists, then $B_{candidate}$ is considered non-canonical. This means that this particular path of adding attributes does not lead to a unique discovery of a concept (as it could have been reached via an earlier, lexicographically smaller attribute addition), and the branch is pruned. This condition ensures that each formal concept is returned exactly once by enforcing a lexicographical ordering on the attributes used to construct the intents. If the canonicity test is satisfied, implying a canonical path, then the corresponding closed intent $D := C^{\uparrow}$ is computed (which will be equivalent to $B_{candidate}$). The procedure is then re-initiated recursively from the new concept $\langle C, D\rangle$. If the canonicity test fails, there is no need to compute D, thus saving irrelevant computations. This recursive procedure guarantees that all formal concepts are discovered.

Algorithm 1: CbO($\mathbb{K}$).

Input: A formal context $\mathbb{K} = (G, M, I)$ where G is the set of objects, M is the set of attributes, and I is the incidence relation.
Output: A set of all formal concepts $\mathbb{C}$.

```
1 ℂ := ∅                                              // Initialize the set of formal concepts
  // Initial concept (Top concept: all objects, their common attributes)
2 A_top := G
3 B_top := G↑
4 Add (A_top, B_top) to ℂ
  // Start exploring from the first attribute (m_1)
5 ExploreConcepts(𝕂, A_top, B_top, 1, ℂ)
6 return ℂ
```

2.4. Computation philosophy of CbO-like algorithms

This section is a brief note on the philosophy employed in this algorithm's design. The algorithms presented in this paper adhere to a consistent design philosophy regarding their inputs, outputs, and overall structure. This approach aims to enhance clarity, modularity, and understanding of their respective roles in the computation of (fuzzy) formal concepts.

Algorithm 2: `ExploreConcepts`($\mathbb{K}$, A, B, y, $\mathbb{C}$).

Input: $\mathbb{K}$: the formal context; A: The extent of the current concept; B: The closed intent of the current concept; y: The index of the minimum attribute to consider for expansion; $\mathbb{C}$: The set of formal concepts (passed by reference).

```
1  for j from y to |M| do
       // Let m_j be the attribute M[j]
       // Compute candidate new extent
2      C := A ∩ {m_j}^↓
       // Compute candidate new intent for canonicity check
3      B_candidate := (B ∪ {m_j})^↓↑
       // Perform canonicity test: check if B_candidate is canonical
4      is_canonical := true
5      foreach attribute j' in B_candidate and j' < j do
6          if m_j' ∉ B then
7              is_canonical := false
8              break                                   // Not canonical, move to next attribute
9      if is_canonical then
           // Compute the closed intent for the new extent C
10         D := C^↑
11         Add ⟨C, D⟩ to ℂ
           // Recursively explore concepts from this new concept,
           // starting from the next attribute after m_j
12         ExploreConcepts(𝕂, C, D, j + 1, ℂ)
```

All algorithms are designed to operate on a formal context $\mathbb{K} = (G, M, I)$, which, in the fuzzy case, is defined over a complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$. This consistency ensures that the underlying fuzzy logic and algebraic structure are uniformly applied across all procedures.

Main algorithms: Procedures such as `CbO` serve as the primary entry points for the concept computation process. These algorithms take the (fuzzy) formal context $\mathbb{K}$ and, in the fuzzy case, the complete residuated lattice $\mathbb{L}$ as their direct and explicit inputs. They consistently return $\mathbb{C}$, which is defined as the comprehensive list of all fuzzy formal concepts of $\mathbb{K}$. Each concept in $\mathbb{C}$ is represented as a pair $\langle A, B\rangle$, where A denotes its extent and B its corresponding intent.

Auxiliary/helper algorithms: Functions like `ExploreConcepts` are designed to support the main algorithms by performing specific sub-tasks, often recursively. These helper functions receive the fuzzy formal context $\mathbb{K}$ and (again, in the fuzzy case) the complete residuated lattice $\mathbb{L}$ as contextual inputs. It is also of importance to notice that they also take the parent concept, typically represented by its extent A and intent B, and an attribute index y from which to initiate the expansion process for discovering child concepts. Furthermore, a global list $\mathbb{C}$ is passed *by reference*, serving as an accumulator for all newly discovered concepts. Unlike the main algorithms, these auxiliary functions do not explicitly return any value. They are executed for their side effects: they populate and modify the global list $\mathbb{C}$ by adding canonical child concepts as they are discovered, thus contributing to the overall collection of (fuzzy) formal concepts. They are also responsible for driving the recursive calls necessary for a complete lattice traversal.

In the next section, we propose fuzzy extensions for the state-of-the-art algorithms in FCA and prove their correctness. These extensions will be natively fuzzy so that the inherent structure of the truth-value structure can play a role in the optimisation of the code.

3. The CbO strategy in the fuzzy setting

In this section, the state-of-the-art algorithms for computing the set of concepts in FCA are extended to their fuzzy versions. In that, we will show a step-by-step perspective of their improvements concerning the others and prove their correctness. A similar analysis was developed by Konečny and Krajča in the crisp case in [27]. The main contribution of the paper is presented in this section, that is, the development of these extensions to the fuzzy framework in a way that outputs all the formal concepts. On a practical note, all the algorithms portrayed in this paper have been implemented in both `R` and `C` languages, to include them into the `fcaR` package [35].

As said in the preliminaries section, the `CbO` strategy allows us to cover all the extents of the formal context by intersecting an already computed extent with an attribute-extent, which results in an extent due to the closure system structure. In the particular case of the fuzzy setting, we have to consider the attribute-extents of each attribute with each truth value. This difference can be spotted in line 8 of Algorithm 4. In order to streamline the presentation of the manuscript, all algorithms are presented with a combined sweep, that is, when a concept is computed, all its child concepts are visited first, then the algorithm processes the corresponding subtrees in the same manner, hence combining the depth-first and breadth-first approaches. This is not a necessary feature, both DFS and BFS could be used on their own. Nevertheless, this combined strategy is employed so that the concepts are output in lexicographical order, which is a standard practice in FCA algorithms.

Remark 1. Throughout the paper we will consider a linear extension of the lattice order in L. In the algorithms, an index k will run over the truth values in L following this linear extension. Thus, $L[k]$ represents $l_k \in L$.

The FuzzyCbO function (Algorithm 3) serves as the primary orchestrator for computing all fuzzy formal concepts within a given fuzzy formal context $\mathbb{K}$. It acts as the initial entry point, setting the stage for a recursive exploration process that will systematically uncover every concept in the fuzzy concept lattice. This algorithm establishes the most general concept and then delegates the iterative discovery of subsequent concepts to its recursive helper function, FuzzyCbO_ChildConcepts.

Upon invocation, the algorithm first takes the fuzzy formal context $\mathbb{K}$ and the underlying complete residuated lattice $\mathbb{L}$ as input. Its initial task is to identify the *top* concept of the lattice: the pair $\langle G, G^{\uparrow}\rangle$ constitutes the most general concept, representing all objects sharing their common attributes. The entrance to the recursion step occurs in line 2, where the algorithm initiates the recursive concept exploration by calling FuzzyCbO_ChildConcepts. It passes the newly formed top concept $\langle G, G^{\uparrow}\rangle$ and a starting attribute index ($y = 1$), signaling the beginning of the attribute-wise search for concept extensions. Once the recursive calls to FuzzyCbO_ChildConcepts have completed, implying that all possible concepts have been identified and stored globally, the FuzzyCbO algorithm concludes by returning the comprehensive set $\mathbb{C}$, which now collectively encompass all extents and intents of the fuzzy formal concepts of $\mathbb{K}$.

Algorithm 3: FuzzyCbO($\mathbb{K}, \mathbb{L}$).

Input: $\mathbb{K}$: the fuzzy formal context (G, M, I) where G is the set of objects, M is the set of attributes and I is the fuzzy incidence relation taking values in $\mathbb{L}$, a complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$.
Output: $\mathbb{C}$: the set of all formal concepts.

1 $\mathbb{C} := \emptyset$
2 FuzzyCbO_ChildConcepts($\mathbb{K}$, $\mathbb{L}$, G, $G^{\uparrow}$, 1, $\mathbb{C}$)
3 **return** $\mathbb{C}$

The auxiliary function named FuzzyCbO_ChildConcepts is presented in Algorithm 4. This function begins by adding the current parent concept $\langle A, B\rangle$ to the global list of concepts. It then initializes an empty *queue* (line 2) to store newly discovered canonical child concepts that require further recursive exploration, and an index k (line 3) to iterate through the truth values in $\mathbb{L}$. A base case is established in line 4: if the parent intent B already covers all attributes M, or if the starting attribute index y exceeds the total number of attributes $|M|$), the function terminates.

The core of the algorithm resides in a nested loop structure (Lines 6-13). The outer loop, controlled by variable j, iterates through attributes in M starting from y up to $|M|$, ensuring a lexicographical order of attribute consideration crucial for CbO's uniqueness property. The inner loop, controlled by variable k (line 7), iterates through each fuzzy truth value $L[k]$ available in the lattice $\mathbb{L}$.

Inside the inner loop, a condition (line 8) checks if extending the current fuzzy intent B with attribute j at truth value $L[k]$ represents a valid progression (i.e., $B[j] < L[k]$). If this condition is met, candidate new extent C and intent D are computed using fuzzy concept-forming operators (lines 9-10). A fundamental canonicity test follows (line 11): it verifies whether the newly formed concept is canonical. This checks whether the restriction of D to attributes up to j (M_j), is lexicographically ordered correctly with respect to B, and also if $D[j]$ does not exceed $L[k]$. If the concept is canonical, the triple (C, D, j) is added to the *queue* (line 12). The inner loop then increments k to consider the next fuzzy truth value.

Finally, after exhausting all possible extensions at the current level, the algorithm iterates through each triple (C, D, j) in the *queue* (line 14). For each such triple, it makes a recursive call to FuzzyCbO_ChildConcepts, ensuring that all canonical child concepts are further explored and the entire fuzzy concept lattice is systematically traversed.

Algorithm 4: FuzzyCbO_ChildConcepts($\mathbb{K}, \mathbb{L}, A, B, y, \mathbb{C}$).

Input: A fuzzy formal context $\mathbb{K} = (G, M, I)$; a complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$; A: the extent of the parent concept; B: the intent of the parent concept; y: the index of the attribute to start the search from (1-based); $\mathbb{C}$: the global list of concepts (modified by reference).
Output: None (modifies $\mathbb{C}$ as a side effect).

1 Add $\langle A, B\rangle$ to the global list of concepts $\mathbb{C}$
2 *queue* $:= \emptyset$
3 $k := 1$
4 **if** $B = M$ **or** $y > |M|$ **then**
5 **return**
6 **for** j **from** y **to** $|M|$ **do**
7 **while** $k \leq |L|$ **do**
8 **if** $B[j] < L[k]$ **then**
9 $C := A \cap \{^{L[k]}/_{j}\}^{\downarrow}$
10 $D := C^{\uparrow}$
11 **if** $D \cap M_j \subseteq B \cap M_j$ **and** $D[j] \leq L[k]$ **then**
12 Add (C, D, j) to *queue*
13 $k := k + 1$
14 **foreach** (C, D, j) in *queue* **do**
15 FuzzyCbO_ChildConcepts($\mathbb{K}, \mathbb{L}, C, D, j + 1, \mathbb{C}$)

Remark 2. The notation M_i in Algorithm 4 is standard in the FCA literature devoted to CbO algorithms. It denotes the set of attributes of index lower than i.

Theorem 1. *Given a fuzzy formal context $\mathbb{K}$, the algorithm* `FuzzyCbO` *terminates by outputting all the formal concepts of $\mathbb{K}$.*

Proof. The `for` and `while` loops go over all the elements in M and L, respectively. Therefore, the algorithm covers all the extents and, since the intension is computed directly, we get all the formal concepts, ensuring its correctness. The only fact to check is that the canonicity test in line 11 of Algorithm 4 does not skip concepts.

Assume $D[j] = \alpha > L[k]$. Then, $D = (B \cup \{{}^{\alpha}/_{j}\})^{\downarrow\uparrow}$, which will be computed later in the algorithm when the `while` loop reaches $L[k] = \alpha$.

On the other hand, if we have $D \cap M_j \not\subseteq B \cap M_j$, there exists $i < j$ such that $B[i] < D[i] = \alpha$. This means that restricting to the first i attributes we get $D \cap M_{i+1} = (B \cap M_i) \cup \{{}^{\alpha}/_{i}\}$, but this has already been taken into account in the iteration where $j = i$, hence we have to skip it in order to avoid repeating the calculation.

Similarly, it is clear that whenever $B[j] \geq L[k]$ then we would get $C^{\uparrow} = D = B \cup \{{}^{L[k]}/_{j}\} = B = A^{\uparrow}$. Since A has already been computed we do not compute it again.

Now, let us turn our attention to the case where $B[j]$ and $L[k]$ are incomparable and prove that not considering this case in Algorithm 4 (line 8) does not remove any concept from the computation. Let $\alpha = B[j] \vee L[k]$. Then $B \cup \{{}^{L[k]}/_{j}\} = B \cup \{{}^{\alpha}/_{j}\}$. Thus, the computation $C = A \cap \{{}^{L[k]}/_{j}\}^{\downarrow} = (B \cup \{{}^{L[k]}/_{j}\})^{\downarrow}$ would provide the same result as in the future iteration where the degree $\alpha > B[j]$ is considered. Therefore, when studying the potential child branches of $\langle A, B\rangle$, it is not necessary to consider those elements in L that are not comparable to $B[j]$. □

Remark 3. Notice that the `for` loop in Algorithm 4 covers from attribute $y + 1$ to $|M|$. This serves the purpose of computing each intersection only once. Since intersection is commutative, it is the same to do $\{x\} \cap \{y\}$ or $\{y\} \cap \{x\}$. Thus, we order the attributes so that every intersection is done only in one direction.

Proving that `FuzzyCbO` is correct is one of the most important results in the paper since the remaining CbO-based algorithms can be obtained from it via adapting the pruning and canonicity tests to this setting. However, the `FuzzyCbO` shares the setbacks it had with its original (crisp) version. Thus, the following step in order to avoid unnecessary iterations is the notion of pruning. This was first introduced as pre-canonicity tests in the `FastCbO` algorithm [28,39] and has been called both ways throughout the literature [27]. The idea of pruning is to keep a list of fuzzy sets that have failed the canonicity test. This way, whenever we compute a new candidate we can compare it with the intents in the list to automatically know if it will fail the canonicity test and skip the iteration.

The `FastCbO`-version of `FuzzyCbO` is presented in Algorithm 5. At its essence, this algorithm embodies an optimized variant of the Close-by-One (CbO) method. The main difference with respect to `FuzzyCbO` is the utilization of an auxiliary data structure specifically designed to support the optimization strategy, which involves *blacklisting* redundant computations in subsequent steps.

Algorithm 5: `FuzzyFastCbO`($\mathbb{K}, \mathbb{L}$).

Input: A fuzzy formal context $\mathbb{K} = (G, M, I)$; A complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$.
Output: $\mathbb{C}$: the global list of all fuzzy formal concepts, represented as pairs $\langle A, B\rangle$.

```
1 ℂ := ∅
2 𝒩 := {𝒩_j | 𝒩_j = ∅ for all j ∈ M}
3 FuzzyFastCbO_ChildConcepts(𝕂, 𝕃, G, G↑, 1, 𝒩, ℂ)
4 return ℂ
```

The `FuzzyFastCbO_ChildConcepts` function (Algorithm 6) is a recursive helper function designed to generate fuzzy formal concepts, building upon the same logic of `FuzzyCbO_ChildConcepts` but incorporating key optimizations for improved performance. Its primary role is to systematically explore the fuzzy concept lattice and populate a global list of concepts ($\mathbb{C}$), passed by reference, as in the previous case.

The core differences and enhancements in `FuzzyFastCbO_ChildConcepts` lie in its strategy to prune the search space, which is often referred to as *blacklisting* redundant computations:

- **Auxiliary sets for optimization ($\mathcal{N}$):** This algorithm introduces an additional input parameter, $\mathcal{N}$, which comprises auxiliary sets of attributes. These sets are crucial for storing information about previously explored (and non-canonical) branches, enabling the algorithm to avoid re-exploring redundant parts of the lattice.
- **Optimized pruning condition:** Within its main loop, the algorithm employs a refined conditional check. Beyond verifying a valid progression in the fuzzy intent, it integrates a new condition, $\mathcal{N}_j \cap M_j \subseteq B \cap M_j$ (line 9), which consults the auxiliary sets. This condition effectively prunes branches of the search tree that are guaranteed not to lead to new canonical concepts, based on the accumulated *blacklist* information. It was proved in [39] that if, in the child concepts of $\langle A, B\rangle$, the algorithm analyses an attribute j for which $\mathcal{N}_j \cap M_j \not\subseteq B \cap M_j$, then the canonicity test will fail. The next result Theorem 2 states that this pruning, in the fuzzy setting, is still correct and, therefore, the algorithm is correct.

- **Conditional blacklisting update:** A significant enhancement is found in the block of lines 12-15. If a newly computed candidate concept is determined to be non-canonical (i.e., it fails the canonicity test), its intent (D) is used to update the local auxiliary set $\mathcal{M}_j$. This update effectively adds the non-canonical intent to the *blacklist* for the current attribute branch, informing future explorations and preventing redundant computations. The updated $\mathcal{M}$ (which holds these blacklisting sets) is then passed along in subsequent recursive calls.

In essence, while both algorithms traverse the fuzzy concept lattice using a CbO approach, `FuzzyFastCbO_ChildConcepts` uses the $\mathcal{N}$ structure and the dynamic updates to $\mathcal{M}_j$ to intelligently skip parts of the search space, leading to a more efficient computation of the complete set of fuzzy formal concepts.

Algorithm 6: `FuzzyFastCbO_ChildConcepts`$(\mathbb{K}, \mathbb{L}, A, B, y, \mathcal{N}, \mathbb{C})$.

Input: A fuzzy formal context $\mathbb{K} = (G, M, I)$; a complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \to, 0, 1)$; A: the extent of the parent concept; B: the intent of the parent concept; y: the index of the attribute to start the search from (1-based); $\mathcal{N} = \{\mathcal{N}_m\}_{m \in M}$: auxiliary sets of attributes for optimization; $\mathbb{C}$: the global list of concepts (modified by reference).
Output: None (modifies $\mathbb{C}$ as a side effect).

```
1  Add ⟨A, B⟩ to ℂ
2  queue := ∅
3  if B = M or y > |M| then
4      return
5  for j from y to |M| do
6      M_j := N_j
7      k := 1
8      while k ≤ |L| do
9          if B[j] < L[k] and N_j ∩ M_j ⊆ B ∩ M_j then
10             C := A ∩ {L[k]/M[j]}^↓
11             D := C^↑
12             if D ∩ M_j ⊆ B ∩ M_j and D[j] ≤ L[k] then
13                 Put (C, D, j) to queue
14             else
15                 M_j := D
16         k := k + 1
17 foreach (C, D, j) in queue do
18     FuzzyFastCbO_ChildConcepts(𝕂, 𝕃, C, D, j + 1, ℳ, ℂ)
```

Theorem 2. *The* `FuzzyFastCbO` *algorithm is correct, that is, it terminates and computes all the concepts of* $\mathbb{K}$.

Proof. The argument on termination and completeness are directly inherited from `FuzzyCbO` except for the pruning. Therefore, let us check that the pruning does not skip any concepts.

Assume $\mathcal{N}_j \cap M_j \not\subseteq B \cap M_j$, this means that there exists $m \in M$ such that $m \leq y$ and $\mathcal{N}_j(m) \not\leq B(m)$. Consider $C = A \cap \{{}^{L[k]}/_{j}\}^{\downarrow}$ and $D = C^{\uparrow}$. Then, since the algorithm covers intents increasingly, $\mathcal{N}_j \subseteq D$, and from that we deduce that $\mathcal{N}_j \cap M_j \not\subseteq B \cap M_j$ implies $D \cap M_j \not\subseteq B \cap M_j$. Thus, the intents skipped by `FuzzyFastCbO` would have failed the canonicity test, hence we save computation time. □

Although `FastCbO` has supposed a big leap in terms of computation time and intents-computing, the construction of an intent from the extent is still made via the concept-forming operator, which is a significantly demanding operation. In order to avoid this, and independently from `FastCbO`, Andrews [2] developed the `InClose` algorithm which computes closures in several steps called partial closures. In this paper, we will consider the enhancement of this algorithm, the so-called `InClose2`, introduced also by Andrews [3], with the main difference being that it utilises the combined sweep instead of the depth-first approach. Partial closures are enough to check the canonicity tests and simplify the computation in the case of non-canonical intents. The `FuzzyInClose2` algorithm (Algorithm 7) is another main contribution of this work, which builds on `FuzzyCbO` and the main difference would be the use of the aforementioned partial closures, at the cost of delaying the canonicity test. At a high level, the algorithm initializes an empty global collection $\mathbb{C}$ to store the discovered concepts. A key distinction for this variant lies in its initial setup of the *top* concept: while its extent is set to the full set of objects G, its intent is uniquely initialized as an empty set $\emptyset$. The intents will be computed incrementally, from a node to the descending branches, if the corresponding extent intersections coincide, as discussed later.

The `InClose2_ChildConcepts` algorithm (Algorithm 8) is a recursive helper function designed for fuzzy formal concept generation, embodying a specific strategy from the InClose family of algorithms. Unlike standard Close-by-One (CbO) approaches, InClose prioritizes the in-place extension of existing concepts when certain conditions are met, aiming to optimize the lattice construction process.

The algorithm operates by systematically exploring potential concept extensions based on a parent concept's extent (A) and intent (B), which are directly referenced in the global concept collection. For each attribute and fuzzy truth value, it computes a candidate new extent. A distinguishing feature lies in its handling of extent stability: if extending the parent's intent (or, equivalently, computing

Algorithm 7: `FuzzyInClose2`$(\mathbb{K}, \mathbb{L})$.

Input: $\mathbb{K}$: the fuzzy formal context (G, M, I) where G is the set of objects, M is the set of attributes and I is the fuzzy incidence relation taking values in $\mathbb{L}$, a complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$
Output: The set $\mathbb{C}$ of all formal concepts

```
1 A := G
2 B := ∅
3 InClose2_ChildConcepts(𝕂, 𝕃, A, B, 1, ℂ)
4 return ℂ
```

a new intersection $C := A \cap \{^{L[k]}/_{M[j]}\}^{\downarrow}$) does not alter the extent, the parent's intent (B) is directly updated in place. This avoids the creation of a new concept object if the extent is the same and instead evolves the existing one.

Let us explain this strategy in more detail: if $C = A \cap \{^{L[k]}/_{y}\}^{\downarrow} = A$ (for given k and y), then the corresponding $\{^{L[k]}/_{y}\}$ must belong to the intent of A, that is, to B. Due to the partial closures feature of InClose2, this is not yet contemplated in B so the algorithm adds it (line 10). On the other hand, if C is not A then it is a new extent, and its canonicity is checked (line 12) via the `IsCanonical` method (Algorithm 9). If it is indeed canonical, we define D as $B \cup \{^{L[k]}/_{j}\}$. Otherwise, this is not a canonical child (i.e., it is not in lexicographical order) of $\langle A, B\rangle$ and this iteration ends.

Conversely, if an intersection as above leads to a genuinely new extent, the algorithm applies a strict canonicity test. Only if this new candidate concept is deemed canonical, it is added to a list for further recursive exploration. This strategic in-place modification of intents and the selective creation of new concepts, guided by the canonicity check, are central to InClose2's efficiency in traversing and constructing the fuzzy concept lattice.

Algorithm 8: `InClose2_ChildConcepts`$(\mathbb{K}, \mathbb{L}, A, B, y, \mathbb{C})$.

Input: A fuzzy formal context $\mathbb{K} = (G, M, I)$; a complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$; A, B: pointers to the extent and the intent of the parent concept (referencing $\langle A, B\rangle \in \mathbb{C}$); y: the index of the attribute to start the search from .
Output: None (side effect: may modify the value of B, or may generate new pairs of extents-intents to be considered for recursion).

```
1  Add ⟨A, B⟩ to ℂ
   // B may be referenced and modified in this iteration or in any child branch. All updates will be made to the
      stored version in ℂ
2  candidates := ∅
3  if y > |M| then return
4  for j from y to |M| do
5      k := 1
6      while k ≤ |L| do
7          if B[j] < L[k] then
8              C := A ∩ {L[k]/M[j]}↓
9              if C = A then
10                 B := B ∪ {L[k]/M[j]}                          // Update B in-place
11             else
12                 if IsCanonical(A, B, C, j) then
13                     D := B ∪ {L[k]/M[j]}
14                     candidates := candidates ∪ {(C, D, j)}
15         k := k + 1
16 foreach (C, D, j) ∈ candidates do
17     InClose2_ChildConcepts(𝕂, 𝕃, C, D, j + 1, ℂ)
```

Algorithm 9: `IsCanonical`(A, B, C, j).

Input: A, B: extent-intent of the parent concept; C: extent of the candidate concept; j: the index of the attribute being processed (1-based).
Output: Boolean value: **true** if the candidate concept is canonical (i.e., their intents are in lexicographic order); **false** otherwise.

```
1 for m from 1 to j do
2     if B[m] < 1 then
3         l* := C↑[m]
4         if l* > B[m] then
5             return false
6 return true
```

Notice that `IsCanonical` checks that the intent corresponding to C up to the $(j-1)$-th attribute is a subset of B just like it did in `FuzzyCbO`, that is, it checks whether $C^{\uparrow} \cap M_{j+1} \subseteq B \cap M_{j+1}$, but computing the $C^{\uparrow}$ part only partially, as needed. This partial closure up to the j-th attribute is usually denoted as $^{\uparrow_{j+1}}$, as can be seen in [27]. It is relevant to mention that even though on line 3 of Algorithm 9 we write $C^{\uparrow}[m]$, we do not compute the intent and search for the m-th attribute, the actual computation will be

$$l^* = C^{\uparrow}[m] = \bigwedge_{g \in G} (C(g) \to I(g,m)),$$

which is much lighter to compute.

Theorem 3. *The* `FuzzyInClose2` *algorithm is correct, that is, it terminates and computes all the concepts of* $\mathbb{K}$.

Proof. Since the algorithm goes through all the attributes and all the truth values, it is clear that it goes through all the extents. Moreover, since the canonicity test `IsCanonical` coincides with the one in `FuzzyCbO`, every candidate concept in $\mathbb{C}$ is a pair $\langle C, D\rangle$ where C is an extent. We will prove that the final output is indeed a formal concept. For clarity, let $D^+ = C^{\uparrow}$ and we will prove $D = D^+$.

Since C is canonical, $D^+ \cap M_{j+1} = C^{\uparrow} \cap M_{j+1} = D \cap M_{j+1}$, that is, up to the j-th attribute, D and D^+ coincide. Now let $j' > j$, if $D[j'] < D^+[j']$ consider the truth value $L[k] = D^+[j']$. As the algorithm continues, the particular value k for truth values will be reached and we will get $C \cap \{{}^{L[k]}/_{M[j']}\}^{\downarrow} = (D^+ \cup \{{}^{L[k]}/_{M[j']}\})^{\downarrow} = D^{+\downarrow} = C$, thus the value of D will be updated to $D := D \cup \{{}^{L[k]}/_{M[j']}\}$. Otherwise, if $D[j'] = D^+[j']$, then for all $k \in \{1, \dots, |L|\}$ such that $D[j'] < L[k]$ we have $C \cap \{{}^{L[k]}/_{M[j']}\}^{\downarrow} \subsetneq C$ since $C \cap \{{}^{L[k]}/_{M[j']}\}^{\downarrow} = C$ is equivalent to $C \subseteq \{{}^{L[k]}/_{M[j']}\}^{\downarrow}$ and this would imply $\{{}^{L[k]}/_{M[j']}\} \subseteq \{{}^{L[k]}/_{M[j']}\}^{\downarrow\uparrow} \subseteq C^{\uparrow}$. In particular, $L[k] \leq \{{}^{L[k]}/_{M[j']}\}^{\downarrow\uparrow}[j'] \leq C^{\uparrow}[j'] = D[j']$, a contradiction.

Therefore, for each attribute $j' > j$, if $D[j'] = D^+[j']$, the value of D is not updated and whenever $D[j'] < D^+[j']$, the value of D is updated exactly to $D[j']$. The finiteness of the attribute set ensures that the final concept to be output will be $\langle C, D^+\rangle$. □

The last two strategies can merge into one algorithm. This is what `InClose3` did in the crisp case, published also by Andrews [4]. The results on termination and correctness for the fuzzy extension (`FuzzyInClose3`) are directly inherited from `FuzzyFastCbO` and `FuzzyInClose2`.

The strategies of a lighter canonicity test or an extra precanonicity test are directly extended to the `Fuzzy` versions from the crisp counterparts of `InClose4a`, `InClose4b`, and `InClose5`. Therefore, the correctness of these techniques is ensured by the correctness of `FuzzyInClose3` and the analysis already done in the crisp setting. Nevertheless, the truth value structure can also provide shortcuts to avoid repeated computation, this situation is motivated by the next example.

Example 1. Assume $L = \{0, 0.25, 0.5, 0.75, 1\}$ and let $\mathbb{K} = (\{o\}, \{a\}, I)$ be the following formal context:

I	a
o	0.75

Let us apply `FuzzyInClose2` (Algorithm 7) for simplicity and pedagogical reasons, although the same issue can be found in the newer versions of this algorithm. We start with the candidate concept $\langle\{o\}, \varnothing\rangle$:

1. The next attribute is a.
2. The next truth value is 0.25.
3. $C := \{o\} \cap \{{}^{0.25}/_{a}\}^{\downarrow} = \{o\}$.
4. $C^{\uparrow_1} = \{{}^{0.75}/_{a}\} \neq \{{}^{0.25}/_{a}\}$.
5. The canonicity test has failed.
6. The next truth value is 0.5, and the algorithm would continue with the next iteration.

Why should the next truth value be 0.5 when we already know that the iteration will be skipped? It makes sense to *skip* directly to 0.75. This intuitive idea of *skipping* is an original approach of this manuscript and is indeed a sound mathematical procedure whenever the residuated lattice is a finite chain. This allows to define a new strategy that can be incorporated into the fuzzy versions of both `FastCbO` and `InClose`. For simplicity, in the rest of the work we will consider this strategy applied only to `FuzzyInClose5`, expressed in a new procedure named `FuzzyInClose5*` (Algorithm 10) whose recursive part is listed in Algorithm 11.

This algorithm initiates the concept generation process by first establishing an empty container, $\mathbb{C}$, which will accumulate all the derived concepts. As in `FuzzyInClose2` (Algorithm 7), the starting top "concept" is $\langle G, \varnothing\rangle$. This specific initial state guides the subsequent concept derivation. The primary task of populating the concept lattice is then entrusted to a recursive auxiliary function, `InClose5*_ChildConcepts`. This helper is provided with the initial top concept, a starting point for attribute exploration, a special parameter $\mathcal{P}$ (initially set to an empty state for the first invocation, serving an optimization role related to path tracking and pruning), and the accumulating set $\mathbb{C}$. Following the exhaustive recursive computation, the algorithm performs a final verification step to explicitly add the bottom concept (characterized by an empty extent and a maximal intent) to $\mathbb{C}$, ensuring its inclusion as the recursive traversal is not able to generate it: (`Fuzzy`)`InClose5` keeps track of empty intersections and uses them to prune the expansion tree. Thus, if $\varnothing^{\uparrow\downarrow} = \varnothing$, the corresponding concept $\langle\varnothing, M\rangle$ is not generated and needs to be explicitly added afterwards.

The parameter $\mathcal{P}$ (a fuzzy vector) is the mechanism designed to prune the search space and enhance performance by avoiding redundant computations. The primary purpose of $\mathcal{P}$ is to store and propagate information about attributes and their associated truth values that have previously led to non-canonical concepts or empty extents during the lattice traversal. This information is then employed to avoid re-exploring branches of the search tree that are guaranteed not to yield new, valid concepts.

Algorithm 10: `FuzzyInClose5*`$(\mathbb{K}, \mathbb{L})$.

Input: $\mathbb{K}$: the fuzzy formal context (G, M, I); a complete residuated lattice $\mathbb{L} = (L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$.
Output: The set $\mathbb{C}$ of all fuzzy formal concepts, represented as pairs $\langle A, B \rangle$.

$\mathbb{C} := \emptyset$
$A_{top} := G$
$B_{top} := \emptyset$
$\mathcal{P} := \emptyset$
`InClose5*_ChildConcepts`$(\mathbb{K}, \mathbb{L}, A_{top}, B_{top}, 1, \mathcal{P}, \mathbb{C})$
if $\emptyset^{\uparrow\downarrow} = \emptyset$ **then**
  $\mathbb{C} := \mathbb{C} \cup \{\langle \emptyset, M \rangle\}$
return $\mathbb{C}$

The auxiliary `InClose5*_ChildConcepts` (Algorithm 11) representing the recursive part of `FuzzyInClose5*` relies on the `ProcessCurrent` function in Algorithm 12. For every combination of attribute j and truth value $L[k]$, `ProcessCurrent` acts as an oracle, evaluating the viability and canonicity of the potential concept extension. Its return value (`SkipAttr`, `JumpTo`(k'), or `Continue`) serves as a dynamic control signal, dictating the flow of the outer loops. A `SkipAttr` signal, often triggered by the $\mathcal{P}$ pruning set or an empty extent, acts as an immediate branch-pruner, indicating that further exploration along the current attribute's path is futile or redundant. Conversely, `JumpTo`(k') allows for an intelligent leap across the truth value lattice, skipping unnecessary intermediate checks when an optimization opportunity is identified. The `Continue` signal simply permits the exploration to advance incrementally.

Algorithm 11: `InClose5*_ChildConcepts`$(\mathbb{K}, \mathbb{L}, A, B, y, P, \mathbb{C})$.

Input: A: the extent of the parent concept; B: the intent of the parent concept (mutable); y: the index of the attribute to start the search from (1-based); $\mathcal{P}$: a set of attributes to guide the pruning (mutable, passed by reference); $\mathbb{C}$: the global collection of all formal concepts (mutable, passed by reference).
Output: None (modifies $\mathbb{C}$ by adding new concepts, and B in-place).

Add $\langle A, B \rangle$ to $\mathbb{C}$
$candidates := \emptyset$
if $y = |M|$ **then return**
for j **from** y **to** $|M|$ **do**
  $k := 1$
  while $k \leq |L|$ **do**
    result := `ProcessCurrent`$(\mathbb{K}, \mathbb{L}, A, B, j, L[k], \mathcal{P}, candidates, \mathbb{C})$
    if result = `SkipAttr` **then**
      break `// Break inner k-loop, effectively next j`
    else if result = `JumpTo`(k') **then**
      $k := k'$
    else
      $k := k + 1$
foreach $(C, D, j) \in candidates$ **do**
  `InClose5*_ChildConcepts`$(\mathbb{K}, \mathbb{L}, C, D, j + 1, \mathcal{P}, \mathbb{C})$

The `ProcessCurrent` auxiliary function (Algorithm 12) encapsulates the core logic for evaluating potential concept extensions originating from a given parent concept $\langle A, B \rangle$ when considering a specific attribute j at a particular truth value l, as was the case in previous algorithms. Its design is critical for the efficiency of `InClose5*`, as it incorporates several pruning and optimization strategies.

Initially, the function checks a crucial pruning condition: if $0 < \mathcal{P}[j] \leq l$. The parameter $\mathcal{P}[j]$ records a minimum truth value for attribute j that has previously led to an unpromising path (e.g., an empty extent or a non-canonical concept) in the previous path of the exploration tree. If the current truth value l for attribute j is greater than or equal to this recorded threshold, it signifies that further exploration down this specific branch for attribute j at or above l is redundant. In such instances, the function immediately returns `SkipAttr`, signaling the calling loop to abandon the current attribute and move to the next. This mechanism effectively prunes the search space, focusing computational effort on paths likely to yield novel and canonical concepts.

Should the pruning condition not be met, the algorithm proceeds to check if extending the parent's intent B with attribute j at truth value l actually represents a new potential addition ($B[j] < l$). If so, it computes a candidate extent C by intersecting the parent's extent A with the derivation of attribute j at degree l. An immediate check for an empty candidate extent C is performed: if $C = \emptyset$, this branch is deemed redundant – as the case $C = \emptyset$ will be computed and added explicitly in the main function `FuzzyInClose5*`. Thus, $\mathcal{P}[j]$ is updated with l (for attribute j) to blacklist this truth value for future pruning, returning `SkipAttr`.

Furthermore, the function calculates l^*, the exact fuzzy truth value of attribute j in the intent derived from the newly formed extent C. If l^* is strictly greater than the initial l, it indicates that a larger truth value could have been used to form C without changing its extent, so all intermediate values can be immediately skipped since they would all fail the canonicity test. This allows for an optimization: the function returns `JumpTo`(k'), instructing the outer loop to directly jump to the smallest index k' in the lattice L

such that $L[k'] \geq l^*$, effectively skipping redundant intermediate truth values. Therefore, the algorithm does not discard any formal concept and the completeness and correctness of the algorithm are inherited from the `FuzzyInClose5` algorithm.

A central element of InClose algorithms is the in-place update strategy. We recall that if the computed candidate extent C is identical to the parent's extent A, it means the extension did not produce a structurally new set of objects. In this scenario, the algorithm directly updates the parent's intent B by incorporating attribute j at degree l, thereby extending the existing concept in place. If C is indeed a genuinely new extent, a candidate intent D is formed. Note that C is effectively an extent, but D is just a candidate intent that may be updated down the path in the tree, in next recursion iterations. This new *candidate* concept pair $\langle C, D\rangle$ then undergoes a canonicity test via the `IsCanonical` function (Algorithm 9). Only if $\langle C, D\rangle$ is deemed canonical is it added to the candidates list for further recursive exploration. If the concept is not canonical, it means it would be discovered elsewhere or is redundant, and $\mathcal{P}[j]$ is updated with l to aid in future pruning. If none of the above conditions trigger a jump or skip, the function returns `Continue`, signaling a normal increment of the outer k loop.

Algorithm 12: `ProcessCurrent`$(\mathbb{K}, \mathbb{L}, A, B, j, l, \mathcal{P}, candidates, \mathbb{C})$.

Input: A: extent of the current parent concept; B: intent of the current parent concept (mutable); j: current attribute index; l: current truth value from lattice L; P: mutable pruning parameter vector; $candidates$: mutable list of child concepts for recursive calls; $\mathbb{C}$: mutable set of all formal concepts.
Output: Control flag: `SkipAttr` to break the current j-loop (outside this function), `JumpTo`(k') to increment the outer k variable to k', `Continue` for normal k increment.

```
if 0 < P[j] ≤ l then
    return SkipAttr
if B[j] < l then
    C := A ∩ {l/M[j]}↓
    if C = ∅ then
        UpdatePruningThreshold(P, j, l)
        return SkipAttr
    l* := C↑[j]
    if l* > l then
        k' := min{i : L[i] ≥ l*}
        return JumpTo(k')
    if C = A then
        B := B ∪ {l/M[j]}
    else
        if IsCanonical(A, B, C, j) then
            D := B ∪ {l/M[j]}
            candidates := candidates ∪ {(C, D, j)}
        else
            UpdatePruningThreshold(P, j, l)
return Continue
```

The `ProcessCurrent` function's pruning strategy relies heavily on the `UpdatePruningThreshold` auxiliary function (Algorithm 13). This critical function is responsible for dynamically updating the $\mathcal{P}$ set, which forms the backbone of the algorithm's proactive pruning mechanism.

Whenever `ProcessCurrent` identifies a scenario leading to an unviable or redundant concept, it invokes `UpdatePruningThreshold`. This function effectively records the pertinent truth value for the involved attribute within the $\mathcal{P}$ set. Its purpose is to store the most restrictive (lowest) truth value that has caused such a pruning event for that specific attribute. By doing so, `UpdatePruningThreshold` continuously refines the blacklisting knowledge that underpins the algorithm's efficiency, enabling it to avoid re-computing redundant or non-canonical concepts across the entire concept lattice exploration.

Algorithm 13: `UpdatePruningThreshold`$(\mathcal{P}, j, x)$.

Input: $\mathcal{P}$: mutable set for pruning thresholds; j: index of the attribute to update; x: the new truth value for the threshold.
Output: None (modifies $\mathcal{P}$ in-place).

```
if P[j] = 0 then
    P[j] := x
else
    P[j] := min{x, P[j]}
```

Even though the idea of skipping truth values seems valid, for qualitative data, the use of a discrete residuated lattice is necessary and this strategy could miss formal concepts. This is shown in the following example.

Example 2. Let $\mathbb{L} = \{\{0, x, y, 1\}, \wedge, \vee, \otimes, \rightarrow, 0, 1\}$, where $\otimes = \wedge$, the fuzzy formal context $\mathbb{K} = (\{o\}, \{a\}, I)$ and the residuum $\rightarrow$ are given in the tables below.

I	a
o	y

$\to$	0	x	y	1
0	1	1	1	1
x	0	1	y	1
y	0	x	1	1
1	0	x	y	1

Let us follow the algorithm with the linearization of the order $0 \preceq x \preceq y \preceq 1$, starting from the candidate concept $\langle o, \varnothing \rangle$.

1. The first attribute is a.
2. The first non-zero truth value is x.
3. $C_1 = \{o\} \cap \{{}^{x}/_{a}\}^{\downarrow} = \{{}^{y}/_{o}\}$.
4. $C_1^{\uparrow_1} = \{{}^{y}/_{o}\}^{\uparrow_1} = \{a\} \neq \{{}^{x}/_{a}\}$.
5. The canonicity test has failed and the truth value is updated to $l^* = 1$.
6. $C_2 = \{o\} \cap \{a\}^{\downarrow} = \{{}^{y}/_{o}\}$.
7. $C_2^{\uparrow_1} = \{{}^{y}/_{o}\}^{\uparrow_1} = \{a\}$.
8. The canonicity test is passed and $\langle \{{}^{y}/_{o}\}, \{a\} \rangle$ is a candidate concept.
9. Since the candidate intent is the attribute set, $\langle \{{}^{y}/_{o}\}, \{a\} \rangle$ is a formal concept and the algorithm terminates.

The algorithm has given as a result a single formal concept $\langle \{{}^{y}/_{o}\}, \{a\} \rangle$. Nevertheless, the pair $\langle \{o\}, \{{}^{y}/_{a}\} \rangle$ is also a formal concept which the algorithm has skipped.

The skipping truth values missing formal concepts can be avoided with the use of blacklisting, that is, instead of skipping directly to our candidate truth value, a set of distinguished truth values is defined in order to skip the iteration for all elements in the set. In particular, if the iteration of the truth value $L[k]$ provides via the canonicity test, a candidate truth value l^*, the blacklist will be the interval $[L[k], l^*)$, that is, the set $\mathcal{B} = \{l \in L \mid L[k] \leq l < l^*\}$. This way, all elements in $\mathcal{B}$ will be skipped, but truth values that are incomparable to $L[k]$ are not neglected.

The adaptation of `InClose5*_ChildConcepts` to this blacklisting scheme is simple by the introduction of the new variable $\mathcal{B}$. Algorithm 11 would feature a line between 4 and 5 "$\mathcal{B} := \varnothing$"; and between line 6 and 7 an additional condition would be added "**if** $L[k] \in \mathcal{B}$ **then next** k". Finally, the signature of `ProcessCurrent` would also have an argument for $\mathcal{B}$, and list an additional line (between lines 10 and 11) with "$\mathcal{B} := [L[k], l^*)$".

Theorem 4. *Let $\mathbb{L}$ be a complete finite residuated lattice and $\mathbb{K} = (G, M, I)$ be a fuzzy formal context. Then,* `FuzzyInClose5*` *with blacklisting terminates and computes all formal concepts.*

Proof. The termination and correctness are directly inherited from the `FuzzyInClose5` algorithm. Hence, it suffices to prove that blacklisting does not skip formal concepts.

Thus, assume the algorithm is running, the current attribute is j, the current truth value is $L[k] \in L$ and $l^* = C^{\uparrow}[j] > L[k]$. Then, $\mathcal{B} = [L[k], l^*)$. Let $L[n] \in \mathcal{B}$, we have that $\{{}^{L[k]}/_{j}\} \subseteq \{{}^{L[n]}/_{j}\} \subseteq \{{}^{l^*}/_{j}\}$, thus $\{{}^{l^*}/_{j}\}^{\downarrow} \subseteq \{{}^{L[n]}/_{j}\}^{\downarrow} \subseteq \{{}^{L[k]}/_{j}\}^{\downarrow}$ and

$$A \cap \{{}^{l^*}/_{j}\}^{\downarrow} \subseteq A \cap \{{}^{L[n]}/_{j}\}^{\downarrow} \subseteq A \cap \{{}^{L[k]}/_{j}\}^{\downarrow} = C.$$

Therefore,

$$C^{\uparrow}[j] = (A \cap \{{}^{L[k]}/_{j}\}^{\downarrow})^{\uparrow}[j] \leq (A \cap \{{}^{L[n]}/_{j}\}^{\downarrow})^{\uparrow}[j] \leq (A \cap \{{}^{l^*}/_{j}\}^{\downarrow})^{\uparrow}[j].$$

The left hand side is l^* by hypothesis.

The right hand side is also l^* since

$$(A \cap \{{}^{L[k]}/_{j}\}^{\downarrow})^{\uparrow}[j] = (B \cup \{{}^{L[k]}/_{j}\})^{\downarrow\uparrow}[j] = l^*,$$

hence $B \cup \{{}^{l^*}/_{j}\} \subseteq (B \cup \{{}^{L[k]}/_{j}\})^{\downarrow\uparrow}$ implies that

$$l^* \leq (A \cap \{{}^{l^*}/_{j}\}^{\downarrow})^{\uparrow}[j] = (B \cup \{{}^{l^*}/_{j}\})^{\downarrow\uparrow}[j] \leq (B \cup \{{}^{L[k]}/_{j}\})^{\downarrow\uparrow}[j] = l^*.$$

Thus, we have that

$$l^* \leq (A \cap \{{}^{L[n]}/_{j}\}^{\downarrow})^{\uparrow}[j] \leq l^*,$$

and the canonicity test would fail.

Therefore, every truth value between $L[k]$ and l^* would result in adding ${}^{l^*}/_{j}$ to the intent, thus it is redundant to compute it with any intermediate value once l^* is already known. □

Let us recall that the enforcement of canonicity tests in Close-by-One algorithms primarily serves to ensure the unique discovery of each formal concept. These tests are designed to prevent redundant computations that would arise from exploring *left* branches of

the search tree—those corresponding to attributes with smaller indices that would have canonically yielded the concept earlier. This mechanism ensures a systematic and non-repetitive enumeration of concepts.

However, a key enhancement introduced in this approach extends this pruning capability. While traditional canonicity focuses on avoiding redundancy stemming from prior (left-side) computations, the new blacklisting strategy, managed by the $\mathcal{B}$ set of truth values, prevents future redundant calculations. This proactive blacklisting allows the algorithm to skip all subsequent iterations for truth values that fall within $\mathcal{B}$, as they are guaranteed not to lead to the discovery of new canonical concepts. In addition, this mechanism ensures that elements incomparable to the current truth degree l within the residuated lattice are not erroneously neglected, preserving the correctness of the overall concept generation. This refined blacklisting strategy significantly enhances efficiency by intelligently bypassing entire ranges of truth values that would otherwise necessitate redundant computational effort.

This two-dimensional pruning, combining established canonicity for *left-side* avoidance with the new *right-side* blacklisting, enhances the overall efficiency of fuzzy concept lattice computation. By intelligently bypassing these additional categories of redundant calculations, the algorithm achieves improved performance and a more streamlined concept discovery process.

The trace of the execution of `FuzzyInClose5*` for the formal context of Table 1 can be found in Appendix A.

Corollary 5. *Let $\mathbb{L}$ be a finite chain, then the* `FuzzyInClose5*` *algorithm is correct, that is, it terminates and computes all the concepts of $\mathbb{K}$.*

Proof. It is an immediate consequence of the blacklisting method in Theorem 4. □

This case covers one of the most used cases in practice, where the residuated lattice is a discretization of the unit interval.

Notice that computing the formal concepts of a fuzzy formal context is still a #P-complete problem [29,31]. In addition, in the worst case scenario, all the algorithms presented degenerate to the `FuzzyCbO`, so they share the same theoretical complexity. However, the algorithms in this paper present promising results in the average-case scenario, which will be shown in the next section.

4. Experimental results

In this section, the outcomes of our empirical investigations into the efficiency and performance of the formal concept analysis algorithms are presented. The algorithms analyzed are InClose2$_w$, the wrapper method that scales the fuzzy formal context, applies the crisp InClose2 algorithm and then descales the formal concepts obtained [9], the fuzzy NextClosure algorithm [5], and the proposed algorithm in this paper, FuzzyFastCbO, FuzzyInClose2, FuzzyInClose5 and FuzzyInClose5*. Since all the experiments are done in the fuzzy framework, we will omit the prefix Fuzzy in the algorithms, thus clarifying the presentation of the results and the charts. Two types of datasets have been used in the experimental evaluation: synthetic (random) contexts, with various configurations, characterised by different attributes and object set sizes, extent sizes, and context densities; and real-world datasets, obtained from the UCI Machine Learning Repository [19].

First, some general considerations about the metrics used in the experiments and the implementation of the algorithms are presented. Next, the particular results for each kind of dataset will be described in detail, together with their corresponding discussions.

4.1. Performance metrics

Our study aims to comprehensively evaluate the efficiency of the algorithms. To achieve this, a set of performance metrics designed to capture various aspects of algorithm behaviour has been selected:

- **Number of canonicity tests**: Determining the number of canonicity tests conducted by each algorithm allows us to assess their optimisation strategies. A lower count indicates a more optimised algorithm. In our plots and tables, we denote this metric as "#tests".
- **Number of attribute intents computed** (computations indicated by expressions such as $X^{\uparrow}[j]$ in the pseudocodes above): Analysing how many attribute intents are calculated is essential for understanding the computational demands of the algorithms. Since this metric is highly dependent on the number of attributes, we normalise the metric by dividing the number of intents by $|M|$. This metric is denoted as "#intents" in our experimental results.
- **Execution time**: Measuring the runtime of the algorithms provides valuable insights into their efficiency. This metric helps us understand how the algorithms handle increasing data sizes and complexity.

4.2. Implementation

All the algorithms studied in this comparison have been developed and tested using the framework provided by the `fcaR` library [35], available for the `R` programming language. The core functions were implemented in pure `C`, while higher level procedures in the experimental analysis were done in `R`. This means that all of them have used the same data types and structures, so no performance difference can be attributed to the implementation.

4.3. Synthetic datasets

A first study uses synthetically generated datasets. The following subsections present the mechanism to generate the formal contexts used in this evaluation, the particular objectives of the study and the obtained results.

4.3.1. Dataset generation

For a comprehensive evaluation of the algorithms, the following parameter settings have been explored:

- Object set size, $|G|$, varied across values of $25, 50, 75, 100, 125, 150$. This range allows us to examine the algorithms' behaviour under varying data volumes.
- Attribute set size, $|M|$, was tested with dimensions in 10, 25, 50, 75, 100, 125, 150, 175, and 200, covering a broad spectrum of complexity.
- We introduced different valuation lattices L_n, for n in 2, 4, 8, 12, 16, 20, 24, 28, 32. These values allow us to investigate how the algorithms perform under different lattice complexities.
- Context density, the ratio of non-zero entries, symbolised by δ, was explored at four different levels, $\{0.1, 0.25, 0.5, 0.75\}$, offering insights into the impact of context sparsity on algorithm performance.

To ensure statistical significance and reliability, ten random contexts (and, therefore, with random attribute orders) for each of the configurations mentioned above were generated. This extensive dataset enables a robust analysis of the algorithms across various scenarios.

4.3.2. Objectives

The primary objective of our study is to compare and contrast the different algorithms based on the performance metrics mentioned above. By doing so, we aim to determine the efficiency of each algorithm.

Additionally, another objective closely related to the above has been set: to investigate how key parameters, namely the number of attributes ($|M|$), the size of the lattice L_n, and the density of the context (δ), influence the performance metrics for every algorithm. This analysis allows us to discern the impact of these crucial factors on the behaviour of the algorithms and provide guidance on optimising their use in different scenarios.

This comparative analysis offers valuable insights into the strengths and weaknesses of each algorithm, guiding researchers and practitioners in selecting the most appropriate approach for their specific use cases.

4.3.3. Results

Here, the data obtained in our experimental tests is presented. Note that, for the sake of clarity, in all the plots the vertical axis has been logarithmically scaled.

Impact of the cardinality of the attribute set M

In terms of the quantity of canonicity tests carried out by the algorithms, `NextClosure` and $\texttt{InClose2}_w$ produced identical results, with `InClose2` and `FastCbO` not far behind, see Fig. 2 (left). This places `InClose5` (incorporating the pruning techniques from `InClose2` and `InClose4`) and `InClose5`* as the most effective algorithms, needing less tests to complete the tree. The efficacy of `InClose5`* in all our experiments can be credited to its utilisation of the *fast noncanonicity check* technique, which enables skipping certain values in the lattice L during the construction of the search tree.

Based on the computed number of attribute intents (Fig. 2, right), it can be concluded that the trends are similar. However, it should be emphasised that even though `NextClosure` seems to outperform $\texttt{InClose2}_w$ and `FastCbO` in terms of efficiency, this is a consequence of `FastCbO` computing extra intents for testing, which results in a larger amount of intents computed but a shorter overall runtime of the algorithm. For large values of M, the latter algorithm is the least efficient in terms of intent computation, nevertheless we will see that in terms of runtime it outperforms both `NextClosure` and $\texttt{InClose2}_w$.

Based on the data collected, it is possible to hypothesise that the relationship between the number of canonicity tests or intents and the cardinality of M adheres to a power law. To validate this hypothesis, we conducted a regression analysis using the equation $y = x^a$, yielding the findings shown in Table 2. The results indicate that there is a power law that governs the relationship between performance metrics and attribute set cardinality. By utilising the exponents obtained from these fittings, recalling that lower exponents indicate a more efficient approach, a discussion with conclusions analogous to those previously noted can be conducted.

Impact of the cardinality of the valuation lattice L

Starting with `NextClosure`, we can observe in Fig. 3 that this algorithm consistently requires the highest number of canonicity tests among all the algorithms, along as $\texttt{InClose2}_w$. As the size of L_n increases, both algorithms show a significant increase in the number of tests, emphasizing their sensitivity to data complexity.

Moving on to `FastCbO`, we can see that it conducts fewer canonicity tests compared to `NextClosure`, on a par with `InClose2`, maintaining efficient computational performance even as the lattice L_n size grows.

The results shown in Fig. 3 demonstrate that `InClose5`* is the most efficient algorithm for computing concept lattices when varying the cardinality of L_n, as it is evident from the facts that `InClose5`* has the lowest number of canonicity tests and attribute intent computations.

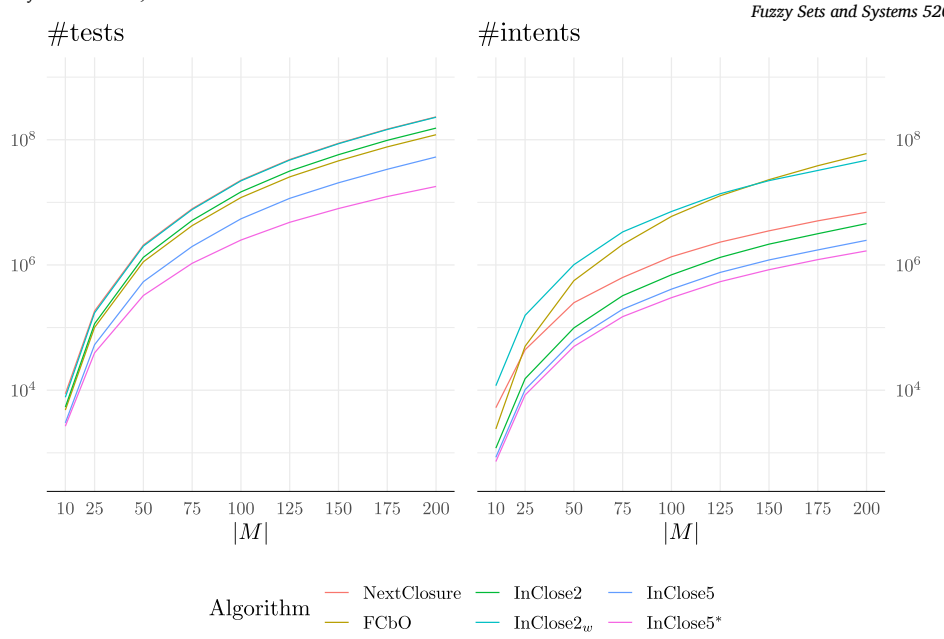

Fig. 2. Evolution of the performance metrics according to the cardinal of M.

Table 2
Power law fitted to the distribution of the number of canonicity tests and intents as a function of $|M|$.

	#tests		#intents	
Algorithm	Equation	R^2	Equation	R^2
NextClosure	$\|M\|^{3.6763}$	0.9997	$\|M\|^{4.0744}$	0.9987
FastCbO	$\|M\|^{3.5355}$	0.9999	$\|M\|^{4.3815}$	1.0000
InClose2	$\|M\|^{3.5804}$	0.9999	$\|M\|^{3.9235}$	0.9999
InClose2$_w$	$\|M\|^{3.6694}$	0.9998	$\|M\|^{4.4414}$	0.9989
InClose5	$\|M\|^{3.3668}$	0.9999	$\|M\|^{3.8068}$	0.9999
InClose5*	$\|M\|^{3.1978}$	0.9997	$\|M\|^{3.7385}$	0.9999

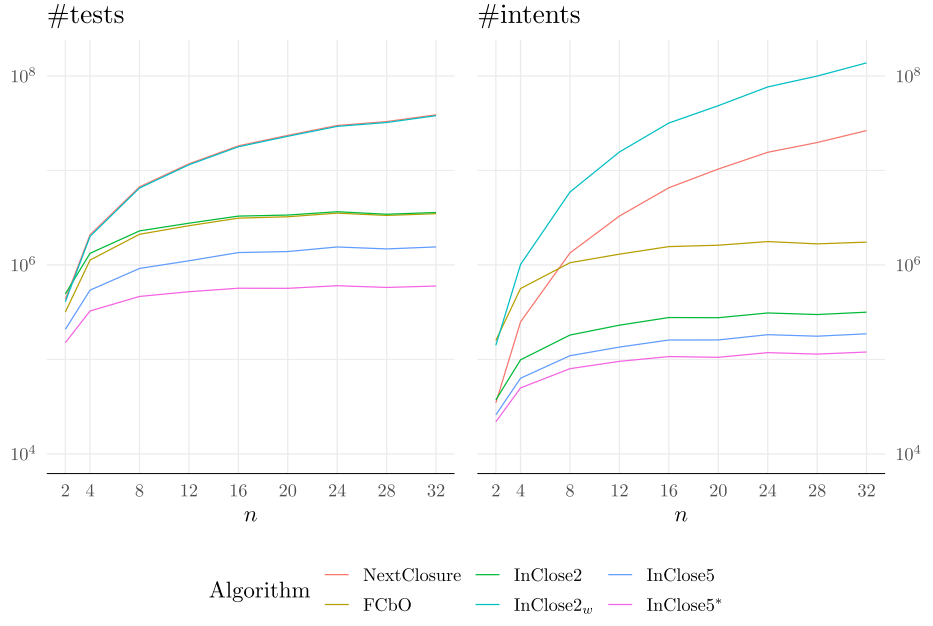

Fig. 3. Detected trends in the metrics as a function of n, the number of non-zero elements in L_n.

Table 3
Power law fitted to the distribution of the number of canonicity tests and intents as a function of $n = |L_n| - 1$.

	#tests		#intents	
Algorithm	Equation	R^2	Equation	R^2
NextClosure	$n^{5.9314}$	0.9344	$n^{6.9823}$	0.9461
FastCbO	$n^{5.2812}$	0.9138	$n^{6.4291}$	0.9088
InClose2	$n^{5.3039}$	0.9093	$n^{5.8131}$	0.9091
InClose2$_w$	$n^{5.9224}$	0.9348	$n^{7.5404}$	0.9440
InClose5	$n^{4.9916}$	0.9112	$n^{5.6276}$	0.9080
InClose5*	$n^{4.6936}$	0.9035	$n^{5.4886}$	0.9050

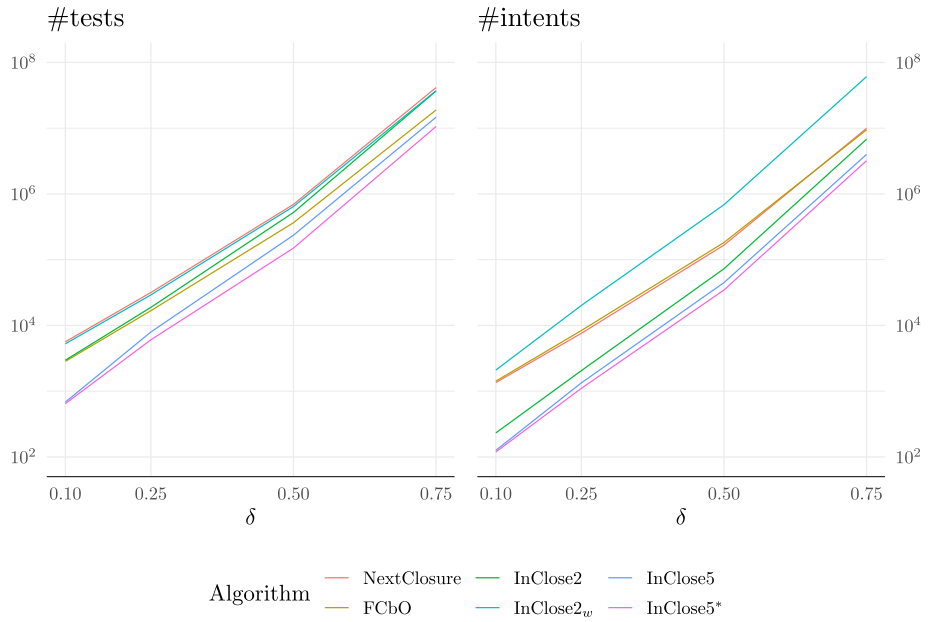

Fig. 4. Impact of the context sparsity on the number of canonicity tests and on the computation of attribute intents. δ denotes context density, that is, the proportion of non-zero elements in the context table.

This study emphasises the significant advantage of implementing the *fast noncanonicity check* technique in `InClose5*` while examining larger lattices. The difference in efficiency between `InClose5*` and the baseline algorithms (more precisely, `NextClosure`) becomes more pronounced as the cardinality of L increases.

Regarding the number of computed attribute intents, although the trends displayed in Fig. 3 (right) are similar to those previously mentioned, some particularities should be highlighted. Although `InClose2`$_w$ and `NextClosure` seem to have the same rapidly-increasing logarithmic behaviour, the number of intents calculated by the *wrapped* version of `InClose2` is significantly higher than that of `NextClosure`, making it, in fact, the least efficient algorithm in this specific aspect.

The next algorithm in order of efficiency concerning the number of intents is `FastCbO`. Its behaviour is akin to all `InClose` algorithms but less efficient due to the strategy of calculating closures at each node. It is worth mentioning that for $n \in \{2,4\}$, `NextClosure` requires fewer calculations of attribute intents than `FastCbO`.

Finally, the `InClose` algorithms exhibit a trend analogous to the one found in the number of canonicity tests, with `InClose5*` being the most efficient among them. These conclusions can also be drawn, as in the previous section, from the power law fit made to the data, presented in Table 3, recalling that the algorithm would be more efficient if the exponent is lower.

Impact of the context density δ

In Fig. 4, the influence of formal context density can be observed in algorithm performance. It should be noted that the vertical axis on the graphs is logarithmic in scale. It can be confirmed that the relationship between the number of canonicity tests and the number of calculated intents, with respect to density δ, is approximately exponential.

Thus, `InClose2`$_w$ consistently exhibits the poorest performance in both metrics (tied with `NextClosure` in number of canonicity tests), while `InClose5*` is the best, ranking at the opposite end of the spectrum. It is noteworthy that while `FastCbO` equals `NextClosure` in number of intent computations, this is not reflected in the number of tests, where `FastCbO` is able to outperform `InClose2`, which generates significantly more canonicity tests.

Table 4
Exponential fit of the distribution of the number of canonicity tests and intents as a function of δ.

	#tests		#intents	
Algorithm	Equation	R^2	Equation	R^2
NextClosure	$e^{26.3471\,\delta}$	0.9132	$e^{29.5864\,\delta}$	0.8940
FastCbO	$e^{25.0601\,\delta}$	0.9198	$e^{29.6265\,\delta}$	0.8920
InClose2	$e^{25.8651\,\delta}$	0.9264	$e^{28.2300\,\delta}$	0.9258
InClose2$_w$	$e^{26.1769\,\delta}$	0.9137	$e^{32.2486\,\delta}$	0.9029
InClose5	$e^{24.2321\,\delta}$	0.9457	$e^{27.3208\,\delta}$	0.9321
InClose5*	$e^{23.6247\,\delta}$	0.9452	$e^{26.9280\,\delta}$	0.9319

Table 5
Runtime (in seconds) of each algorithm for the configuration of $|G| = 75$ objects and varying number of attributes and different L_n lattices. The context density was fixed at 0.25 for these results.

$\|M\|$	L_n	Concepts	NextClosure	FastCbO	InClose2	InClose2$_w$	InClose5	InClose5*
50	L_2	14 551.8	0.352	0.118	0.034	0.065	0.029	0.025
	L_4	34 803.8	1.559	0.349	0.080	0.325	0.067	0.060
	L_8	57 559.0	4.852	0.649	0.137	1.437	0.109	0.095
75	L_2	37 961.2	1.532	0.512	0.117	0.270	0.101	0.086
	L_4	93 276.0	6.701	1.670	0.296	1.427	0.234	0.198
	L_8	165 637.2	23.493	3.708	0.562	7.055	0.425	0.349
100	L_2	69 983.0	4.149	1.479	0.283	0.680	0.231	0.193
	L_4	188 331.6	21.505	5.842	0.796	4.001	0.597	0.473
	L_8	340 043.4	76.039	13.407	1.543	20.812	1.110	0.855
125	L_2	118 358.2	9.871	3.742	0.584	1.462	0.461	0.372
	L_4	325 776.2	50.845	15.314	1.731	9.581	1.252	0.959
	L_8	567 660.2	165.340	34.116	3.145	46.477	2.214	1.657

As noted earlier, the relationship between these metrics and contextual density can be modelled using an exponential function based on the data we collected. To confirm this, we fitted the data to an equation of the type $y = e^{a\delta}$, and the results are presented in Table 4. The coefficients analysis (lower is better) supports our previous findings.

Analysis of execution time

Finally, Table 5 presents the average execution times of the different algorithms for one of the tested configurations. It is evident that the number of canonicity tests and attribute intents parameters studied earlier play a crucial role in determining the algorithm's performance, as they have a high computational cost. Therefore, the discussion of this table is similar to the already presented results. The `NextClosure` algorithm consistently gathered the greatest number of tests in the experiments conducted, and it also takes the longest time in all scenarios.

The new algorithms exhibit enhanced efficiency when contrasted with the classical `NextClosure`, considering both the number of computed intents (as already pointed out) and execution time. As far as `FastCbO` is concerned, it is perceptible that it entails a substantially greater number of intents to be computed when juxtaposed with the studied version of `InClose`. The lengthy execution time of `FastCbO` can be attributed to its computation of the closure of the attribute set at each node in the search tree, which necessitates calculating an extent for its respective intent. Conversely, `InClose` only computes the intents necessary for the canonicity test, with extents being effortlessly computed incrementally. This fundamental difference is the reason why `FastCbO` takes longer to execute.

If we compare `InClose2` to its "wrapped" version, `InClose2`$_w$, we can observe how the computation of an exceptionally high number of intents by `InClose2`$_w$ significantly affects its execution time.

It is remarkable that the pruning in `InClose5` and the *fast noncanonicity check* in `InClose5`* are the reasons for their high efficiency. Being able to remove redundant computations, with lower number of canonicity tests and attribute intents, the algorithms became more efficient and are able to tackle larger problem sizes.

To provide a sense of the improvement achieved, the `NextClosure` algorithm for the configuration with $|G| = 100$ objects, $|M| = 150$ attributes, and values in L_8, took 889.598 seconds and computed 7 722 482 500 intents, while `InClose5`* only computed 504 365 635 intents in 8.116 seconds.

4.4. Real datasets

Although the preceding section presented the outcomes in a variety of scenarios, this section is devoted to the evaluation of the algorithms on real-world datasets.

4.4.1. Dataset description

Five datasets from real-world problems have been used in the experiments:

Table 6
Properties of the datasets used in this experimentation.

Dataset	$\|G\|$	$\|M\|$	Density	Concepts
COBRE	105	32	0.176	14 706
HCV	589	10	1	835 115
ILPD	579	8	1	56 295
LEAF	340	14	1	1 796 124
ACCENT	329	12	1	1 455 641

Table 7
Results on real datasets.

	#tests					
Dataset	NextClosure	FastCbO	InClose2	InClose2$_w$	InClose5	InClose5*
COBRE	795 800	408 596	430 882	758 433	144 015	68 117
HCV	3 696 119	1 752 112	2 130 919	1 813 129	2 039 641	2 037 413
ILPD	246 687	117 458	143 031	121 489	136 793	136 793
LEAF	10 166 870	4 231 302	7 342 310	5 258 693	5 354 512	5 295 428
ACCENT	8 101 675	4 026 302	6 195 662	4 853 810	4 687 348	4 538 058
	#intents					
Dataset	NextClosure	FastCbO	InClose2	InClose2$_w$	InClose5	InClose5*
COBRE	4 774 800	6 537 568	1 100 743	12 272 968	691 352	426 918
HCV	22 176 714	8 760 570	8 671 667	78 396 455	8 491 663	6 894 656
ILPD	1 480 122	469 840	475 983	4 209 459	462 018	372 044
LEAF	61 001 220	29 619 128	37 099 801	293 707 173	27 928 160	23 839 816
ACCENT	48 610 050	24 157 824	24 684 757	200 549 057	20 345 993	17 600 692
	Time (sec.)					
Dataset	NextClosure	FastCbO	InClose2	InClose2$_w$	InClose5	InClose5*
COBRE	0.434	0.081	0.038	0.102	0.022	0.015
HCV	40.724	10.017	9.433	40.240	9.924	6.578
ILPD	2.105	0.527	0.513	2.140	0.474	0.344
LEAF	77.853	15.228	18.199	82.767	15.045	10.211
ACCENT	53.271	12.483	13.273	57.093	11.077	8.135

- **COBRE**: This dataset is provided by the `fcaR` library [35]. It is a collection of scores from several cognitive assessment instruments for patients with schizophrenia and bipolar disorder.
- **HCV**: Laboratory values of blood tests from blood donors and hepatitis C patients [32].
- **ILPD**: Dataset encompassing 584 patient records obtained from the NorthEast of Andhra Pradesh, India. These records include measurements of several biochemical markers, such as albumin and liver enzymes, which play a crucial role in metabolic processes [42].
- **LEAF**: Dataset of shape and texture features extracted from digital images of leaf specimens. These specimens represent a total of 40 distinct plant species [45].
- **ACCENT**: The data set comprises a series of single English words, read by speakers from six different countries, with the objective of facilitating the detection and recognition of accents. The features are derived from the spectral measures of the sound waves corresponding to each pronounced word [1].

Note that the first dataset, COBRE, inherently possesses a fuzzy formal context structure with values in L_4. For the remaining datasets, each numerical feature undergoes discretization into four intervals corresponding to quartiles. Formally, a value $V(i,f)$ (i representing the instance, f the feature index), is mapped to $\frac{j}{4}$, for $j \in \{1,2,3,4\}$, if $Q_{j-1}(f) < V(i,f) \leq Q_j(f)$, indicating its placement within the j-th quartile-interval for that specific feature. Thus, their corresponding formal contexts are also valuated in L_4. This discretization process transforms continuous numerical features into *ordered* categories of approximately the same size, capturing the statistical distribution of values, and making them more suitable for FCA. The number of intervals, 4, has been selected taking into account the balance between minimizing information loss while maintaining representativity of the results. An overview of the datasets properties, including their densities and corresponding number of formal concepts, is presented in Table 6.

4.4.2. Results

The six algorithms tested in the preceding section have been used to obtain the set of formal concepts for each of the real-world datasets. The metrics used to compare the performance of the algorithms are the same as in the case of synthetic datasets: number of canonicity tests, number of attribute intents and execution time. The results for this experimental evaluation are presented in Table 7.

Across all datasets, NextClosure has the highest number of canonicity tests, followed by InClose2$_w$, and InClose5* needing the lowest number of such tests. This suggests that NextClosure and InClose2$_w$ are involved in more canonicity computations compared to

other algorithms, and that InClose5* has the most advanced pruning strategy. Similar to the number of tests, NextClosure generally has the highest number of intents across all datasets, followed by InClose2$_w$, again. This potentially suggests more redundant computations in NextClosure and InClose2$_w$. Lastly, in terms of execution time, the same conclusions can be drawn: the fuzzy versions of the CbO family of algorithms obtain the set of concepts in less time than NextClosure or the scaled version of InClose2. Notably, InClose5* is able to obtain the results faster than the other algorithms. In summary, the native fuzzy versions of CbO-like algorithms in this evaluation demonstrate superior performance compared to other alternatives. This is attributed to the more efficient pruning strategy, which reduces the occurrence of duplicated computations in the search tree. In general, these results corroborate our observations from the evaluation of the synthetic datasets, with InClose5* being the most effective across all problem instances.

5. Conclusions and future work

In this paper, we have presented extensions of the most-cited CbO-based algorithms for computing the concept lattice to the fuzzy framework. In particular, the goal has been to implement native fuzzy approaches instead of applying the crisp algorithms to a scaling of the fuzzy formal context. Thus, the algorithms are presented and proved to be correct. In addition, experimental results are portrayed to showcase the performance enhancement that these methods present compared to scaling. As a matter of fact, not only are the fuzzy methods more optimal than the ones obtained via scaling, but there are also properly fuzzy strategies that aid speed up the algorithms, it is the case of "skipping" which we have presented at the end of Section 3.

As a prospect of future work, on a theoretical note, the aspects of average-case complexity and ramifications of the worst-case scenario are still open problems. On a more practical note, we intend to study the role that the ordering of the attributes plays in the speed of performance. As a matter of fact, the doubly-lexically ordered matrices can also play a role in optimising the runtime of the algorithms. Hence, these two strategies will be considered and thoroughly studied in order to make the computations feasible in applications.

Some applications may benefit from the contributions of this paper, since these new algorithms can help in extracting knowledge in real-world problems, as confirmed by the experimental evaluation in Section 4. For instance, the analysis of social networks [24, 37,36], the construction of recommender systems [15,13] or its application to more general frameworks, such as the multi-adjoint framework [17,16], among other fields of application. The extraction of logical rules and association rules can also be seen as a significant step forward in this area, therefore some research lines will be devoted to adapt and to improve these methods to find rules in the fuzzy setting.

It is noteworthy the increasing interest in studying the applications of FCA to data streams since, nowadays, these conform one of the main sources of information in the connected world. There have been some works that analyze the incremental construction of the concept lattice in the classical binary setting [26]. However, the study of data streams require another perspective and mechanisms to update and modify an existing concept lattice to detect *changes of concept* or *concept drifts* [20]. In this sense, the modelization of data streams with FCA will be a promising line of work, providing robust and interpretable formal methods specifically adapted to the properties of data streams.

CRediT authorship contribution statement

Domingo López-Rodríguez: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Manuel Ojeda-Hernández:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Ángel Mora:** Writing – review & editing, Validation, Formal analysis, Conceptualization. **Carlos Bejines:** Writing – review & editing, Investigation, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been partially funded by the State Agency of Research (AEI), the Ministerio de Ciencia, Innovación y Universidades (MCIU), the European Social Research Fund (FEDER), the Junta de Andalucía (JA), and the Universidad de Málaga (UMA) through the Postdoctoral Scholarship JCSMK24-0053 (Kempe Foundation), the VALID research project (PID2022-140630NB-I00 funded by MCIN/ AEI/ 10.13039/ 501100011033) and the research project PID2021-127870OB-I00 (MCIU/AEI/FEDER, UE).

Appendix A. Trace of `FuzzyInClose5*`

In this section, the trace of the execution of the `FuzzyInClose5*` algorithm on the example of Table 1 is presented. This example has been selected as it illustrates the contributions of the algorithms, prioritizing the conciseness and clarity of the presentation. For another example, with its corresponding trace, please visit https://dominlopez.quarto.pub/fuzzyinclose5/.

Node index = 1 (top): $(\{g1, g2, g3, g4, g5\}, \{\})$. $y = 1$. $\mathcal{P} = \{\}$.
Testing attribute m1

- $l = 0.25$: Canonical, added new concept with index = 2, with extent = $\{g1, g3, g4\}$ and partial intent = $\{m1[0.25]\}$
- $l = 0.5$: Partial test failed: $l^* = 0.75 \neq l = 0.5 \implies \texttt{JumpTo}(k = 3, l = 0.75)$
- $l = 0.75$: Canonical, added new concept with index = 3, with extent = $\{g1[0.25], g3, g4\}$ and partial intent = $\{m1[0.75]\}$
- $l = 1$: Canonical, added new concept with index = 4, with extent = $\{g1[0.25], g3[0.75], g4\}$ and partial intent = $\{m1\}$

Testing attribute m2

- $l = 0.25$: Partial test failed: $l^* = 0.5 \neq l = 0.25 \implies \texttt{JumpTo}(k = 2, l = 0.5)$
- $l = 0.5$: Canonical, added new concept with index = 5, with extent = $\{g2, g3\}$ and partial intent = $\{m2[0.5]\}$
- $l = 0.75$: Canonical, added new concept with index = 6, with extent = $\{g2[0.5], g3\}$ and partial intent = $\{m2[0.75]\}$
- $l = 1$: Canonical, added new concept with index = 7, with extent = $\{g2[0.5], g3[0.75]\}$ and partial intent = $\{m2\}$

Testing attribute m3

- $l = 0.25$: Partial test failed: $l^* = 0.75 \neq l = 0.25 \implies \texttt{JumpTo}(k = 3, l = 0.75)$
- $l = 0.75$: Canonical, added new concept with index = 8, with extent = $\{g1, g2, g5\}$ and partial intent = $\{m3[0.75]\}$
- $l = 1$: Canonical, added new concept with index = 9, with extent = $\{g1[0.75], g2, g5\}$ and partial intent = $\{m3\}$

Node index = 2: $(\{g1, g3, g4\}, \{m1[0.25]\})$. $y = 2$. $\mathcal{P} = \{\}$.
Testing attribute m2

- $l = 0.25$: Partial test failed: $l^* = 0.75 \neq l = 0.25 \implies \texttt{JumpTo}(k = 3, l = 0.75)$
- $l = 0.75$: Non-canonical
- $l = 1$: Non-canonical

Testing attribute m3

- $l = 0.25$: Partial test failed: $l^* = 0.75 \neq l = 0.25 \implies \texttt{JumpTo}(k = 3, l = 0.75)$
- $l = 0.75$: Canonical, added new concept with index = 10, with extent = $\{g1\}$ and partial intent = $\{m1[0.25], m3[0.75]\}$
- $l = 1$: Canonical, added new concept with index = 11, with extent = $\{g1[0.75]\}$ and partial intent = $\{m1[0.25], m3\}$

Node index = 10: $(\{g1\}, \{m1[0.25], m3[0.75]\})$. y out of bounds (leaf node).
Node index = 11: $(\{g1[0.75]\}, \{m1[0.25], m3\})$. y out of bounds (leaf node).
Node index = 3: $(\{g1[0.25], g3, g4\}, \{m1[0.75]\})$. $y = 2$. $\mathcal{P} = \{\}$.
Testing attribute m2

- $l = 0.25$: Partial test failed: $l^* = 0.75 \neq l = 0.25 \implies \texttt{JumpTo}(k = 3, l = 0.75)$
- $l = 0.75$: Canonical, added new concept with index = 12, with extent = $\{g3\}$ and partial intent = $\{m1[0.75], m2[0.75]\}$
- $l = 1$: Non-canonical

Testing att m3

- $l = 0.25$: Partial test failed: $l^* = 1 \neq l = 0.25 \implies \texttt{JumpTo}(k = 4, l = 1)$
- $l = 1$: Non-canonical

Node index = 12: $(\{g3\}, \{m1[0.75], m2[0.75]\})$. $y = 3$. $\mathcal{P} = \{\}$.
Testing attribute m3

- $l = 0.25$: Empty intersection, add to $\mathcal{P}$ (no effect, as no child branches are created).

Node index = 4: $(\{g1[0.25], g3[0.75], g4\}, \{m1\})$. $y = 2$. $\mathcal{P} = \{\}$.
Testing attribute m2

- $l = 0.25$: Partial test failed: $l^* = 1 \neq l = 0.25 \implies \texttt{JumpTo}(k = 4, l = 1)$
- $l = 1$: Canonical, added new concept with index = 13, with extent = $\{g3[0.75]\}$ and partial intent = $\{m1, m2\}$

Testing attribute m3

- $l = 0.25$: Partial test failed: $l^* = 1 \neq l = 0.25 \implies \texttt{JumpTo}(k = 4, l = 1)$
- $l = 1$: Canonical, added new concept with index = 14, with extent = $\{g1[0.25]\}$ and partial intent = $\{m1, m3\}$

Node index = 13: $(\{g3[0.75]\}, \{m1, m2\})$. $y = 3$. $\mathcal{P} = \{\}$.
Testing attribute m3

- $l = 0.25$: Empty intersection, add to $\mathcal{P}$ (no effect).

Node index = 14: $(\{g1[0.25]\}, \{m1, m3\})$. y out of bounds (leaf node).
Node index = 5: $(\{g2, g3\}, \{m2[0.5]\})$. $y = 3$. $\mathcal{P} = \{\}$.
Testing attribute m3

- $l = 0.25$: Partial test failed: $l^* = 1 \neq l = 0.25 \implies \texttt{JumpTo}(k = 4, l = 1)$
- $l = 1$: Canonical, added new concept with index = 15, with extent = $\{g2\}$ and partial intent = $\{m2[0.5], m3\}$

Node index = 15: $(\{g2\}, \{m2[0.5], m3\})$. y out of bounds (leaf node).
Node index = 6: $(\{g2[0.5], g3\}, \{m2[0.75]\})$. $y = 3$. $\mathcal{P} = \{\}$.
Testing attribute m3

- $l = 0.25$: Partial test failed: $l^* = 1 \neq l = 0.25 \implies \texttt{JumpTo}(k = 4, l = 1)$
- $l = 1$: Non-canonical

Node index = 7: $(\{g2[0.5], g3[0.75]\}, \{m2\})$. $y = 3$. $\mathcal{P} = \{\}$.
Testing attribute m3

- $l = 0.25$: Partial test failed: $l^* = 1 \neq l = 0.25 \implies \texttt{JumpTo}(k = 4, l = 1)$
- $l = 1$: Canonical, added new concept with index = 16, with extent = $\{g2[0.5]\}$ and partial intent = $\{m2, m3\}$

Node index = 16: $(\{g2[0.5]\}, \{m2, m3\})$. y out of bounds (leaf node).
Node index = 8: $(\{g1, g2, g5\}, \{m3[0.75]\})$. y out of bounds (leaf node).
Node index = 9: $(\{g1[0.75], g2, g5\}, \{m3\})$. y out of bounds (leaf node).
In the main function: Detected empty extent, added concept with index = 17: $(\{\}, \{m1, m2, m3\})$.

Data availability

Data will be made available on request.

References

[1] Speaker accent recognition, UCI Machine Learning Repository, https://doi.org/10.24432/C52329, 2020.
[2] S. Andrews, In-close, a fast algorithm for computing formal concepts, in: Proceedings of the International Conference on Conceptual Structures (ICCS 2009), Moscow, 2009.
[3] S. Andrews, In-close2, a high performance formal concept miner, in: International Conference on Conceptual Structures (ICCS 2011), in: Lecture Notes in Computer Science, vol. 6828 LNAI, Springer, 2011, pp. 50–62.
[4] S. Andrews, A 'Best-of-Breed' approach for designing a fast algorithm for computing fixpoints of Galois Connections, Inf. Sci. 295 (2015) 633.
[5] R. Bělohávek, Algorithms for fuzzy concept lattices, in: Int. Conf. on Recent Advances in Soft Computing, 2002, pp. 200–205.
[6] R. Bělohávek, Fuzzy Relational Systems: Foundations and Principles, vol. 20, Springer Science & Business Media, 2002.
[7] R. Bělohávek, B. De Baets, J. Outrata, V. Vychodil, Lindig's Algorithm for Concept Lattices over Graded Attributes, LNCS, vol. 4617, 2007, pp. 156–167.
[8] R. Bělohlávek, V. Vychodil, Concept lattices in data analysis, in: International Conference on Formal Concept Analysis, Springer, Berlin, Heidelberg, 2003, pp. 19–36.
[9] R. Bělohlávek, J. Konečny, Fixpoints of fuzzy closure operators via ordinary algorithms, in: 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2017, pp. 1–6.
[10] S.S. Bhowmick, A. Mukherjee, Formal Concept Analysis: Applications in Data Mining, CRC Press, 2012.
[11] A. Burusco-Juandeaburre, R. Fuentes-González, The study of the L-fuzzy concept lattice, Mathw. Soft Comput. 1 (3) (1994) 209–218.
[12] G. Chemmalar Selvi, G.G. Lakshmi Priya, Rating prediction method for item-based collaborative filtering recommender systems using formal concept analysis, EAI Endorsed Trans. Energy Web 8 (33) (2021).
[13] P. Cordero, M. Enciso, D. López, A. Mora, A conversational recommender system for diagnosis using fuzzy rules, Expert Syst. Appl. 154 (2020).
[14] P. Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego, C. Rossi, Knowledge discovery in social networks by using a logic-based treatment of implications, Knowl.-Based Syst. 87 (2015) 16–25.
[15] P. Cordero, M. Enciso, Á. Mora, M. Ojeda-Aciego, C. Rossi, A formal concept analysis approach to cooperative conversational recommendation, Int. J. Comput. Intell. Syst. 13 (1) (2020).
[16] M.E. Cornejo, J. Medina, F.J. Ocaña, Attribute implications in multi-adjoint concept lattices with hedges, Fuzzy Sets Syst. 479 (2024) 108854.
[17] M.E. Cornejo, J. Medina, E. Ramírez-Poussa, C. Rubio-Manzano, Preferences in discrete multi-adjoint formal concept analysis, Inf. Sci. 650 (2023) 119507.

[18] M. Couceiro, A. Napoli, Elements about exploratory, knowledge-based, hybrid, and explainable knowledge discovery, in: Formal Concept Analysis, 2019, pp. 3–16.
[19] D. Dua, C. Graff, UCI Machine Learning Repository, 2017.
[20] G. Fenza, M. Gallo, V. Loia, A. Petrone, C. Stanzione, Concept-drift detection index based on fuzzy formal concept analysis for fake news classifiers, Technol. Forecast. Soc. Change 194 (2023) 122640.
[21] B. Ganter, Two basic algorithms in concept analysis, in: Formal Concept Analysis: 8th International Conference, ICFCA 2010, Agadir, Morocco, March 15-18, 2010. Proceedings 8, Springer, 2010, pp. 312–340.
[22] B. Ganter, S. Obiedkov, Conceptual Exploration, Springer Berlin Heidelberg, 2016.
[23] B. Ganter, R. Wille, Formal Concept Analysis - Mathematical Foundations, Springer, 1999.
[24] F. Hao, J. Gao, Y. Lin, Y. Wu, J. Shang, Concept stability entropy: a novel group cohesion measure in social networks, IEEE Trans. Emerg. Top. Comput. (2023).
[25] M. Kaytoue, S. Kuznetsov, A. Napoli, S. Rudolph, A survey of formal concept analysis support for knowledge discovery in databases, Knowl. Eng. Rev. 29 (2) (2014) 119–146.
[26] Y. Ke, J. Li, S. Li, Bit-Close: a fast incremental concept calculation method, Appl. Intell. (2024) 1–12.
[27] J. Konecny, P. Krajča, Systematic categorization and evaluation of CbO-based algorithms in FCA, Inf. Sci. 575 (2021) 265–288.
[28] P. Krajča, J. Outrata, V. Vychodil, Advances in algorithms based on CbO, in: CLA, vol. 672, Citeseer, 2010, pp. 325–337.
[29] S.O. Kuznetsov, Interpretation on graphs and complexity characteristics of a search for specific patterns, Autom. Doc. Math. Linguist. 24 (1) (1989) 37–45.
[30] S.O. Kuznetsov, A fast algorithm for computing all intersections of objects from an arbitrary semilattice, Naučno-Teh. Inf., Ser. 2 Inf. Processy Sist. 1 (1993) 17–20.
[31] S.O. Kuznetsov, On computing the size of a lattice and related decision problems, Order 18 (2001) 313–321.
[32] R. Lichtinghagen, F. Klawonn, G. Hoffmann, HCV data, UCI Machine Learning Repository, https://doi.org/10.24432/C5D612, 2020.
[33] C. Lindig, Fast concept analysis, in: Working with Conceptual Structures-Contributions to ICCS 2000, 2000, pp. 152–161.
[34] L. Longo, M. Brcic, F. Cabitza, J. Choi, R. Confalonieri, J.D. Ser, R. Guidotti, Y. Hayashi, F. Herrera, A. Holzinger, R. Jiang, H. Khosravi, F. Lecue, G. Malgieri, A. Páez, W. Samek, J. Schneider, T. Speith, S. Stumpf, Explainable artificial intelligence (xai) 2.0: a manifesto of open challenges and interdisciplinary research directions, Inf. Fusion 106 (2024) 102301.
[35] D. López-Rodríguez, A. Mora, J. Domínguez, A. Villalón, I. Johnson, fcaR: Formal Concept Analysis, 2020.
[36] R. Missaoui, S.O. Kuznetsov, S. Obiedkov, Formal Concept Analysis of Social Networks, Springer, 2017.
[37] S.M. Neto, S. Dias, R. Missaoui, L. Zárate, M. Song, Identification of substructures in complex networks using formal concept analysis, Int. J. Web Inf. Syst. 14 (3) (2018) 281–298.
[38] M. Ojeda-Hernández, F. Pérez-Gámez, Á. Mora Bonilla, D. López-Rodríguez, Using logic to determine key items in math education, in: 15th International Conference e-Learning, EL 2021, 2021, pp. 62–69.
[39] J. Outrata, V. Vychodil, Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data, Inf. Sci. 185 (1) (2 2012) 114–127.
[40] S. Pollandt, Fuzzy-kontexte, in: Fuzzy-Begriffe: Formale Begriffsanalyse unscharfer Daten, 1997, pp. 21–49.
[41] U. Priss, A preliminary semiotic-conceptual analysis of a learning management system, Proc. Comput. Sci. 176 (2020).
[42] B. Ramana, N. Venkateswarlu, ILPD (Indian Liver Patient Dataset), UCI Machine Learning Repository, https://doi.org/10.24432/C5D02C, 2012.
[43] H.E. Salman, Feature-based insight for forks in social coding platforms, Inf. Softw. Technol. 140 (2021) 106679.
[44] A. Sangroya, C. Anantaram, M. Rawat, M. Rastogi, Using formal concept analysis to explain black box deep learning classification models, in: FCA4AI@ IJCAI, 2019, pp. 19–26.
[45] P. Silva, A. Maral, Leaf, UCI Machine Learning Repository, https://doi.org/10.24432/C53C78, 2014.
[46] B. Venkatsubramaniam, P.K. Baruah, A novel approach to explainable ai using formal concept lattice, Int. J. Innov. Technol. Explor. Eng. 11 (7) (2022) 1–13.
[47] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts, in: I. Rival (Ed.), Ordered Sets, Springer Netherlands, Dordrecht, 1982, pp. 445–470.
[48] F. Zheng, L. Cui, 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.