Incrementally updating Concept Lattices in Arbitrarily Distributed Formal Contexts

Nicolás Leutwyler[1,2,3,4][0000−0003−2115−0430],
Mario Lezoche[1][0000−0002−3271−1742],
Hervé Panetto[1][0000−0002−5537−2261], and
Diego Torres[2,3][0000−0001−7533−0133]

[1] University of Lorraine, CNRS, CRAN, Nancy, F54000, France
firstname.lastname@univ-lorraine.fr
[2] LIFIA, CICPBA-Facultad de Informática, UNLP, La Plata, 1900, Buenos Aires, Argentina firstname.lastname@lifia.info.unlp.edu.ar
[3] Dto. CyT, UNQ, Bernal, 1876, Buenos Aires, Argentina
[4] SNMSF, Meylan 38240, France

Abstract. Decision-making can be fostered by knowledge extraction methods such as that of Formal Concept Analysis (FCA). On top of that, information is not available as a whole at all times in certain contexts, such as when it is distributed, and consulting all of it would be too time-consuming. However, there only exists one algorithm for concept lattice batch computation that does not require full knowledge of the entire set of attributes. But batch algorithms are not best suited for stream processing. For that reason, in this article, we present an incremental algorithm for computing a concept lattice coming from an arbitrarily distributed formal context. And finally, we compare its complexity with that of the existing distributed algorithm.

Keywords: Decision-making systems · Incremental algorithm · Formal Concept Analysis · Algorithm Complexity.

1 Introduction

Introduced by Wille in [16], Formal Concept Analysis (FCA) is a mathematical framework that provides a systematic approach to analyzing complex decision-making systems by modeling relationships between concepts and attributes. At its core, FCA explores the inherent structure of data through the identification of *formal concepts*, which represent sets of objects sharing common properties. These concepts are organized into a lattice structure, often referred to as *concept lattice*, where each concept represents a combination of attributes that define a particular decision criterion or classification rule. By leveraging the principles of lattice theory and order theory, FCA facilitates the exploration of decision spaces, enabling decision-makers to identify relevant patterns, dependencies, and decision rules within their datasets.

FCA has found applications in various domains, including data mining [11, 3], knowledge representation [4, 15], and information retrieval [12], due to its ability to uncover hidden relationships and structures in data. In decision-making systems, FCA serves as a powerful tool for knowledge extraction and decision support, allowing decision-makers to derive actionable insights from complex datasets. By formalizing the relationships between concepts and attributes, FCA enables decision-makers to analyze decision criteria, evaluate alternatives, and identify optimal courses of action. Moreover, FCA provides a principled framework for handling uncertainty and ambiguity in decision-making processes, making it a valuable asset for organizations seeking to improve their decision-making capabilities.

Several algorithms exist to compute the concept lattice of a formal context (i.e., objects and their attributes). These algorithms are usually categorized into two categories: *batch* and *incremental*. The first one comprises algorithms that have full knowledge of the whole input (i.e., the formal context), and compute the concept lattice with it [1, 10]. The latter includes algorithms that do not know the whole set of objects in advance, and update the current lattice as they arrive [9]. Moreover, contrary to batch algorithms, incremental algorithms have the advantage that they can be used in stream processing applications, because they do not need to recompute everything from scratch every time a new object is added to the context. However, incremental algorithms consider that the set of all attributes is known from the beginning, which might not be the case in all scenarios.

Goel and Chaudary in [6] present an algorithm to compute a concept lattice in a distributed environment, where the formal context is arbitrarily distributed. In other words, since there are no constraints in how the formal context (i.e., object, attributes, and the incidence relationship) is distributed across nodes, not only all objects are unknown in advance, but neither the attributes are. Additionally, the algorithm leverages the advantages of the Apache Spark[1] batch streams, making it suitable to analyze *static* snapshots of the stream. Therefore, the algorithm they present falls into the batch category, even if the batch comes from a data stream. Nevertheless, an algorithm that takes advantage of the concept lattice already being computed would be necessary to aid the flexibility and usability of FCA.

Since to the best of our knowledge, no algorithm exists to *incrementally* compute a concept lattice from *arbitrarily distributed formal contexts*, in this article an algorithm to do so is presented. Moreover, its computational complexity bound is studied. In addition, a discussion on different approaches to address the problematic is provided.

This work is structured as follows, section 2 presents necessary notation and concepts to understand the contents of the article. Moreover, it presents the algorithm of Goel and Chaudhary as the only related know to the best of our knowledge. Then, section 3 presents the AddPair incremental algorithm for lattice construction coming from an arbitrarily distributed formal context. Further-

[1] https://spark.apache.org/

more, section 4 carries on with a discussion on the big $\mathcal{O}$ complexity bound of both algorithms. And finally, in section 5, the conclusions and future work are presented.

2 Preliminaries and Related Works

In this section, preliminary notation will be given so that the rest of the article is easier to follow. Furthermore, the related work [6] is presented in detail.

2.1 Formal Concept Analysis

Firstly, let us define the core formalities of FCA,

Definition 1. *A **formal context** $\mathcal{K}$ is a triple $\langle G, M, I\rangle$, where G is a set of objects, M is a set of attributes, and I is an incidence matrix where iIj if $g_i \in G$ has the attribute $m_j \in M$, and $i\not{I}j$ otherwise.*

Let $'$ be the derivation operation on a set of objects $X \subseteq G$ (dually, on a set of attributes $Y \subseteq M$) given by

$$X' = \{m \in M \mid \forall g \in X,\ gIm\}$$
$$Y' = \{g \in G \mid \forall m \in Y,\ gIm\}$$

A *formal concept* is a pair $C = \langle X, Y\rangle$ where $X \subseteq G, Y \subseteq M$, $X' = Y$, and $Y' = X$. X is called the extent and Y the intent. In other words, if we add any attribute to Y, then $(Y \cup \{m\})' \subset X$ for $m \in M \ \wedge \ m \notin Y$, and analogously, if we add any object to X, $X \cup \{g\} \subset Y$ for $g \in G \wedge g \notin X$. With these definitions, the set of all formal concepts, typically noted with the letter $\mathcal{C}$, and the relation of inclusion of extents ($\leq$) form the so-called *concept lattice*, which is a partially ordered set, and is usually noted with the letter $\mathcal{L} = \langle \mathcal{C}, \leq\rangle$.

2.2 Goel and Chaudhary's Algorithm

Multiple algorithms exist to mine formal concepts from a given data-set on standalone and single threaded systems. However, these algorithms are restricted in applicability as the size of the context increases. One of the reasons why this happens is that lattice construction algorithms are inherently exponential in the worst case: the amount of formal concepts is bound by $2^{O(|G|+|M|)}$. Consequently, interest in finding distributed solutions that would reduce the time for mining as well as enable distributed storage using low-cost networked computing nodes has been spurred by these limitations.

For instance, it has been successfully used in a distributed environment in [2], but in conjunction with the extensions called Fuzzy Formal Concept Analysis (i.e., an extension of FCA [8] where the incidence matrix is a relation $I \subseteq G \times M \rightarrow [0, 1]$) and Temporal Concept Analysis ([17]). This work takes advantage of the fuzzy and temporal characteristics of the extensions in order

to deal with the exponential growth of the fuzzy lattice. In addition, numerous works have been conducted to address the problem of scalability by leveraging the MapReduce paradigm. However, these algorithms encounter constraints that hinder their scalability the larger the formal context size gets. One of these constraints involves the necessity to store the entire context on every node, while the other is weighed down by significant communication overhead at the conclusion of each iteration. In practical scenarios, we encounter extensive datasets that exceed the storage capacity of a single machine. For instance, consider a dataset comprising emails sent by customers to a company, which may be utilized to identify correlations among customers. Another illustration could be an e-commerce platform that provides a diverse array of products to its customers, aiming to categorize customers based on their specific product browsing behaviors.

In the light of this context, in [6], the authors present an algorithm for computing concept lattices from an unconstrained (i.e., arbitrarily distributed) formal context. The algorithm utilizes the Apache Spark MapReduce framework and leverages its core abstraction called Resilient Distributed Dataset (RDD). In their approach, the formal context is only used at the beginning of the algorithm to generate a distributed collection of key-value pairs (object-attribute if $|G| < |M|$, or attribute-object otherwise). Then, they generate a new key-value pair from an existing one representing the formal concepts of size k, with $k \in \mathbb{N}$. The process is repeated until $k = |G|$ and, as a result, all formal contexts have been generated.

Going more in depth, we will first set a common ground on what MapReduce is. Generally, we can think of it as a computational model where two operations can be performed: *map* and *reduce*. Map refers to a function applied to every element of the input, and reduce is the combination of the results performed by the map step. Typically, the map is the operation that is expected to leverage concurrency, while reduce is the one that "combines" the results computed in several nodes. In Figure 1, a basic example is given, where a driver node first performs a map($*2$) which makes each node to multiply every value (assuming they are all numbers) by two. Then, it performs a reduce($+$) which combine the results of the map, returning the sum of the results performed in each of the nodes.

More precisely, the algorithm proposed in [6] can be summarised as follows,

Input: A key-value pair P of object-attribute or attribute-object, i.e., $\langle g, m \rangle$ such that $g \in |G| \ \wedge \ m \in |M|$ if $|G| < |M|$, $\langle g, m \rangle$ otherwise.

Output: An RDD $\mathcal{C}$ with pairs $\langle k, v \rangle$ where k is an extent (or intent if $|M| < |G|$), v an intent (or extent), and $\langle k, v \rangle \in \mathcal{C}$ iff $\langle k, v \rangle$ is a formal concept from the underlying formal context $\mathcal{K} = \langle \{\pi_1(p) \mid \forall\ p \in \mathrm{P}\}, \{\pi_2(p) \mid \forall\ p \in \mathrm{P}\}, \mathrm{P} \rangle$.

Procedure: Since mining concepts on a formal context or its transpose yields the same result, the authors chose to aggregate data by objects or by attributes,

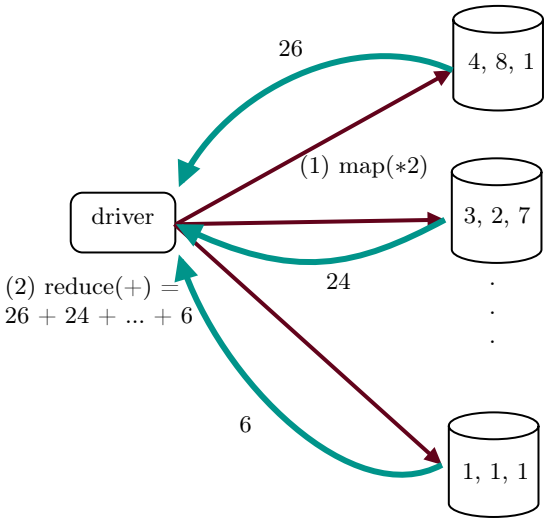

Fig. 1: Basic Map Reduce example.

depending on which one has the lower count. To make this decision, they utilize the countApproximate function present in Apache Spark, that does not need to actually count *all* the elements on each dimension. Once the dimension is chosen, and assuming for simplicity that the objects are more than the attributes, the idea of the algorithm relies on generating candidate formal concepts by key size (i.e., attributes in this case).

The generation of candidate formal concepts of size 1 (referred as CC_1) is performed by aggregating pairs on the key as an *id* in the following way,

$$CC_1 = \{\langle g, M\rangle \mid g \in G \ \wedge \ M = \{m \mid \langle g, m\rangle \in \mathrm{P}\}\} \tag{1}$$

Then, an iterative process is defined based on the candidate set of formal concepts CC_N consisting of key-value pairs (as defined in Equation 1) $\langle K_N, V\rangle$ of keys with size N. First, let us define K_{N+1} by considering K_N to be lexicographically ordered, i.e., $k_i \leq k_{i+1} \forall i \in [1, l)$ for $K_N = \{k_1, k_2, \ldots, k_l\}$. Given two keys of size N with their first N elements in common, $P_N = \{e_1, e_2, \ldots, e_{n-1}, a\}$ and $Q_N = \{e_1, e_2, \ldots, e_{n-1}, b\}$, a new key of size $N+1$ would be generated as follows:

$$K_{N+1} = e_1, e_2, \ldots, e_{n-1}, a, b \tag{2}$$

Then, C_{N+1} is defined by combining all possible combinations of lexicographically ordered keys in CC_N ($\frac{l(l-1)}{2}$ combinations where $|CC_N| = l$) by using the formula Equation 2. And finally, given $P_N = \{e_1, e_2, \ldots, e_{n-1}, a\}$ and $Q_N = \{e_1, e_2, \ldots, e_{n-1}, b\}$ two keys of size N, and V_1, V_2 their values, then, the set of pairs of the form $\langle e_1, e_2, \ldots, e_{n-1}, a, b, V_1 \cap V_2\rangle$ defines CC_{N+1}.

Now, the set of valid formal concepts of size N are defined using CC_N and CC_{N+1}. Let $V_X = \{\pi_2(p) \mid p \in CC_{N+1}\}$ be the set of values of CC_{N+1}. Then, the set of valid formal concepts of size N is

$$VC_N = \{\langle K_N, V\rangle \mid \langle K_N, V\rangle \in CC_N \text{ and } V \notin V_X\} \tag{3}$$

The final result is given by all the valid formal concepts VC_i with $i \in \mathbb{N}$ and $1 \leq i < |M|$ (or $|G|$ if there were more attributes than objects). The bottom and top concepts are computed separately.

2.3 Additional relevant methods

In the literature, at least three other related methods have been found that seek to make the FCA computation more flexible, considering constraints such as not having the entire input in memory. Firstly, Valtchev and Missaoui in [14] generalized the theory regarding incremental algorithms, by proposing an algorithm that can add several objects in one incremental step. To do so, they used the notion of *subposition*, i.e., the assembly of contexts sharing the same attributes, or in other words, adding objects with certain non-new attributes to a context. Secondly, in [13] the idea of assembling *partial* lattices to construct a global one is introduced using both subposition and apposition (i.e., the horizontal concatenation of contexts sharing the same set of objects [5]). This is done with the goal of allowing FCA algorithms to compute lattices beyond what fits in memory. Finally, in [7], the problem of knowledge loss while maintaining a bounded-sized lattice in data stream processing is introduced, and the merging (assembling) of lattices is proposed as a solution. However, although they certainly contribute to the application of FCA to data streams, they are still not enough to consider *arbitrarily* distributed formal contexts incrementally.

3 AddPair algorithm

In the last section, we presented an algorithm that performs a distributed batch algorithm to compute the set of formal concepts from an arbitrarily distributed formal context. In this section, we present an *incremental* algorithm to compute a set of formal concepts and its line diagram.

In order to incrementally add pairs $\langle g, m\rangle$ to a lattice $\mathcal{L}$, we need to first clarify certain aspects. First, all incremental algorithms to the best of our knowledge consider that either M or G is known in advance, and then the incremental steps happen on the elements that are unknown. Second, an incremental update is usually considered between $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ where the difference between the two lattices is that g is being added to $\mathcal{L}_i$ with all attributes g', and additionally $g \notin G_i$. Third, the techniques these algorithms use in order to efficiently traverse the lattice and avoid doing unnecessary computations usually rely on having the line diagram ($\preceq$ lower and upper neighbors in [5]) updated after each incremental step. Moreover, in this type of algorithm, given an update from the lattice $\mathcal{L}_i$ to $\mathcal{L}_{i+1}$, they typically separate the concepts into the categories

AI.1 *New*: a concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$ where D is not an intent in $\mathcal{L}_i$.
AI.2 *Modified*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that $B \subseteq g'$ since g has to be added to its extent in $\mathcal{L}_{i+1}$.
AI.3 *Generator*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that given a *new* concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$, $D = B \cap g' \neq B$. And
AI.4 *old*: any other concept.

The primary challenge of incremental construction lies in the identification of all modified concepts and the determination of all canonical generators for new concepts (to ensure each new concept is generated precisely once). Efficient algorithms aim to minimize the effort spent on searching through unmodified and non-canonical generators. Some algorithms such as AddIntent, defined in [9], address this challenge by recursively traversing the diagram graph of $\mathcal{L}$.

For the purpose of this conception of 'incrementally adding pairs", we will define new versions of the concepts categories,

AP.1 *New*: a concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$ where D is not an intent in $\mathcal{L}_i$ and C is not an extent in $\mathcal{L}_i$.
AP.2 *Modified*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that $B \subseteq g'$ since g has to be added to its extent in $\mathcal{L}_{i+1}$ or $A \subseteq m'$ since m has to be added to its intent in $\mathcal{L}_{i+1}$.
AP.3 *Generator*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that given a *new* concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$, $D = B \cap g' \neq B$ or $C = A \cap m' \neq A$ (these second ones will be referred to as generator concept^{-1}). And
AP.4 *old*: the rest.

So, let us firstly identify what are the different scenarios when adding $\langle g, m\rangle$ to $\mathcal{L}_i$ where g might or might not be in G_i, m might or might not be in M_i, and $\langle g, m\rangle \notin I_i$.

First, let us consider the case where $g \notin G_i$, and $m \notin M_i$. In this case, the only concepts $\langle A, B\rangle \in \mathcal{L}_i$ that could be *modified* are the bottom and the top. In fact, if $\bot_i = \langle \varnothing, M_i\rangle$, then, it is a modified concept and its new version would be $\bot_{i+1} = \langle \varnothing, M_i \cup \{m\}\rangle$. Analogously, if $\top_i = \langle G_i, \varnothing\rangle$, its modified version would be $\top_{i+1} = \langle G_i \cup \{g\}, \varnothing\rangle$. However, if $\bot_i = \langle X, M_i\rangle$, $X \neq \varnothing$, then it is the canonical generator of the *new* concept $\bot_{i+1}$. Similarly, if the intent of $\top_i$ is not empty, then it will be the canonical generator^{-1} of the *new* concept $\top_{i+1} = \langle G_i \cup \{g\}, \varnothing\rangle$.

Second, if $g \notin G_i$, but $m \in M_i$ the cases are, in fact, exactly the same as in AddIntent (AI.1-AI.4). Analogously, if $m \notin M_i$, but $g \in G_i$, the dual version of the AddIntent algorithm (e.g., we can call it "AddExtent") would suffice, hence, the cases are also dual (see the additions of AP.1-AP.4 with respect to AI.1-AI.4).

Finally, let us consider the case when $g \in G_i$ and $m \in M_i$. As we can see in Figure 2, in $\mathcal{L}_i$, assuming that b is the generator concept^{-1} of g, whereas d is the generator concept of m, we can see in **red** all concepts containing g in their extents, and in **light-blue**, all concepts containing m. Moreover, concepts *below* b might or might not contain m in their intent, whilst concepts *above* d might or might not contain g in their extent. For instance, in this particular scenario, w contains m in its intent because it is a sub-concept of d. What we know, from

a procedural point of view, is that the red concepts *may be* modified because their intent now might include m, and light-blue concepts *may be* modified analogously because their extent might have to include g now.

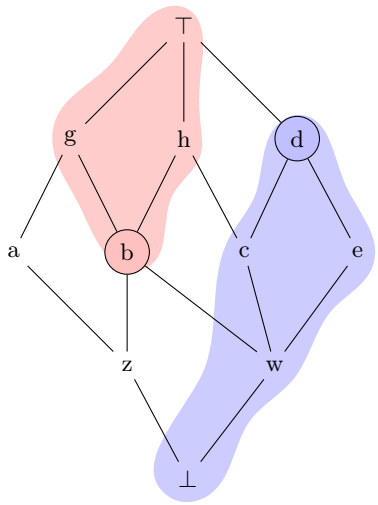

Fig. 2: Concept categories during the an AddPair incremental step in a concept lattice.

Considering that finding one generator can be done in $\mathcal{O}(|G|^2|M|)$[2], while, dually, finding a generator^{-1}, in $\mathcal{O}(|G||M|^2)$, we could argue that an algorithm to find the generator of m and the generator^{-1} of g would cost $\mathcal{O}(|G|^2|M| + |G||M|^2)$, or its simplified version $\mathcal{O}(\max(|G|^2|M|, |G||M|^2))$. Additionally, in the worst case, either above the generator^{-1} of g or below the generator of m, there could be $|\mathcal{L}_i|$ concepts.

In the light of that, we propose Algorithm 1 to incrementally add a pair $\langle g, m\rangle$ to a lattice $\mathcal{L}_i$ such that both g and m may or may not be present in G_i and M_i respectively. The algorithm consists in the combination of two other incremental algorithms: AddIntent [9] and DeleteInstance [18]. Basically, if $g \in \mathcal{L}_i$, between lines 1-5 the algorithm sets a variable old_intent with the value of g' (i.e., in $\mathcal{L}_i$, thus it does not include m yet), then, it deletes the object g (i.e., equivalent to deleting the row in which that instance appears in the formal context), and finally, it sets a variable new_intent as $g' \cup \{m\}$. Otherwise, $new\_intent = \{m\}$, since g is new in $\mathcal{L}_{i+1}$ and its only related attribute is m (line 6). Then, at line 7, $new\_intent = g'$ regardless of whether $g \in G_i$ or not. Continuing, in line 8, in

[2] This bound depends on several aspects, notably, the worst case happens when the algorithm start traversing the lattice from the $\bot$ (or dually from the $\top$). However, this can be considerably faster if we start from higher in the hierarchy (or lower in the case of generator^{-1})

case $m \notin M_i$, the algorithm updates the bottom using Algorithm 2, which simply creates a new bottom containing all attributes (as it should be) when there exists at least an object in $\perp$, or it updates the existing one otherwise. Then, we are in the case in which $m \in M$, and $g \notin G$, hence, we can use AddIntent given that we also have g' in the *new_intent* variable (line 11). Finally, the resulting $\mathcal{L}$ is returned which corresponds to $\mathcal{L}_{i+1}$.

Algorithm 1 Algorithm to incrementally add the attribute m to the object g in the lattice $\mathcal{L}$

```
1: if g ∈ L.G then
2:     old_intent ← g'
3:     delete_instance(g, L)
4:     new_intent ← old_intent ∪ {m}
5: else
6:     new_intent ← {m}
7: end if
8: if m ∉ L.M then
9:     update_bottom(m, L)
10: end if
11: add_intent(g, m, L)
12: return L
```

Algorithm 2 Algorithm to update the $\perp$ concept in $\mathcal{L}$ considering that $m \notin M$

```
1: L.M ← L.M ∪ {m}
2: if L.⊥.E = ∅ then
3:     L.⊥.I ← L.M
4: else
5:     new_bottom ← ⟨L.⊥.E, L.M⟩
6:     set_link(L.⊥, new_bottom)
7:     L.⊥ ← new_bottom
8: end if
9: return L
```

4 Discussion

As stated before, one of the challenges of incremental construction algorithms is efficiently identifying the *modified* concepts. Furthermore, for each of the modified concepts, it would be necessary to find its generator (or dually, its generator^{-1}), because by adding g to its extent (or m to its intent), it is possible that a concept with that extent (or intent) already exists in $\mathcal{L}_i$. Since the amount of modified concepts in the worst case is $|\mathcal{L}_i|$, and finding the generator^{-1} of g

and the generator of m would cost $\mathcal{O}(\max(|G|^2|M|, |G||M|^2))$, the best we could hope for is bounded[3] by,

$$\begin{aligned}\mathcal{O}(|\mathcal{L}|\ \max(|G|^2|M|, |G||M|^2)) &= \mathcal{O}(|\mathcal{L}|\ (|G|^2|M| + |G||M|^2)) \\ &= \mathcal{O}(|\mathcal{L}||G||M|\ (|G| + |M|)) \\ &= \mathcal{O}(|\mathcal{L}||G||M|\ \max(|G|, |M|))\end{aligned} \tag{4}$$

Then, if we analyze Algorithm 1, the complexity between the lines 1-7 is the complexity of DeleteInstance, i.e., $\mathcal{O}(|\mathcal{L}||G||M|^2)$ [18]. Then, the complexity between the lines 8-12 is the maximum between Algorithm 2 and AddIntent. Since the complexity of Algorithm 2 is negligible compared to that of AddIntent, the complexity between these lines ends up being $\mathcal{O}(|\mathcal{L}||G|^2|M|)$[4] ([9]). Further, the total complexity of Algorithm 1 is

$$\begin{aligned}\mathcal{O}((|\mathcal{L}||G|^2|M| + |\mathcal{L}||G||M|^2)) &= \mathcal{O}(|\mathcal{L}||G||M|\ (|G| + |M|)) \\ &= \mathcal{O}(|\mathcal{L}||G||M|\ \max(|G|, |M|))\end{aligned} \tag{5}$$

Then, it is noticeable that Equation 5 is exactly the same bound as Equation 4, which in turn is the best we can hope for when updating every modified object exactly once.

Finally, let us give an asymptotic bound for the Goel and Chaudhary's Algorithm. Recalling Equation 1, which in their implementation is performed with a *reduceByKey* function on the distributed pairs. This first step does not play a considerable role in the complexity of the algorithm, and for that reason, we will leave it aside. The most crucial part of this algorithm for the asymptotic behavior is the calculation of C_{N+1} from C_N, since it generates $\mathcal{O}(|C_N|^2)$ (or more precisely $\frac{|C_N|(|C_N|-1)}{2}$, which is quadratic nonetheless) new candidate concepts. This is repeated for every C_N with $1 \leq N < \max(|g'|)$ (or $\max(|m'|)$). Since in the worst case $\max(|g'|) = |M|$, and considering that $|C_1| \leq |M|$ (or $|G|$ when there are more attributes than objects), a big O bound for these steps would be

$$\mathcal{O}\left(\sum_{i=1}^{|M|-1} |M|^{2^i}\right) \tag{6}$$

Although $|\mathcal{L}| = \mathcal{O}(2^{\ max(|G|,|M|)})$ in the worst case, in practical scenarios, when there is highly correlated data, the amount of concepts could be much smaller than its exponential bound. However, Equation 6 reveals that the complexity of Goel and Chaudhary's Algorithm, although scalable and parallelizable, suffers from the *exponential* generation of candidate concepts, *regardless* of the

[3] Notice that $\mathcal{O}$ is an *upper* bound.

[4] With certain optimizations, a tighter bound is $\mathcal{O}(|\mathcal{L}||G|^2\max(|g'|))$ where $\max(|g'|)$ is the maximum amount of attributes related to an object. However, to simplify the math, we will use the simpler version of the two.

sparsity of the formal context, e.g., if only one object in the formal context has its $g' \approx |M|$, then the algorithm would necessarily compute Equation 6 candidate concepts up to $|M|$.

5 Conclusions and Future Work

In this article, a novel algorithm for incrementally updating a lattice $\mathcal{L}$ with a complexity of $\mathcal{O}(|\mathcal{L}||G||M| \max(|G|, |M|))$ has been presented. Furthermore, the results of the theoretical analysis suggest that Algorithm 1 should be a better fit than the algorithm of Goel and Chaudhary complexitywise, when the formal context is *sparse* (which is often the case in practice). However, both algorithms have several differences that make the comparison very hard to perform. Firstly, being *batch* vs *incremental*, and secondly, being parallel vs single threaded. The existing batch algorithm have the upper hand in scalability regarding the fact that it is an inherently distributed algorithm, or in other words, if needed, more nodes can be added to the used cluster in order to compute the lattice faster. Furthermore, when it comes to flexibility, Algorithm 1 is preferable because it does not require to restart the computation when new objects or attributes are added. Yet, ideally, a fully distributed solution to the incremental computation of arbitrarily distributed formal context must be explored.

Acknowledgments. 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).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Andrews, S.: In-Close, a Fast Algorithm for Computing Formal Concepts. p. 15. Moscow (2009)

[2] De Maio, C., Fenza, G., Loia, V., Orciuoli, F.: Distributed online Temporal Fuzzy Concept Analysis for stream processing in smart cities. Journal of Parallel and Distributed Computing **110**, 31–41 (Dec 2017). https://doi.org/10.1016/j.jpdc.2017.02.002, https://www.sciencedirect.com/science/article/pii/S0743731517300503

[3] Dolques, X., Le Ber, F., Huchard, M., Grac, C.: Performance-friendly rule extraction in large water data-sets with AOC posets and relational concept analysis. International Journal of General Systems **45**(2), 187–210 (Feb 2016). https://doi.org/10.1080/03081079.2015.1072927, https://doi.org/10.1080/03081079.2015.1072927, publisher: Taylor & Francis _eprint: https://doi.org/10.1080/03081079.2015.1072927

[4] Fu, G.: FCA based ontology development for data integration. Information Processing & Management **52**(5), 765–782 (Sep 2016). https://doi.org/10.1016/j.ipm.2016.02.003, https://www.sciencedirect.com/science/article/pii/S030645731630019X

[5] Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/978-3-642-59830-2, http://link.springer.com/10.1007/978-3-642-59830-2

[6] Goel, V., Chaudhary, B.D.: Concept Discovery from Un-Constrained Distributed Context. In: Proceedings of the 4th International Conference on Big Data Analytics - Volume 9498. pp. 151–164. BDA 2015, Springer-Verlag, Berlin, Heidelberg (Dec 2015). https://doi.org/10.1007/978-3-319-27057-9_11, https://doi.org/10.1007/978-3-319-27057-9_11

[7] Leutwyler, N., Lezoche, M., Torres, D., Panetto, H.: Towards a Flexible and Scalable Data Stream Algorithm in FCA. In: Ojeda-Aciego, M., Sauerwald, K., Jäschke, R. (eds.) Graph-Based Representation and Reasoning. pp. 104–117. Lecture Notes in Computer Science, Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-40960-8_9

[8] Majidian, A., Martin, T., Cintra, M.: Fuzzy Formal Concept Analysis and Algorithm (Jan 2011), pages: 7

[9] van der Merwe, D., Obiedkov, S., Kourie, D.: AddIntent: A New Incremental Algorithm for Constructing Concept Lattices. In: Eklund, P. (ed.) Concept Lattices. pp. 372–385. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24651-0_31

[10] Nourine, L., Raynaud, O.: A fast algorithm for building lattices. Information Processing Letters **71**(5-6), 199 – 204 (Sep 1999). https://doi.org/10.1016/S0020-0190(99)00108-8, https://hal.archives-ouvertes.fr/hal-01765512, publisher: Elsevier

[11] Seid, D., Mehrotra, S.: Efficient relationship pattern mining using multi-relational iceberg-cubes. In: Fourth IEEE International Conference on Data Mining (ICDM'04). pp. 515–518 (Nov 2004). https://doi.org/10.1109/ICDM.2004.10059

[12] V S, A., S., A.: Extracting Conceptual Relationships and Inducing Concept Lattices from Unstructured Text. Journal of Intelligent Systems **28** (Jan 2017). https://doi.org/10.1515/jisys-2017-0225

[13] Valtchev, P., Missaoui, R., Lebrun, P.: A partition-based approach towards constructing Galois (concept) lattices. Discrete Mathematics **256**(3), 801–829 (Oct 2002). https://doi.org/10.1016/S0012-365X(02)00349-7, https://www.sciencedirect.com/science/article/pii/S0012365X02003497

[14] Valtchev, P., Missaoui, R.: Building Concept (Galois) Lattices from Parts: Generalizing the Incremental Methods. In: Delugach, H.S., Stumme, G. (eds.) Conceptual Structures: Broadening the Base. pp. 290–303. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-44583-8_21

[15] Wang, P., Lu, W., Meng, Z., Wei, J., Fogelman-Soulié, F.: Towards Knowledge Structuring of Sensor Data Based on FCA and Ontology p. 1

[16] Wille, R.: Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In: Rival, I. (ed.) Ordered Sets. pp. 445–470. NATO Advanced Study Institutes Series, Springer Netherlands, Dordrecht (1982). https://doi.org/10.1007/978-94-009-7798-3_15

[17] Wolff, K.E.: Temporal Concept Analysis. In: ICCS-2001 International Workshop on Concept Lattices-Based Theory, Methods and Tools for Knowledge Discovery in Databases, Stanford University, Palo Alto (CA). pp. 91–107 (2001)

[18] Zou, L., Zhang, Z., Long, J., Zhang, H.: A fast incremental algorithm for deleting objects from a concept lattice. Knowledge-Based Systems **89**, 411–419 (Nov 2015). https://doi.org/10.1016/j.knosys.2015.07.022, https://www.sciencedirect.com/science/article/pii/S0950705115002762