Towards a Flexible and Scalable Data Stream Algorithm in FCA

Nicolás Leutwyler[1,2,3,4](✉), Mario Lezoche[1], Diego Torres[2,3], and Hervé Panetto[1]

[1] University of Lorraine, CNRS, CRAN, 54000 Nancy, France
{nicolas.leutwyler,mario.lezoche,herve.panetto}@univ-lorraine.fr
[2] LIFIA, CICPBA-Facultad de Informática, UNLP, La Plata, 1900 Buenos Aires, Argentina
{nicolas.leutwyler,diego.torres}@lifia.info.unlp.edu.ar
[3] Dto. CyT, UNQ, Bernal, 1876 Buenos Aires, Argentina
[4] SNMSF, 38240 Meylan, France

Abstract. The amount of different environments where data can be exploited have increased partly because of the massive adoption of technologies such as microservices and distributed architectures. Accordingly, approaches to treat data are in constant improvement. An example of this is the Formal Concept Analysis framework that has seen an increase in the methods carried out to increment its capabilities in the mentioned environments. However, on top of the exponential nature of the output that the framework produces, the data stream processing environment still poses challenges regarding the flexibility in the usage of FCA and its extensions. Consequently, several approaches have been proposed to deal with them considering different constraints, such as receiving unsorted elements or unknown attributes. In this work, the notion of *flexibly scalable* for FCA distributed algorithms consuming data streams is defined. Additionally, the meaning of different scenarios of lattice merge in a particular data stream model is discussed. Finally, a pseudo-algorithm for merging lattices in the case of disjoint objects is presented. The presented work is a preliminary result and, in the future, it is expected to cover the other aspects of the problem with real data for validation.

Keywords: Formal Concept Analysis · Lattice Merge · Scalability · Incremental Algorithm · Data Stream

1 Introduction

Formal Concept Analysis (FCA), introduced by Wille in [16], is a method for knowledge extraction from a dataset consisting of instances and their attributes. The knowledge it allows to extract is a set of formal concepts, which can be

Partly funded by SNMSF.

understood as natural clusters of instances sharing certain properties. For example, as depicted in Table 1 and Fig. 1, in a dataset with electronic devices, there would probably be a concept with the attribute "screen" in which the instances of *television* and *mobile phone* would belong to. In addition, there is the notion of *sub-concept* by inclusion of attributes, e.g., following the last example, there could be a sub-concept with the attributes "screen" and "battery" that would include *mobile phone* and, unlike its *super-concept*, it would not include the instance of *television*. These formal concepts can give additional information with the form of "having X attributes implies also having Y", e.g., "instances with batteries also have screen with a certain confidence". Notice that the structural output is bounded to the input, meaning that the mentioned result does not imply that all instances with batteries also have a screen in the *real world*.

Table 1. Electronic devices dataset

Objects	Attributes	
	Screen	Battery
Television	X	
Mobile Phone	X	X

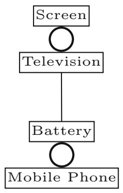

Fig. 1. Electronic devices conceptual hierarchy

Over the last decades, FCA has been used in several areas such as knowledge discovery, information retrieval, machine learning, or automatic software modelling, among others. Additionally, it has been extended in a plethora of ways to deal with the arising problems posed in the mentioned different areas. To name a few, Relational Concept Analysis (RCA) [13], is one extension that allows dealing with multi-relational data-mining (MRDM) [4]. Fuzzy Formal Concept Analysis (FFCA) [8] is the extension that looks to model the uncertainty in data by considering that the instances can have attributes with a degree of certainty in the range of $[0, 1]$, instead of their traditional binary nature, i.e., either having an attribute or not. And Temporal Concept Analysis (TCA) [17], that is the theory of temporal phenomena described with tools of FCA. All these extensions add some degree of flexibility to the FCA method by incrementing the amount of applications it can naturally interact with. However, one of the main pitfalls of FCA is that the amount of formal concepts is exponential in the size of the input in the worst case, making even the best algorithms not directly usable with huge datasets. This poses a problem for scalability because as the input grows bigger, algorithms and computers would require exponentially more resources in order to handle the calculation of the entire set of formal concepts.

Moreover, the need for processing large datasets is becoming more and more common nowadays and a considerable effort has been put into allowing FCA to be applied in such datasets. For organization purposes, we consider three ways of addressing the mentioned problem: reducing the size of the input [1], reducing the size of the output [12,14], and allowing the algorithm to scale in resources and capabilities [2,5,6]. In this work, we focus on the third one, although it is important to understand that the approaches are not mutually exclusive, e.g., there could be a method for reducing both the input and output, or any other combination. Moreover, although there could be other approaches to address the problem, they are out of the scope of this paper.

There are several reasons why reducing the input or the output is sometimes not enough. On the one hand, not all datasets are made of relevant and not relevant data, for example, we could consider a large dataset that has already been reduced to the minimum, i.e., there is only relevant information left. In addition, even though reducing the size of the output could work very well, the information loss could be not acceptable in some situations. Hence, it is important to have ways of scaling without reducing the size of the input or the output. On the other hand, not all environments for knowledge discovery are the same, in some of them it is reasonable to work with static data and also to wait a considerable amount of time until the algorithm finishes. However, there are other scenarios in which the data is dynamic and there is the need for processing it in a short span of time as it arrives. For this reason, we will focus on the online real-time data streams processing environment.

To deal with the aforementioned problem, as mentioned before, many approaches have been proposed and are currently used both in the industry and the academy. Firstly, one of them is the reduction of the input size by considering only a part of, for example, the attributes [1]. Secondly, another way of dealing with the exponential size of the formal concepts is to calculate only a subset of them, as it is in the case of Iceberg Lattices [14] and AOC-posets [12]. Lastly, a huge effort has also been put in developing distributed algorithms in order to take advantage of parallelization when possible [2,5,6]. Nevertheless, to the best of our knowledge, there is still no algorithm, distributed or not, that can be *flexibly* used to extract knowledge from *infinite* data streams (i.e., once the data stream starts, it never stops producing data). Despite some algorithms being able to process data streams, and even if they are incremental, this does not scale well because at some point even adding one row to the formal context would be computationally too costly. Thus, in this work, we present the definition of a problem to solve in order to have a more flexible scalability in the environment of FCA and its extensions considering the *infinite* constraint in data stream processing.

As for the structure of the paper, in Sect. 2, a set of relevant works in the area of distributed algorithms for FCA for data stream processing are presented and contrasted. In Sect. 3, the definitions and the problem are precisely defined. In Sect. 4 the first approach of a solution is proposed. Finally, in Sect. 5 the conclusion, future work, and final discussions are given.

2 Related Works

Many distributed algorithms have been proposed [2,5,6,18], with their main advantage being the ability to compute in parallel and in that way reducing the amount of time it takes to process the output. Most of the distributed algorithms use the MapReduce framework [3] which is appealing to practitioners mainly because they are easily implementable into cloud infrastructures.

In this section, we direct our attention to three main works in the field of distributed algorithms in FCA: one presenting the incremental distributed algorithm based on AddIntent in an iterative fashion, other considering the constraint of not knowing the attributes in advance, and the last one being the only work, to the best of our knowledge, that can run infinitely by forgetting concepts as it processes the data stream. Thanks to an ongoing state-of-the-art study on the characteristics of formal methods for knowledge extraction, and to the best of our knowledge, we detected that these articles represent well the state of the art in data stream processing algorithms for FCA (a subset of the reviewed articles can be found in [7]).

Firstly, Xu et al. [18] contribute with the *iterative distributed* implementations of some known lattice calculation algorithms such as Ganter and Ganter+ which they call MGanter and MGanter+. They present theoretical properties about partitioning the input and working with the partitions instead of the entire formal context. And finally, they use the properties to argue about the correctness of their MapReduce algorithms. The strengths of this approach are that it uses a well accepted framework such as MapReduce, and that it is incremental, which means that it updates the final lattice as the new rows arrive. This also means it is possible to adapt it to process streams.

Secondly, Goel et al. [5] present a MapReduce algorithm that updates the final lattice without assuming prior knowledge of the attributes and thus allows the *arbitrary distribution* of the formal context. They calculate the lattice from a *snapshot* taken at a particular time, and leave the merging step of several *snapshots* out of the scope. Additionally, the algorithm is more suitable for sparse context than for dense ones.

Lastly, De Maio et al. [2] present an incremental distributed algorithm based on AddIntent [10], and suitable for online stream processing. It uses time interval windows in which only one observation is taken into account from each node. Moreover, the proposed algorithm uses Temporal FFCA instead of plain FCA, in order to maintain a reasonably small lattice by taking full advantage of the concepts' support. In the algorithm, they also forget old concepts when there have not been many occurrences of an object in it in a certain time window.

$$\text{Supp}(C) = \frac{|\text{extent}(C)|}{|G|} e^{-\lambda(t - t_{lio})} \tag{1}$$

The mechanic of "forgetting" is calculated based on a *decay factor* $0 \leq \lambda \leq 1$ that decreases the support of a formal concept C based on how much time has passed since the last object has been inserted into it. In the Eq. 1, G is the set of all objects, t is the current timestamp and t_{lio} is the timestamp of the last inserted object into the concept C.

All these distributed algorithms have their advantages and disadvantages, and particularly, the ones presented in [5,18] are not directly prepared for infinite data stream processing, contrary to [2] which, although it is an online data stream processing algorithm, it is based on an extension of FCA [8], and it is not trivial to migrate it into its FCA version.

3 Flexibly Scalable FCA for Data Streams

The goal of this section is to present a discussion about how to provide enough *flexibility* in terms of scalability for stream processing. As presented in the previous section, there are several algorithms that are able to process data streams, and some of them can run infinitely by "forgetting" some information. However, doing so risks losing important concepts. For instance, let us suppose an FCA algorithm that runs in a data stream to study temporal phenomena in smart cities. By forgetting concepts without an object contributing to it in the defined threshold λ, it is possible to lose concepts representing scenarios that rarely occur, but have a huge impact when they do occur. Such a concept could be the one representing a flood in a certain neighborhood or a protest that block certain streets. Therefore, the ideal scenario would be to be able to perform FCA on a data stream infinitely, without compromising *too much* the amount of information lost.

As illustrated in Fig. 2, and defined in [9], a *data stream* is a *countably infinite* sequence of elements.

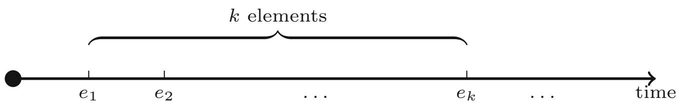

Fig. 2. Data stream over time

Regarding the usage of FCA in data stream processing, the model this work considers is the one in which elements represent observations (instances) occurring at a certain point in time, having certain properties (attributes). This model has been addressed in [2,11], where the goal is to extract relevant knowledge from the relations between events (concepts) in time (temporal paths). In these works, the models were based on FCA and FFCA respectively. We call it Ordered Temporal Model (OTM) because the observations occur ordered in time.

3.1 Merge Lattices

A way of dealing with both the necessity of running infinitely, and the ability to regain a part of the lost information, is to consider an algorithm that keeps the size of the lattice bounded, and another algorithm that can perform the union or merge of the lattices calculated in different points in time. Doing so would give the practitioners the flexibility to choose the size they are able to maintain given their resources, without the worry of losing *all* the information outside the lattice they are maintaining. This, however, comes at the cost of having to store the snapshots in a certain way, which goes outside the scope of this paper.

Definition 1. *Let G and M be the objects and attributes of a formal context respectively. The content of a* ***Formal Context data stream*** *in a given moment $k \in \mathbb{N}$ is defined as an indexed family $S_k = (r_j)_{j \in 1..k}$ where $r_j \subseteq G \times M$.*

Definition 2. *Given a Formal Context data stream S_k, we define the underlying traditional Formal Context $K = (G, M, I)$, where $G = \{\pi_1(r) \mid r \in S_k\}$, $M = \{\pi_2(r) \mid r \in S_k\}$, $gIm \iff \langle g, m\rangle \in S_k$, and both π_1 and π_2 are the functions that return the first and second element of the tuple respectively.*

Definition 3. *A* ***snapshot*** *of a Formal Context data stream between moments $k, l \in \mathbb{N}$ is the sequence of tuples $S_{k,l} = S_l \setminus S_k$.*[1]*. For convenience, we will use the notation $\mathcal{L}_{k,l}$ when speaking about the underlying lattice from $S_{k,l}$.*

In particular, there are three cases we consider important to highlight for merging different lattice snapshots. The first one, depicted in Fig. 3, is the one in which the goal is to merge $\mathcal{L}_{k,l}$ with $\mathcal{L}_{l+1,m}$. The second one, depicted in Fig. 4, is the one where the goal is to merge $\mathcal{L}_{k,l}$ and $\mathcal{L}_{n,m}$ and $l < n$ i.e., there is at least one element outside the covered range. And the last one, depicted in Fig. 5, is the one in which the goal is to merge two snapshots $\mathcal{L}_{k,l}$, and $\mathcal{L}_{n,m}$, where $k \leq l$, $n \leq m$, and $n < l$. The separation in cases is thought with two things in mind (1) Computation: in case there is repeated information between the two snapshots, the computation cost of merging might be reduced in comparison with the case in which both snapshots are completely disjoint. (2) Interpretation: what should be the interpretation of the result when merging two snapshots that are not contiguous?

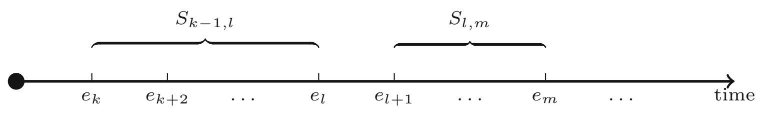

Fig. 3. Contiguous lattice snapshots merge.

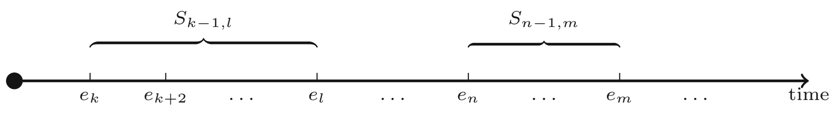

Fig. 4. Spaced lattice snapshots merge case.

[1] Notice that $l \leq k$ implies $S_{k,l} = \varnothing$.

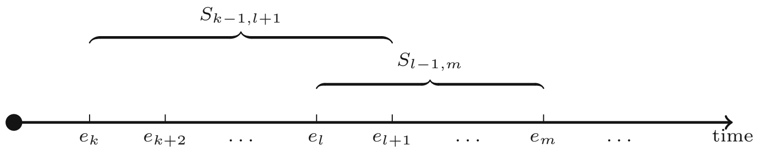

Fig. 5. Intersected snapshots merge case.

Regarding the model, the main problem is that although we could use incremental algorithms to calculate their lattices, at some point the k would be so large that even performing one more update would be too costly, either in time or in space. Thus, for a flexible method whose goal is to run infinitely, it is mandatory to consider "forgetting" objects, attributes, or concepts. Nevertheless, doing so implies potentially losing relevant information, meaning that it is equally necessary to have a way of merging different *snapshots* of the stream, as suggested in [5]. Although one option is to simply join the intervals, remove duplicates, and apply the algorithm to the result, it could be possible that there are not enough resources to run the algorithm with such an input. Moreover, considering that a lattice for each interval has already been calculated, maybe working with them would be more efficient than recalculating everything from the ground up.

Considering this, we say that a method is *flexibly scalable* for knowledge extraction in data stream processing, if it allows running in the data stream infinitely, and it provides a way of considering "forgotten" information without the necessity to *recalculate* or *traverse* the whole lattice.

Merge Problem in the OTM. As defined in [2] and in the previous section, the data stream model merges FCA with Conceptual Time Systems by indexing the timestamped objects adding a time variable to the subject of the formal context, i.e., g_{t_i} with $g \in G$. In this definition, t represents a time window, $i \in \mathbb{N}$, and g_{t_i} is the latest observation g occurring in the time window assigned to that specific time variable. The larger the time window, the more general the study, the smaller, the more detailed it is. Formally (based on [11]):

- g represents a type of observation, e.g., change in temperature of the environment,
- t_i is the i-th element in a discretization of time with the time window t,
- g_{t_i} is the last occurrence of g in the time t_i,
- t_i precedes t_j if $i < j$ for any two $i, j \in \mathbb{N}$.

Definition 4 *(Definition 3.4 in [2]). The intention of an object O at time t_i is defined as*

$$i\{O_{t_i}\} = \{m \mid m \in M \wedge O_{t_i} I m\}$$

which is the set of all attributes of that particular object at time t_i.

Definition 5. *A **Temporal Lattice** $\mathcal{L}^t$ is a pair $(\mathcal{L}, E^t)$ where $\mathcal{L} = (\mathcal{C}, \leq)$ is a concepts lattice and E^t is a set of temporal edges (see Definition 6).*

Definition 6 *(Based on the definition 3.7 in [2]). Given the set of concepts $\mathcal{C}$, a **Temporal Edge** $e^t_{ij} \in E^t$ is a pair $(C_i, C_j) \in \mathcal{C} \times \mathcal{C}$ iff there exist two objects $g_{t_s} \in C_i$ and $g_{t_k} \in C_j$ such that the time t_s precedes the time t_k.*

The temporal edges have a weight associated with the time granule size function $\Delta t : E^t \to \mathbb{R}$ defined by $\Delta t(e^t_{ij}) = 1/|t_{s'} - t_{k'}|$. Furthermore, they define a function that filters edges that are not so representative, called Temporal Edge Support (TES). For the purpose of this work, we would consider the function, but without any particular definition.

Definition 7 *(Definition 3.9 in [2]). A **temporal path** is defined as a path $\pi = (C_s, \ldots, C_t) \in \mathcal{C} \times \mathcal{C} \times \ldots \mathcal{C}$, such that there exist a temporal edge $e^t_{ij} \in E^t$ for $s \leq i \leq j \leq t$.*

In this context, the merge problem is defined by, given two temporal lattice snapshots $\mathcal{L}^t_{k,l} = (\mathcal{L}_1, E^t_1)$, $\mathcal{L}^t_{n,m} = (\mathcal{L}_2, E^t_2)$, return a lattice $\widehat{\mathcal{L}}^t = (\widehat{\mathcal{L}}, \widehat{E}^t)$ such that $\widehat{\mathcal{L}} = \mathcal{L}_1 \mid \mathcal{L}_2$ (see Definition 8), and $\widehat{E}^t$ is the set of temporal edges in the new lattice with a TES support above the given threshold.

Definition 8. *Given two lattices $\mathcal{L}_1$, and $\mathcal{L}_2$ coming from the contexts $K_1 = (G_1, M_1, I_1)$, and $K_2 = (G_2, M_2, I_2)$, their merged lattice $\mathcal{L}_1 \mid \mathcal{L}_2$ is defined as the resulting lattice from the following formal context:*

$$K_1 \mid K_2 = (G_1 \cup G_2, M_1 \cup M_2, I_1 \cup I_2) \tag{2}$$

Notice that in the worst case, the $|G_1 \cup G_2| = (l - k) + (n - m)$, so the merge algorithm that calculates it will still be bounded by its size, that could be up to $2^{\max(|G_1 \cup G_2|, |M_1 \cup M_2|)}$.

4 Merge Lattice Snapshots

In this section, the different cases and particularities in the implementation of lattice snapshot merge in the OTM data stream model are discussed. Moreover, a pseudo-algorithm for the lattice snapshot merge in the disjoint data streams case is presented.

4.1 Interpretation

In the *contiguous* and *intersected* merge cases, the model should have the interpretation: after performing the merge, the resulting lattice represents the same as if the incremental algorithm had run from point k to m. However, that is not the case when there are elements in the middle that are lost between l and n. In OTM, there would be some g_{t_i} where g either has been previously introduced or not. On the one hand, if it was not previously introduced (i.e., $i = 1$) there would be no impact in the temporal paths, besides losing the first part of them (see Fig. 6). On the other hand, if g was previously introduced, it would simplify the path in the erased part by taking much less granular edges (see Fig. 7).

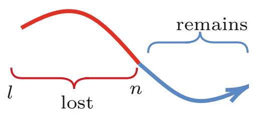

Fig. 6. Temporal path after merging when g_{t_i} occurs for the first time between l and n

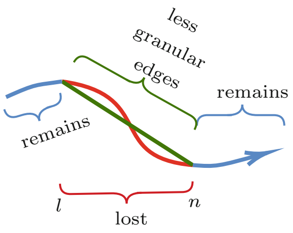

Fig. 7. Temporal path after merging when g_{t_i} occurs before l

4.2 Computation

Computing the merge of lattices needs to be treated differently depending on the model. Particularly, in the OTM, the implementation should consider the repeated objects that could belong to different concepts in the two lattices in the intersected snapshots merge case. Additionally, in the other cases, it is possible that the snapshot had objects with incomplete attributes, i.e., if the snapshot had been taken some time later, some objects in it would have more attributes. The rest of the section from now on focuses on the OTM when objects are disjoint, i.e., spaced, and contiguous snapshots merge cases without incomplete attributes.

Given two concept lattices $\mathcal{L}_1 = (\mathcal{C}_1, \leq)$ and $\mathcal{L}_2 = (\mathcal{C}_2, \leq)$, whose formal context are $K_1 = (G_1, M_1, I_1)$ and $K_2 = (G_2, M_2, I_2)$ respectively. When G_1 and G_2 are disjoint, we claim that each intent would remain *unchanged* after the merge since closures at most would have more objects. Moreover, the only new intents that *could* be added to the merged lattice are: $\varnothing$ and $M_1 \cup M_2$, and the ones resulting from the intersection of two intents in $\mathcal{L}_1$ and $\mathcal{L}_2$.

Lemma 1. *Let $K_1 = (G_1, M_1, I_1), K_2 = (G_2, M_2, I_2), K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = (\mathcal{C}_1, \leq)$, $\mathcal{L}_2 = (\mathcal{C}_2, \leq), \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = (\mathcal{C}_s, \leq)$ be their respective lattices. Then, given a concept $(X, Y) = C_1 \in \mathcal{C}_1$, such that $Y \neq \varnothing$ and $X \neq \varnothing$, there exist a $(Z, Y) \in \mathcal{C}_s$, where $X \subseteq Z$.*

Proof. For reading purposes, we will say $Y = Y_1 = Y_2 = Y_s$. Y_1' will be used when speaking about the derivation in the context of $\mathcal{L}_1$. Similarly, Y_2' will be used when speaking about the derivation in the context of $\mathcal{L}_2$, and Y_s' when speaking about the derivation in the context of $\mathcal{L}_s$. Analogously, we will use the same notation for the set of objects $X = X_1 = X_2 = X_s$.

Since (X, Y) is a formal concept in $\mathcal{C}_1$, $X_1' = Y$ and $Y_1' = X$.

1. If X_1 at least has one element, it would be an element of G_1, i.e., $g \in G_1$. Then $X_s' = Y$, because the merge does not change the attributes held by any of the G_1 objects.

2. $Y'_s = \{g \in G_1 \cup G_2 \mid gI_s m, \forall m \in Y_s\} = \{g \in G_1 \mid gI_s m, \forall m \in Y\} \cup \{g \in G_2 \mid gI_s m, \forall m \in Y\} = Y'_1 \cup Y'_2$. Then $Y''_s = (Y'_1 \cup Y'_2)' = (X \cup Y'_2)' = X' \cap Y''_2$.
3. Since $Y'_s = Y'_1 \cup Y'_2$ (because of step 2), then $Y'_1 \subseteq Y'_s$. Therefore, $X \subseteq Y'_s$.
4. Since $X' = X'_s = Y$ (because of the step 1)., $Y''_s = X' \cap Y''_2 = Y \cap Y''_2 \implies Y''_s \subseteq Y$.
5. Since $Y \neq \varnothing$, let $m \in Y$. Let us suppose that $m \notin Y''_s$, then, there exist an object $x \in Y'_s$ such that $x \not{I}_s m$, which is **absurd** because $Y'_s = \{g \in G_1 \mid gI_s m, \forall m \in Y\} \cup \{g \in G_2 \mid gI_s m, \forall m \in Y\}$, $I_s = I_1 \cup I_2$ and objects are disjoint, thus gI_1m implies $gI_s m$. Analogously, gI_2m implies $gI_s m$. Therefore, $m \in Y''_s$, which means $Y \subseteq Y''_s$.
6. Since $Y''_s \subseteq Y$ and $Y \subseteq Y''_s$, then $Y = Y_s = Y''_s$ (steps 4 and 5). Thus, $(Z, Y) = (Y'_s, Y_s) \in \mathcal{C}_s$, and $X \subseteq Z$ (step 3).

□

Lemma 2. *Let $K_1 = (G_1, M_1, I_1), K_2 = (G_2, M_2, I_2), K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = (\mathcal{C}_1, \leq)$, $\mathcal{L}_2 = (\mathcal{C}_2, \leq), \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = (\mathcal{C}_s, \leq)$ be their respective lattices. Let $(X, Y) \in \mathcal{C}_s, Y \neq \varnothing$, $Y \neq M_1 \cup M_2$ be a formal concept such that its intent Y is not empty nor the whole set of attributes, and it is not an intent of $\mathcal{C}_1$ nor in $\mathcal{C}_2$, then $Y = Z_1 \cap Z_2$ where Z_1 is an intent in $\mathcal{C}_1$ and Z_2 is an intent in $\mathcal{C}_2$.*

Proof. Let $Y'_1 = \{g \in G_1 \mid gI_1m, \forall m \in Y\}$ and $Y'_2 = \{g \in G_2 \mid gI_2m, \forall m \in Y\}$. By definition $Y' = Y'_1 \cup Y'_2$. Since Y is not an intent in $\mathcal{C}_1$ nor in $\mathcal{C}_2$, both Y'_1 and Y'_2 yield a set of objects whose derivatives are larger than Y, i.e., $Y \subset Y''_1$ and $Y \subset Y''_2$. Then $Y = Y''_1 \cap Y''_2$. Since Y is an intent of $\mathcal{C}_s$, $Y = Y''$ and $X = X''$. Moreover, $Y'_1 \subset X$ and $Y'_2 \subset X$ because otherwise, their derivative could not possibly yield more attributes. Let us suppose that Y''_1 is not an intent of $\mathcal{C}_1$, then $Y''_1 \subset Y''''_1$ which is absurd because $B' = B'''$ for any attribute set in the same context (proofed in Sect. 1.1, Proposition 10 of [15]). Similarly, Y''_2 is an intent in $\mathcal{C}_2$. If we rewrite $Y''_1 = Z_1$ and $Y''_2 = Z_2$, we have that $Y = Z_1 \cap Z_2$. □

Considering these properties, a naive algorithm to compute the merged lattice could be the one presented in Algorithm 1. Firstly, it initializes the set of different intents between the two given sets of concepts, and the set of concepts of the new lattice with the top and bottom ones (commonly referred to as $\top$ and $\bot$). Secondly, for each different intent Y, it adds a concept (X, Y) to the $\mathcal{C}_s$ set, where Y is exactly the intent being iterated and X is the union of extents of the respective concepts in each lattice including that intent. For reading purposes, we say that $\mathcal{C}[Y]$ is the concept $(X, Z) = C \in \mathcal{C}$ such that $Y \subseteq Z$ and Z is minimal (i.e., $\nexists (U, V) \in \mathcal{C}$ such that $Y \subseteq V \wedge |V| < |Z|$). If no such concept exists, it returns a tuple $(\varnothing, \varnothing)$, so that $\pi_1(C) = \varnothing$. Lastly, for each pair of formal concepts in both lattices, if their intersection is not $\varnothing$, the formal concept $\{(X_1 \cup X_2, Y_1 \cap Y_2)\}$ is added.

As an example, let us suppose two different lattice snapshots calculated from the formal context defined in Table 1 and Table 2. The merged underlying context is shown in Table 3. For the sake of readability, let us rename Television $= o_1$,

Algorithm 1. Merge $\mathcal{L}_1 = (\mathcal{C}_1, \leq)$ and $\mathcal{L}_2 = (\mathcal{C}_2, \leq)$

```
 1: $All\_Intents \leftarrow \{Y \mid (X,Y) \in \mathcal{C}_1 \cup \mathcal{C}_2 \wedge X \neq \varnothing \wedge Y \neq \varnothing\}$
 2: $\mathcal{C}_s \leftarrow \{\top, \bot\}$
 3: for $Y \in All\_Intents$ do                                          }
 4:     $\mathcal{C}_s \leftarrow \mathcal{C}_s \cup \{(\pi_1(\mathcal{C}_1[Y]) \cup \pi_1(\mathcal{C}_2[Y]), Y)\}$   }  Lemma 1
 5: end for                                                             }
 6: for $(X_1, Y_1) \in \mathcal{C}_1$ do                                      }
 7:     for $(X_2, Y_2) \in \mathcal{C}_2$ do                                  }
 8:         if $Y_1 \cap Y_2 \neq \varnothing$ then                             }
 9:             $\mathcal{C}_s \leftarrow \mathcal{C}_s \cup \{(X_1 \cup X_2, Y_1 \cap Y_2)\}$          }  Lemma 2
10:         end if                                                      }
11:     end for                                                         }
12: end for                                                             }
13: return $(\mathcal{C}_s, \leq)$
```

Mobile Phone $= o_2$, Remote Control $= g_1$ and Notebook $= g_2$ for the objects, and for the attributes Screen $= a_1$, Battery $= a_2$, Camera $= a_3$. Considering this, the concepts of their respective lattices are $\mathcal{C}_1 = \{(\{o_1, o_2\}, \{a_1\}), (\{o_2\}, \{a_1, a_2\})\}$ on the one hand, and on the other $\mathcal{C}_2 = \{(\{g_1, g_2\}, \{a_2\}), (\{g_2\}, \{a_2, a_3\})\}$. Following, if we run the algorithm with the input $\mathcal{L}_1 = (\mathcal{C}_1, \leq), \mathcal{L}_2 = (\mathcal{C}_2, \leq)$, in the line 1, $All\_Intents = \{\{a_1\}, \{a_2\}, \{a_1, a_2\}, \{a_2, a_3\}\}$. Then, in the line 2, $\mathcal{C}_s = \{(\{o_1, o_2, g_1, g_2\}, \varnothing), (\varnothing, \{a_1, a_2, a_3\})\}$. Then, the iterations between line 3 and line 5 go in order:

1. $(Y = \{a_1\})$: $\mathcal{C}_s = \mathcal{C}_s \cup \{(\{o_1, o_2\}, \{a_1\})\}$
2. $(Y = \{a_1, a_2\})$: $\mathcal{C}_s = \mathcal{C}_s \cup \{(\{o_2\}, \{a_1, a_2\})\}$
3. $(Y = \{a_2\})$: $\mathcal{C}_s = \mathcal{C}_s \cup \{(\{o_2\} \cup \{g_1, g_2\}, \{a_2\})\}$
4. $(Y = \{a_2, a_3\})$: $\mathcal{C}_s = \mathcal{C}_s \cup \{(\{g_2\}, \{a_2, a_3\})\}$

Afterwards, between line 6 and line 12, the only combination with a non-empty intersection is $(\{o_2\}, \{a_1, a_2\})$ and $(\{g_2\}, \{a_2, a_3\})$, adding the formal concept $(\{o_2\} \cup \{g_1, g_2\}, \{a_2\})$ which was already added. This shows how the naive algorithm potentially repeats calculations, and that there is room for improvement.

Finally, in line 13, the returned value is

$$\begin{aligned}\mathcal{C}_s = \{&(\{o_1, o_2, g_1, g_2\}, \varnothing), (\varnothing, \{a_1, a_2, a_3\}), (\{o_1, o_2\}, \{a_1\}),\\ &(\{o_2\}, \{a_1, a_2\}), (\{o_2, g_1, g_2\}, \{a_2\}), (\{g_2\}, \{a_2, a_3\})\}\end{aligned}$$

and its line diagram representation is depicted in Fig. 8.

Considering the given algorithm for merging lattices in the case of disjoint objects, it would be possible to use a method that runs maintaining a bounded size, as done in [2], but adapted to the case in which elements are like the ones defined in Definition 1.

Table 2. Extended Formal Context on electronic devices

Objects	Attributes	
	Battery	Camera
Remote Control	X	
Notebook	X	X

Table 3. Merged Formal Context on electronic devices

Objects	Attributes		
	Screen	Battery	Camera
Television	X		
Mobile Phone	X	X	
Remote Control		X	
Notebook		X	X

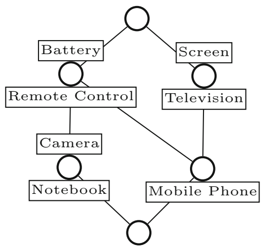

Fig. 8. Merged lattice $\mathcal{L}_s$

5 Conclusion and Future Work

In this paper, the problem of scalability in algorithms for data stream knowledge extraction with FCA has been approached by starting a discussion from several points of view. On the one hand, the notion of *flexibly scalable* for FCA distributed algorithms consuming data streams has been defined in Sect. 3. Furthermore, in the same section, the work formalized the notion of lattice snapshot between times $k \in \mathbb{N}$ and $l \in \mathbb{N}$, and opened a discussion about the meaning of merging two snapshots in three different cases as a way of dealing with the loss of information when keeping a bounded-sized lattice. Additionally, in Sect. 4, the *interpretation* and *computation* of the three different merge cases is discussed considering the model OTM. Finally, a pseudo-algorithm for merging lattices in the case of disjoint objects has been presented, with the addition of the properties that prove its correctness.

The contributions of this paper can be summarized as: (1) The starting point of the discussion about having a general and flexible method for practitioners in order to find a balance between scalability and losing information. (2) A first result in a very specific but real scenario: the intents in the merged lattice when objects are disjoint between the respective formal contexts are composed of all the intents in the two lattice snapshots plus their intersections. (3) The presentation of a naive pseudo-algorithm to compute the lattice in the case of disjoint objects.

For the future work, the study about the non-disjoint objects should be addressed. In addition, it would be interesting to understand whether it is possible to compute the merge using a distributed algorithm to increase the flexibility even further. Particularly, we will work on the study of the rest of the cases to understand their properties and to implement the needed algorithms. Additionally, we plan on working in the study of the merge applied to the multi-relational extensions of the FCA framework (Relational Concept Analysis, Polyadic Concept Analysis, Graph-FCA), to understand whether it could add flexibility there as well or not.

Furthermore, concerning the given properties about the merge with disjoint objects, they should be dual in the sense that they work also with disjoint attributes. However, we did not present the particular proofs for that, so it would be interesting to add them in an extension of the work.

Finally, the study of the time complexity of the algorithm is also needed. For that reason, part of the plan is to study and compare different implementations of Algorithm 1 big-o complexities. Furthermore, we plan on performing benchmarking tests using a real-world case study in the field of e-commerce applied to ski lessons.

Acknowledgements. This work has been funded with the help of the French National Agency for Research and Technology (ANRT) and French National Syndicate of Ski Teachers (SNMSF).

References

1. Co, V., Taramasco, C., Astudillo, H.: Cheating to achieve formal concept analysis over a large formal context. In: CEUR Workshop Proceedings, vol. 959 (2011)
2. De Maio, C., Fenza, G., Loia, V., Orciuoli, F.: Distributed online temporal fuzzy concept analysis for stream processing in smart cities. J. Parallel Distrib. Comput. **110**, 31–41 (2017). https://doi.org/10.1016/j.jpdc.2017.02.002. https://www.sciencedirect.com/science/article/pii/S0743731517300503
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008). https://doi.org/10.1145/1327452.1327492. https://doi.org/10.1145/1327452.1327492
4. Džeroski, S.: Multi-relational data mining: an introduction. ACM SIGKDD Explor. Newslett. **5**(1), 1–16 (2003). https://doi.org/10.1145/959242.959245. https://doi.org/10.1145/959242.959245
5. Goel, V., Chaudhary, B.D.: Concept discovery from un-constrained distributed context. In: Kumar, N., Bhatnagar, V. (eds.) BDA 2015. LNCS, vol. 9498, pp. 151–164. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27057-9_11
6. Krajca, P., Vychodil, V.: Distributed algorithm for computing formal concepts using map-reduce framework. In: Adams, N.M., Robardet, C., Siebes, A., Boulicaut, J.-F. (eds.) IDA 2009. LNCS, vol. 5772, pp. 333–344. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03915-7_29
7. Leutwyler, N., Lezoche, M., Torres, D.: Systematic Literature Mapping - Selected Articles Data Extraction (2022). https://doi.org/10.5281/zenodo.7307957
8. Majidian, A., Martin, T., Cintra, M.: Fuzzy Formal Concept Analysis and Algorithm (2011). pages: 7
9. Margara, A., Rabl, T.: Definition of Data Streams. In: Sakr, S., Zomaya, A.Y. (eds.) Encyclopedia of Big Data Technologies, pp. 648–652. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-77525-8_188

10. van der Merwe, D., Obiedkov, S., Kourie, D.: AddIntent: a new incremental algorithm for constructing concept lattices. In: Eklund, P. (ed.) ICFCA 2004. LNCS (LNAI), vol. 2961, pp. 372–385. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24651-0_31
11. Neouchi, R., Tawfik, A.Y., Frost, R.A.: Towards a temporal extension of formal concept analysis. In: Stroulia, E., Matwin, S. (eds.) AI 2001. LNCS (LNAI), vol. 2056, pp. 335–344. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45153-6_33
12. Osswald, R., Petersen, W.: A logical approach to data-driven classification. In: Günter, A., Kruse, R., Neumann, B. (eds.) KI 2003. LNCS (LNAI), vol. 2821, pp. 267–281. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39451-8_20
13. Rouane-Hacene, M., Huchard, M., Napoli, A., Valtchev, P.: Relational concept analysis: mining concept lattices from multi-relational data. Ann. Math. Artif. Intell. **67** (2013). https://doi.org/10.1007/s10472-012-9329-3
14. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing iceberg concept lattices with Titanic. Data Knowledge Engineering **42**(2), 189–222 (2002). https://doi.org/10.1016/S0169-023X(02)00057-5. https://linkinghub.elsevier.com/retrieve/pii/S0169023X02000575
15. Tamrakar, E.S.: Formal concept analysis: mathematical foundations (1997). https://www.academia.edu/3362029/Formal_concept_analysis_mathematical_foundations
16. Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In: Rival, I. (ed.) Ordered Sets. NATO Advanced Study Institutes Series, pp. 445–470. Springer, Dordrecht (1982). https://doi.org/10.1007/978-94-009-7798-3_15
17. Wolff, K.E.: Temporal Concept Analysis (2001)
18. Xu, B., de Fréin, R., Robson, E., Ó Foghlú, M.: Distributed formal concept analysis algorithms based on an iterative MapReduce framework. In: Domenach, F., Ignatov, D.I., Poelmans, J. (eds.) ICFCA 2012. LNCS (LNAI), vol. 7278, pp. 292–308. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29892-9_26