A Proposal for Building a Compact and Tunable Representation of a Concept Lattice Based on Clustering

Jaume Baixeries[1], Alexandre Bazin[2], Jérôme David[3], and Amedeo Napoli[4]

[1] Universitat Politècnica de Catalunya, Computer Science Dpt. 08034 Barcelona, Catalonia jbaixer@cs.upc.edu
[2] Université de Montpellier, CNRS, LIRMM, F-34095 Montpellier, France alexandre.bazin@lirmm.fr
[3] Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France jerome.david@inria.fr
[4] Université de Lorraine, CNRS, Loria, F-54000 Nancy, France amedeo.napoli@loria.fr

Abstract. A concept lattice provides a model of a dataset that can be navigated and explored by an analyst in an interactive way, except when the concept lattice is too large. Such a problem can be overcome by building a representation of the whole concept lattice that keeps a reasonable size and that can be interpreted by the analyst. Relying on previous work about link key discovery, we revisit in this paper an approach based on Formal Concept Analysis (FCA) and Agglomerative Hierarchical Clustering (AHC) applied to a set of concepts for building a representative set of clusters. Accordingly, we propose an AHC algorithm that (a) efficiently computes this representative set, and (b) respects the ordinal structure of the original concept lattice. A set of experiments performed over real datasets shows the effectiveness of our approach.

Keywords: Formal Concept Analysis · Agglomerative Hierarchical Clustering · Concept Lattice Representation · Representative Set · Order Preservation.

1 Introduction

This study is a follow-up of a preceding work about the discovery of link keys for data interlinking in RDF datasets [1,2]. A link key can be seen as a rule identifying pairs of individuals in two different RDF datasets that represent the same real-world entity. Link keys can be discovered thanks to Formal Concept Analysis (FCA) [14], where they correspond to concepts in a specific concept lattice. The conditions related to a link key are materialized by the intent of the corresponding concept while the pairs of identified individuals constitute the extent of the concept. In general, the number of link keys can be very large making their interpretation quite hard and raising the following representation problem: is it possible to design a *compact* and *representative set* of link keys

preserving a maximal number of identity links and providing navigation and interpretation capabilities to a domain analyst?

Following this track, the objective of this paper is to propose a general method for concept lattice reduction based on Agglomerative Hierarchical Clustering (AHC [15,21]). In the long term, we aim at building a tool for the graphical exploration and interpretation of a concept lattice thanks to a mapping between the concept lattice and a hierarchical partition of clusters –aka dendrogram. The granularity of the partition may be tuned by allowing an analyst to navigate the representation of a concept lattice going from a coarser clustering to a finer clustering thanks to the variation of a cut-level related to a dendrogram. Among the approaches relating clustering and FCA, the present approach is original and proposes a reliable, readable, and interpretable summary of a concept lattice. There are many possible applications such as data cleaning (duplicate detection), data exploration with a variable precision, concept interpretation, and knowledge representation in a particular domain.

More precisely we aim at constructing a compact and representative set of clusters offering interactive data analysis capabilities. Finding such a representative set is not an easy task and relies on various constraints and a compromise. Firstly, a cluster consists of a subset of concepts, where one specific concept should be selected as the "medoid", i.e., the barycenter which minimizes the distances to all other concepts in the cluster, or the supremum or infimum of the cluster if any. The representative set, abbreviated as RepSet, is composed of such medoids, one per cluster. Secondly, the RepSet set should be kept small while the largest number of objects in the concept lattice should be preserved. The size of the RepSet is related to the number of clusters it includes and is measured by a "compression rate". Thirdly, the number of preserved objects is another suitable property to be considered when computing the RepSet, which is related to the "proportion of preserved objects" (PPO) attached to the medoids composing the RepSet. Optimizing at the same time the compression rate and the proportion of preserved objects entails a controversy, but an acceptable compromise can be achieved.

For building the RepSet, we propose the CLClust algorithm that is based on AHC [15,21]. As input, CLClust takes a concept lattice denoted as CTL, a dissimilarity measure defined with respect to (w.r.t.) the concept extents, a linkage criterion, and a cutting level. As output CLClust returns a set of clusters from which a RepSet can be issued, composed of the medoids selected in the clusters. Moreover, CLClust applies AHC to FCA for reducing a possibly large concept lattice in satisfying a third constraint, i.e., the set of concepts in a cluster should respect as well as possible the ordering of concepts in CTL, as this is the case for example when a cluster corresponds to a sublattice in CTL.

In particular, this AHC process can be regarded as a general method for concept lattice reduction based on a similarity between concepts. Indeed this contrasts the reduction of a concept lattice about link keys presented in [1], which is based on crisp equality between sets in FCA and in partition pattern

structures. Here the constraint of crisp equality is relaxed and replaced by a dissimilarity used to build a partition of concepts.

The summary of the paper is as follows. In Section 2 we propose a review of papers related either to clustering and FCA or to reduction of concept lattices. Then, in Section 3 we introduce the properties of a RepSet, while in Section 4 we present the CLClust algorithm for building such a RepSet. Section 5 reports experiments performed on different datasets that demonstrate the effectiveness of the present approach and the efficiency of the algorithm in terms of lattice reduction, before conclusion.

2 Related Work

Many different methods introduced to reduce the complexity of a concept lattice are based on the clustering of the set of concepts. For example, there is a summary of classical clustering algorithms used in FCA in [22] (Section 2.9), mostly based on constructing clusters of concepts and then lattice of clusters, while alternative techniques are limiting the number of detected concepts by selecting some relevant subset or in building a new concept hierarchy.

Clustering relies on the definition of different measures to compute the similarity or proximity of concepts in order to apply a clustering algorithm. For example, three similarity measures are presented in [3], while a method combines the k-means algorithm with a quality measure in [16]. A method based on fuzzy k-means is described in [6] while [23] proposes a distance functions and a related clustering algorithm. Finally, [11] relies on pattern structures [13] and on SOFIA algorithm [8] to discover cluster candidates, and then introduces an algorithm to select non-overlapping clusters from the candidate set. In all cases the goal is the reduction of the size of the concept lattice.

Although remaining one of the main methods to reduce the dimension of a concept lattices, some alternative techniques to clustering have appeared. An extensive survey in [4] presents a classification of concept lattice reduction methods into three main categories: context pre-processing (reducing the number of attributes and yet preserving the structure of the lattice), elimination of non-essential distinctions (methods that highlight the most significant features) and concept filtration (a relevance criterion to choose formal concepts, objects, or attributes).

Another approach is presented in [10], where the reduction of a context is performed according to the properties of the implications holding in the context, while in [18] axial concepts are used to cluster data that are difficult to separate in plain FCA.

Finally, we mention some papers more related to the present paper based on the use of a congruence relation for reducing the concept lattice. In [5] a weaker notion of congruence to reduce a concept lattice is proposed, while in [9], revisited in [17], authors construct a quotient lattice based on an equivalence relation. In [25] the authors investigate how a congruence relation can be used to decompose a lattice into meaningful parts. A close approach was presented

before in [24], where authors were studying the decomposition of a lattice into smaller parts using the so-called subdirect decompositions into factors.

3 Properties of a Representative Set of Concepts

It is well-known that the number of concepts in a concept lattice can be very large. Thus it can be convenient, e.g., for an interactive analysis, to build a compact *representative set* of concepts, denoted hereafter as a RepSet. Such a RepSet can be viewed as a summary capturing as much as possible the content of the original set of concepts. In the present case, the construction of a RepSet is based on the application of AHC to an original CTL. The output of AHC is a hierarchy of clusters where a "medoid", acting as a representative of the cluster, can be selected in each cluster. The medoid in a cluster can be defined in different ways, e.g., as the barycenter minimizing the distances to all other concepts in the cluster, or the supremum/infimum of the set of concepts in a cluster when they exist.

Then, given a set of clusters generated by the AHC process, the RepSet is composed of a set of selected medoids and should satisfy three properties. The first property, "ensure a *maximal preservation of the ordinal structure of the CTL*", is related to the clustering process itself. The two other properties, namely "observe a *high compression rate*" and "observe a *high proportion of preserved objects*" should be satisfied by the RepSet as better as possible. Moreover, it should be noticed that, given a CTL, the RepSet is not unique and depends on the selection of the medoids. Below we formally present and discuss these properties.

The preservation of CTL ordering.

One requirement in applying AHC and then building the RepSet is to preserve the CTL ordering in the clustering. In particular, a cluster corresponds to a sublattice whenever the lower bound $x \wedge y$ and the upper bound $x \vee y$ of any pair of concepts x and y are also included in the cluster. In the present case, based on the fact that the dissimilarity measure used in the AHC process and introduced in the next section is not a congruence, only a "local preservation" of the lattice ordering can be achieved. However, in the CLClust algorithm, there exists an option constraining all clusters to be convex, i.e., a cluster corresponds to a chain or a sublattice in CTL.

In particular, congruences and ordering preservation are detailed in [5] while a general and extensive discussion about the preservation of a poset ordering is proposed in [7].

The compression rate of a RepSet w.r.t. CTL.

A RepSet should be minimal in size, i.e., the smaller the number of medoids in a RepSet the better the compression rate. The compression rate cr of a RepSet w.r.t. CTL is calculated as follows:

$$\text{cr}(\text{RepSet}, CTL) = 1 - \frac{|\text{RepSet}|}{|CTL|}$$

where $\top$ and $\bot$ are included in CTL, and |RepSet | denotes the number of medoids in CTL. The compression rate is ranging from 0 to 1, and the best values are close to 1, i.e., |RepSet| is small compared to |CTL|. It is equal to 0 when |RepSet| = |CTL|, i.e., there is no compression at all. The compression rate cannot be equal to 1 as the RepSet cannot be empty, i.e., it is assumed that there exists at least one concept whose extent is not empty.

The proportion of preserved objects (PPO).

In general a clustering process focuses on objects and this explains why we are more interested in maximizing the proportion of preserved objects than in maximizing the proportion of preserved attributes (the latter is an option that should be studied in future work).

Then a RepSet should preserve a maximal number of objects, i.e., the higher the number of objects lying in an extent and preserved in the RepSet the better the representativeness of the RepSet. The proportion of preserved objects PPO is calculated as follows:

$$PPO(\text{RepSet}, CTL) = \frac{|\bigcup_{md_i \in \text{RepSet}} Ext(md_i)|}{|G|}$$

where md_i denotes a medoid in the RepSet, and $|G|$ denotes the cardinality of the set of objects G which is assumed to be not empty. The proportion of preserved objects PPO ranges in $]0, 1]$, cannot be equal to zero, and is equal to 1 when all objects are preserved, i.e., RepSet = CTL.

The two former properties are in opposition as, for achieving a reliable and maximal representativeness, the number of medoids composing a RepSet should be low while the proportion of preserved objects should be high. Thus the compression rate and the proportion of preserved objects cannot be simultaneously optimized, but a good compromise can be achieved, i.e., discovering a relatively small set of medoids forming the RepSet and maximizing the compression rate, while retaining a relatively high number of preserved objects, in respecting at the best CTL local ordering.

In the next section, we make precise the construction of a RepSet, starting from a CTL and using agglomerative hierarchical clustering.

4 The CLClust Algorithm for Building a RepSet

4.1 Characteristics of the AHC Process in CLClust

The CLClust algorithm (see Function 1) constructs a hierarchy of clusters thanks to AHC [15,21]. The AHC process takes as input the set of concepts in CTL and builds a hierarchical partition of clusters. As usual, AHC is based on a bottom-up strategy, starting with each concept as a singleton cluster and then successively merging pairs of clusters w.r.t. their dissimilarity. AHC stops when all clusters are merged into a top cluster.

AHC relies on two main parameters, i.e., (i) a *dissimilarity* measure between concepts, and (ii) a *linkage criterion* for inserting a new element into a cluster. The dissimilarity between two concepts c_1 and c_2 is defined as follows:

$$\delta(c_1, c_2) = 1 - \frac{|Ext(c_1 \wedge c_2)|}{|Ext(c_1 \vee c_2)|}$$

where $c_1 \vee c_2 \neq \perp$ and $Ext(c)$ is the function returning the extent of concept c. The dissimilarity δ ranges from 0 to 1 and exhibits the following properties:

(i) $\delta(c_1, c_2) = 0$ iff $c_1 = c_2$ (no dissimilarity),
(ii) $\delta(c_1, c_2) = \delta(c_2, c_1)$ (symmetry),
(iii) $\delta(c_1, c_2) = 1$ iff $ext(c_1 \wedge c_2) = 0$.

In (iii), the lower bound of c_1 and c_2 is $\perp$ and then c_1 and c_2 are not comparable in CTL. However, when c_1 and c_2 are comparable, we may observe that the closer c_1 is to c_2 the smaller $\delta(c_1, c_2)$. For example, if $c_1 \leq c_2$, then $\delta(c_1, c_2) = 1 - |Ext(c_1)|/|Ext(c_2)|$ as $c_1 \wedge c_2 = c_1$ and $c_1 \vee c_2 = c_2$. In this way, given a concept say c_1, the least dissimilar concept c_2 from c_1 is in either the *lower cover* or the *upper cover* of c_1. In particular, intervals of concepts in CTL can be preserved and the following proposition can be stated:

Proposition 1 (see [2]). *If $c_1 \leq c_2 \leq c_3$ in CTL then $\delta(c_1, c_2) \leq \delta(c_1, c_3)$.*

This proposition, that is also discussed and proven in [2], introduces the "cover property in AHC", i.e., given three concepts, c_1, c_2, and c_3 such that $c_1 \leq c_2 \leq c_3$, c_2 will be agglomerated before c_3, and the ordering between c_1 and c_2 in CTL is locally preserved in the cluster (about metrics and order see also [20]). More precisely, the proposition ensures that, given a concept c_i lying in a cluster K_ℓ, the next concept c_j to be agglomerated in K_ℓ w.r.t. c_i is the concept the least dissimilar to c_i, and c_j is necessarily lying in the lower cover or the upper cover of c_i. In particular, this is in agreement with the properties of a RepSet, to preserve as well as possible the CTL ordering. It should be noticed that the full preservation of sublattices in the clustering requires the use of a *congruence* as explained in [5].

The second parameter required in AHC is related to the distance $dclust(X, Y)$ between two clusters X and Y, actually between a cluster and a singleton cluster. There are several alternatives detailed in [15,21]. The classical candidates are *single linkage* based on "min", *complete linkage* based on "max", and average linkage based on "mean distance". The CLClust algorithm relies on *complete linkage*, i.e., $dclust(X, Y) = \max_{x \in X, y \in Y} \delta(x, y)$.

It should be noticed that when complete linkage connects two clusters, say $X = \{x_1, x_2\}$ and $Y = \{y\}$, then the pair exhibiting the max of the dissimilarities, say x_1 and y, is selected. As x_1 and x_2 are already verifying the cover property, say $x_1 \leq x_2$, where $\leq$ denotes the lattice ordering, then x_1 and y will also be comparable. These two configurations may happen, either $y \leq x_1 \leq x_2$ or $x_1 \leq x_2$ and $x_1 \leq y$. This shows that the concept lattice ordering is again locally respected within the new cluster $X \cup Y = \{x_1, x_2, y\}$.

obj./att.	a	b	c	d	e	f	g	h	i	j	k
0	x					x			x		
1	x	x									
2			x								
3	x		x	x			x			x	x
4	x		x		x		x			x	x
5	x		x	x			x			x	x
6	x		x				x			x	x
7	x		x					x			
8										x	x
9	x		x	x	x		x			x	x

Fig. 1: The context for the running example.

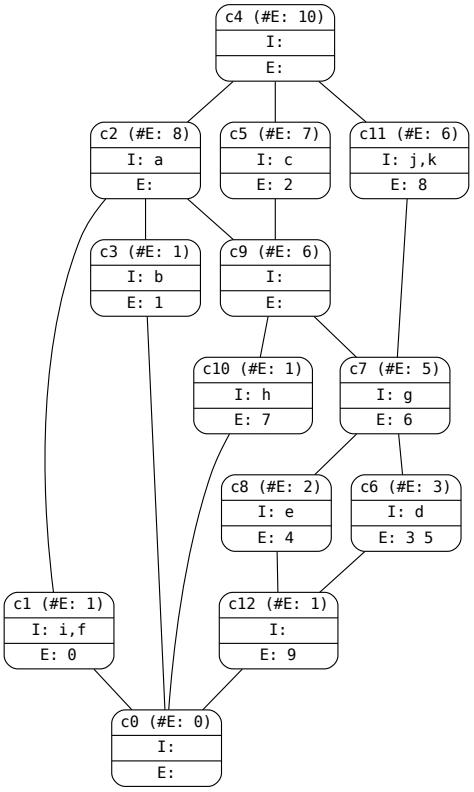

Fig. 2: The CTL lattice for the running example.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
c_0													
c_1	**1.**												
c_2	1	**0.88**											
c_3	**1.**	1	**0.88**										
c_4	1	0.9	**0.2**	0.9									
c_5	1	1	0.4	1	**0.3**								
c_6	1	1	0.62	1	0.7	0.57							
c_7	1	1	0.38	1	0.5	0.29	**0.4**						
c_8	1	1	0.75	1	0.8	0.71	0.8	**0.6**					
c_9	1	1	**0.25**	1	0.4	**0.14**	0.5	**0.17**	0.67				
c_{10}	**1.**	1	0.88	1	0.9	0.86	1	1	1	**0.83**			
c_{11}	1	1	0.5	1	**0.4**	0.5	0.5	**0.17**	0.67	0.5	1		
c_{12}	**1.**	1	0.88	1	0.9	0.86	**0.67**	0.8	**0.5**	0.83	1	0.83	

Table 1: Dissimilarity table between concepts lying in CTL. Only values in bold are needed by the CLClust algorithm.

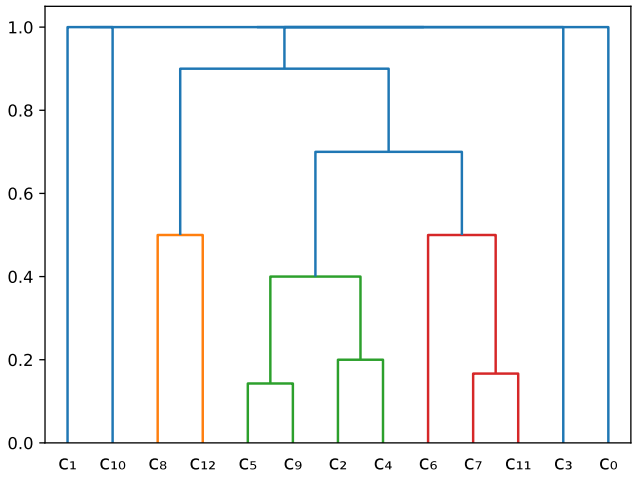

Fig. 3: Dendrogram built by AHC set up based on δ and complete linkage.

4.2 The CLClust Algorithm at Work

Let us now explain how the CLClust algorithm works in practice and propose a concrete example. AHC is initialized by computing all $\delta(x, y)$ between singleton clusters x and y and recording in a dissimilarity table, as shown in Table 1. However, in CLClust, thanks to the cover property in Proposition 1, only dissimilarities between neighboring candidates are computed, marked in bold in

Table 1. Instead of computing $n.(n-1)/2$ dissimilarity values where n is the number of clusters as in AHC, a much smaller number should be computed in CLClust, depending on the number of neighbors of the concept under study.

Moreover, at each step of AHC, the dissimilarity table should be updated, requiring the computation of $n-2$ dissimilarity values, i.e., the number of clusters minus the two clusters that are merged. Again, in CLClust, due to the cover property, this number is much smaller as it only involves neighbor concepts. This particular aspect is illustrated in the next section where the running times of standard AHC and CLClust are compared.

As a concrete example, the CTL in Figure 2 whose context is shown in Figure 1 contains 13 concepts. Thus there are 78 dissimilarity values to compute at initialization. However, as shown in the dissimilarity table in Table 1, the computation is restricted to pairs of neighbor concepts, and only 18 values are calculated in practice. At first iteration, concepts c_5 and c_9 are merged as their dissimilarity value is 0.14, the merging involving the minimal dissimilarity. Then only 5 values need to be updated among the 11 possibilities at the next step (i.e., $n-2=11$ where $n=13$). When two dissimilarities are equal, the pair of comparable concepts, if any, is preferred in the merging.

It should be noticed that, with complete linkage, it may be necessary at some steps to calculate values that were not previously computed. For example, calculating the similarity between (c_5, c_9) (after merging) and c_2 requires computing the similarity value between c_2 and c_5 which was not already computed.

The CLClust algorithm stops when all clusters are merged into one final cluster corresponding to the whole set of concepts. The output can be visualized as a *dendrogram*, i.e., a tree-based representation as shown in Figure 3. In such a dendrogram, clusters are determined thanks to a *cutting level*. Setting a cutting level mainly depends on the objectives of the analyst. More importantly, setting a cutting level will determine the RepSet. For example, a cutting level of 0.8 will give 6 clusters and a compression rate of $1-6/13 \simeq 0.54$, i.e., $\{c_1\}$, $\{c_{10}\}$, $\{c_8, c_{12}\}$, $\{c_2, c_4, c_5, c_9, c_6, c_7, c_{11}\}$, $\{c_3\}$, and $\{c_0\}$, with the related RepSet $\{\{c_1\}, \{c_{10}\}, \{c_8\}, \{c_7\}, \{c_3\}, \{c_0\}\}$. By contrast, a cutting level of 0.55 will give 7 clusters and a compression rate of $1-7/13 \simeq 0.46$, i.e., $\{c_1\}$, $\{c_{10}\}$, $\{c_8, c_{12}\}$, $\{c_2, c_4, c_5, c_9\}$, $\{c_6, c_7, c_{11}\}$, $\{c_3\}$, and $\{c_0\}$, with the related RepSet $\{\{c_1\}, \{c_{10}\}, \{c_8\}, \{c_4\}, \{c_7\}, \{c_3\}, \{c_0\}\}$. In the next section, we show how the compression rate and the proportion of preserved objects vary with the setting of the cutting level.

Even though the combination of dissimilarity $\delta(x, y)$ and complete linkage is compatible with lattice ordering, this is not sufficient to guarantee the whole partitioning into convex sublattices. The algorithm may produce non-convex groupings when the supremum is already associated with a cluster or in the event of a tie in dissimilarity values (this tie being arbitrarily resolved). Therefore the CLClust algorithm is extended with an option forcing the partitioning into convex sublattices.

CLClust algorithm makes use of a priority queue to store pairs of clusters and their associated distance values in ascending order. It first computes and adds distance values between every pair of neighboring concepts (lines 4-7). Then

Algorithm 1 The function CLClust iteratively builds the hierarchy of clusters, minimizing the number of dissimilarity values to be computed.

```
1: function CLCLUST(L : a lattice, forceConvexity : Boolean)
2:     values ← createMinPriorityQueue()
3:     ▷ Computing dissimilarity between a concept and concepts in its cover ◁
4:     for x ∈ L do
5:         for y ∈ cover(x, L) do
6:             d ← δ(x, y)
7:             add(values, ⟨x, y⟩, d)
8:     while size(values) > 1 do
9:         ▷ Select and merge the clusters with the smallest dissimilarity ◁
10:        ⟨x, y⟩, d ← extractMin(values)
11:        clust ← createClust(x, y, d)
12:        ▷ Check cluster convexity if needed ◁
13:        if (¬forceConvexity or isConvex(clust)) then
14:            ▷ Update the dissimilarity values based on complete linkage ◁
15:            for all (⟨x′, y′⟩, d′) ∈ values do
16:                if ( x = x′ or y = x′ ) then
17:                    add(values, ⟨clust, y′⟩, dclust(clust, y′))
18:                    remove(values, ⟨x′, y′⟩)
19:                else if ( x = y′ or y = y′ ) then
20:                    add(values, ⟨clust, x′⟩, dclust(clust, x′))
21:                    remove(values, ⟨x′, y′⟩)
    return extractMin(values)
```

the algorithm adopts the following greedy approach (lines 8-21). It iteratively extracts and merges the pair of clusters having the smallest distance values (lines 10-11). If the convexity is forced (line 13), the new cluster is added to the queue only if it is effectively convex. The convexity of a cluster is satisfied if the cluster has an infimum, a supremum, and if each concept in the cluster is lying in the interval between the infimum and the supremum. The addition of a cluster consists in iterating over the queue and recalculating every distance value where one of the elements belongs to the new cluster (lines 15-21).

5 Experiments

The CLClust algorithm is evaluated by using two series of experiments. Firstly, we demonstrate that CLClust builds a RepSet more efficiently w.r.t. running time than a classical AHC algorithm. We also discuss the variations related to the setting of a cutting level in the dendrogram. Secondly, the experiments evaluate both the qualities of the clusters, and the behavior of CLClust w.r.t. precision and recall in a supervised setting. These results confirm the effective capabilities of the CLClust algorithm in building a compact RepSet from a concept lattice.

dataset	#concepts	#attributes	#objects
abalone	254	9	64
atom-sites	1,034	12	41
bridges	1,266	12	63
fd-reduced-250k-30	351	26	313
flight-1k-30c	3,058	19	288
flights-20-500k	490	12	23
glass	134	10	36
iris	27	5	10
page-blocks	254	11	27
wine	117	14	76

Table 2: Statistics about the datasets considered in the experiments.

5.1 Datasets and Protocol

The experiments are performed on 10 different datasets from the UC Irvine Machine Learning Repository[5] displayed in Table 2. Every dataset is transformed as a reduced and clarified context, possibly leading to a reduction of the initial numbers of objects and attributes. The AddIntent algorithm [19] is used to compute the concept lattices whose concept count is shown in column #concepts in Table 2 . Moreover, all experiments have been conducted on a laptop equipped with an Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz and 8GB of memory dedicated to the JVM.

5.2 Evaluation of the CLClust algorithm

The two main aspects evaluated hereafter are the performance of the CLClust algorithm compared with a classical AHC algorithm and the quality of the partitioning. The experiments were carried out with four configurations based on different variations of the two following parameters: (i) the clustering method, i.e., CLClust or AHC algorithms, (ii) the convexity of a cluster is imposed or not.

For every dataset and every configuration, the running time for clustering and the number of required dissimilarities are recorded. The results are presented in Figures 4 and 5. Figure 4 only shows the three datasets in which the clustering procedure takes at least 1 second.

As expected, configurations based on CLClust algorithm are more efficient than those based on a classical AHC algorithm, as shown in Figure 4. In addition, the variation CLClust-convex works faster in general. In the case of AHC, convexity checking generates additional computational costs. In CLClust-convex, the convexity requirement reduces the number of possible groupings and therefore the number of dissimilarities to be computed as shown in Figure 5. In line with these observed runtimes, Figure 5 shows that the CLClust algorithm significantly reduces the number of calculated dissimilarities. This reduction is even

[5] `https://archive.ics.uci.edu/`

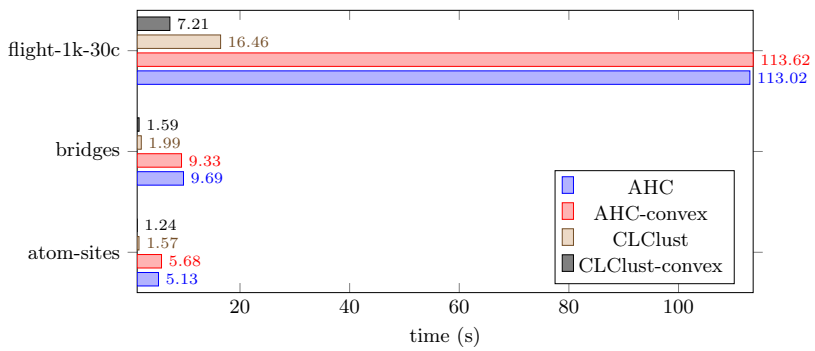

Fig. 4: Clustering running times in different configurations.

more radical when the convexity constraint is enforced. The number of calculated dissimilarities is divided on average by 3.6, with significant differences between datasets. The gain in terms of execution time is even greater than the gain in terms of calculated values: reducing the number of dissimilarities also reduces the size of the priority queue and the number of updates during grouping (lines 15-21 of Algorithm 1).

The study of the quality of a cluster partition built by CLClust is based on three criteria, namely compression rate, object preservation, and convexity. In the configurations involving CLClust and CLClust-convex, the cutting level is changed from 0 to 1 in steps of size 0.1. For every cut level, the three criteria are monitored. Figure 6 shows the evolution of the three criteria as a function of the cut level while the observations that can be made on these datasets also apply to the other datasets.

For each dataset, we can observe that, initially, compression increases faster than preservation decreases. With the exception of flight-1k-30c, a compression ratio of at least 0.5 can be achieved before preservation begins to decrease. As Figure 6 shows, when convexity is not enforced, the decrease of preservation is greater than the increase in the compression ratio. This observation can be made for all datasets except fd-reduced-250k-30 and wine datasets, for which preservation remains maximum.

With the exception of the iris dataset, convexity, when not enforced, is also stable in low cut levels, but tends to decrease earlier and faster than preservation. The stability of both preservation and convexity in low cut levels can be explained by the fact that most clusters are made up of two concepts at most (the medoid is always the top concept of such sublattices).

When convexity is required, the preservation of objects is higher and compression rate is only slightly reduced. The object preservation is higher because imposing convexity implies that every cluster has a supremum, increasing the probability that the medoid is the supremum and thus increasing the preservation of objects. For example, considering the atom-site dataset with a cut level

of 0.8, the ratio of clusters whose medoid is a supremum increases from 17% to 21% when convexity is enforced.

Among the ten datasets considered in the experiments, eight datasets achieve high compression rates such as atom-site. Only two datasets show low compression rates (< 0.5 with a cut level at 0.8), namely wine and fd-reduced-250k-30. These differences cannot be explained either by the size of datasets or by the number of concepts. More probably the reasons are related to the shapes of the lattices. Indeed, wide but shallow lattices imply many incomparable concepts, meaning that many clusters contain less than three concepts (91% in the case of fd-reduced-250k-30). This also explains why convexity is preserved in such lattices.

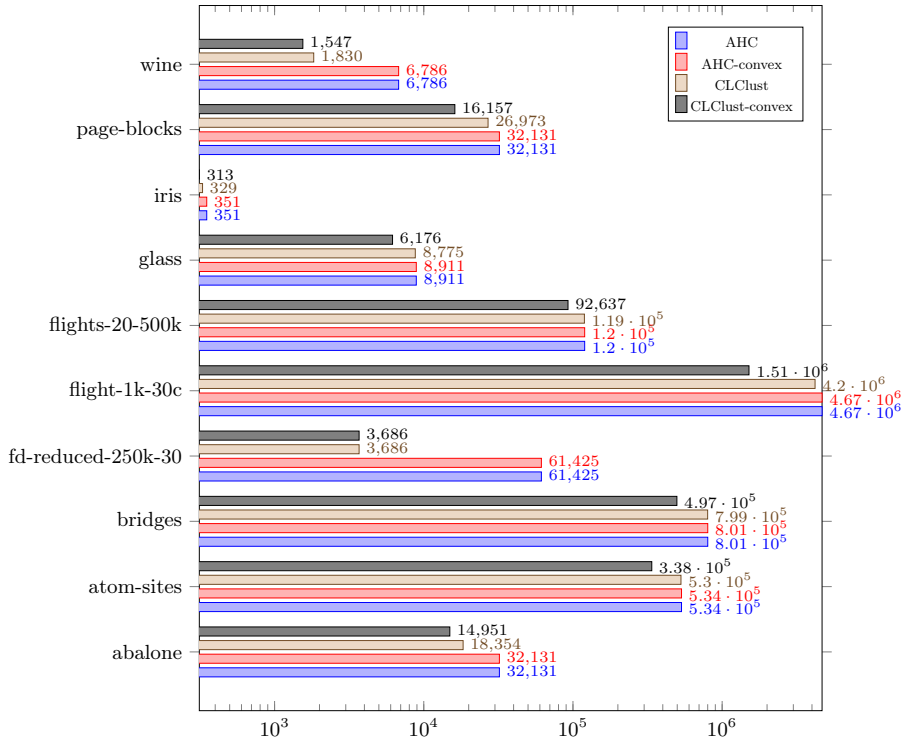

Fig. 5: The numbers of dissimilarities required in the different configurations (log scale).

To sum up, these results show that CLClust is more efficient than a classical version of AHC. In addition, combining CLClust with the convexity constraint provides the best computation times, preserves almost all the objects, with a weak effect on the compression rate.

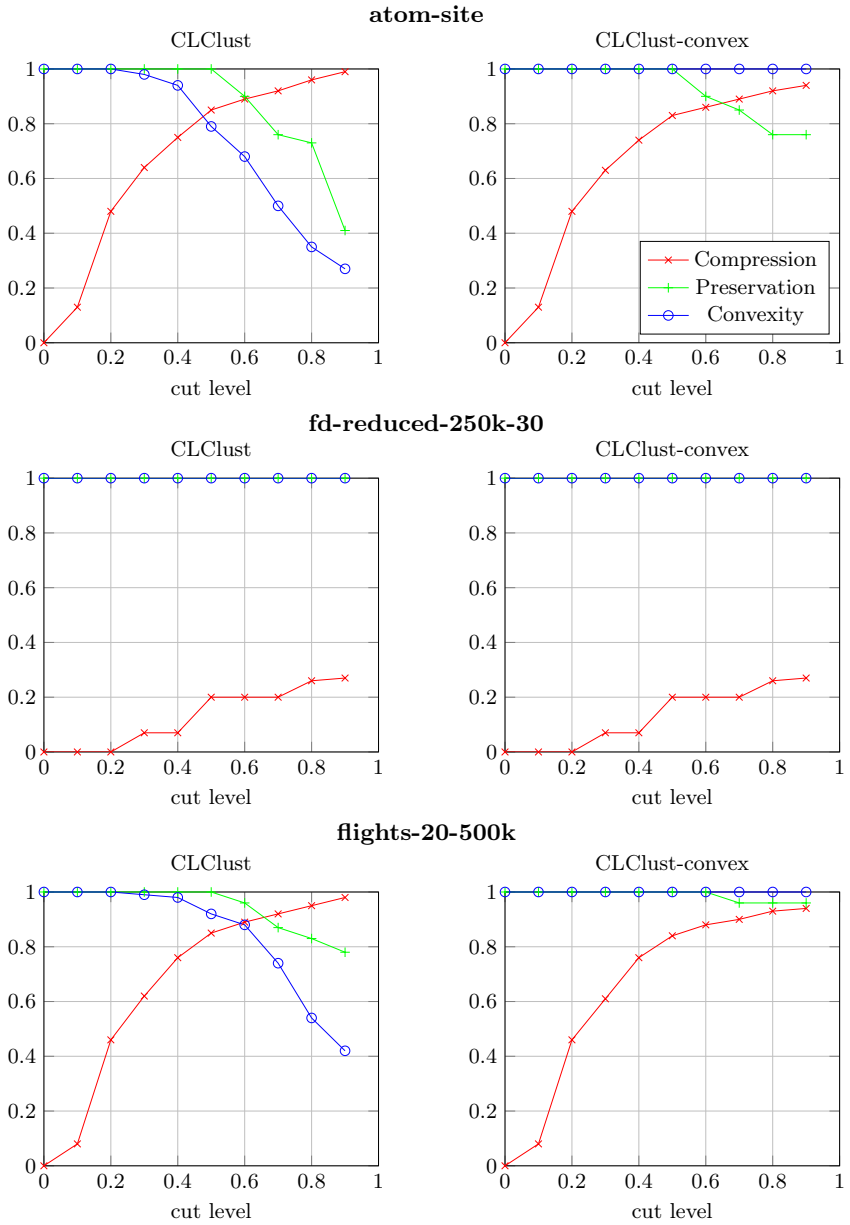

Fig. 6: The variations of compression, object preservation, and convexity w.r.t the cutting level.

6 Conclusions and Perspectives

In this paper, we are interested in reducing a concept lattice in building a representative set of clusters, named RepSet, composed of medoid concepts, for facilitating the visualization and interpretation of possibly large concept lattices. The RepSet is obtained thanks to the CLClust algorithm based on AHC, and depends on three main properties, i.e., a low number of clusters, a hight number of preserved objects, and the local preservation of the original lattice ordering.

As future work, many elements remain to be more precisely studied and several research directions can be explored. Firstly, we would like to extend the experiments and to examine alternative clustering algorithms and dissimilarity measures, and as well to investigate the impacts of the size and of the density of the dataset at hand. Then the preservation of attributes along with objects should also be questioned using alternative dissimilarity measures as proposed in [12]. Finally, we would like to more deeply study the impact of the cut level tuning on the RepSet, and the preservation of the concept lattice ordering in the clusters, possibly using congruence relations.

Acknowledgements

Jaume Baixeries is supported by a recognition 2021SGR-Cat (01266 LQMC) from AGAUR (Generalitat de Catalunya) and the grants AGRUPS-2022 and AGRUPS-2023 from Universitat Politècnica de Catalunya. Alexandre Bazin and Amedeo Napoli are carrying out this research work as part of the French ANR-21-CE23-0023 SmartFCA Research Project.

References

1. Abbas, N., Bazin, A., David, J., Napoli, A.: Discovery of link keys in resource description framework datasets based on pattern structures. International Journal of Approximate Reasoning **161**, 108978 (2023)
2. Abbas, N., Bazin, A., David, J., Napoli, A.: Discovering a Representative Set of Link Keys in RDF Datasets. In: Alam, M., Rospocher, M., van Erp, M., Hollink, L., Gesese, G.A. (eds.) Proceedings of the 24th International Conference EKAW. Lecture Notes in Computer Science, vol. 15370, pp. 53–68. Springer (2024)
3. Alqadah, F., Bhatnagar, R.: Similarity measures in formal concept analysis. In: International Symposium on Artificial Intelligence and Mathematics (ISAIM) (2010)
4. Alwersh, M., Kovàcs, L.: Survey on attribute and concept reduction methods in formal concept analysis. Indonesian Journal of Electrical Engineering and Computer Science **30**(1), 366–387 (2023)
5. Aragón, R.G., Medina, J., Ramírez-Poussa, E.: Reducing concept lattices by means of a weaker notion of congruence. Fuzzy Sets and Systems **418**, 153–169 (2021)
6. Aswanikumar, C., Srinivas, S.: Concept lattice reduction using fuzzy K-Means clustering. Expert Systems With Applications **37**(3), 2696–2704 (2010)
7. Bakkelund, D.: Order preserving hierarchical agglomerative clustering. Machine Learning **111**(5), 1851–1901 (2022)

8. Buzmakov, A., Kuznetsov, S.O., Napoli, A.: Fast Generation of Best Interval Patterns for Nonmonotonic Constraints. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD). pp. 157–172. Lecture Notes in Computer Science 9285, Springer (2015)
9. Cheung, K.S.K., Vogel, D.R.: Lattice-Based Information Retrieval Application. In: The Seventh Pacific Asia Conference on Information Systems (PACIS). p. 78. AISeL (2003)
10. Dias, S.M., Vieira, N.J.: A methodology for analysis of concept lattice reduction. Information Sciences **396**, 202–217 (2017)
11. Dudyrev, E., Zueva, M., Kuznetsov, S.O., Napoli, A.: Clustering with Stable Pattern Concepts. In: Napoli, A., Rudolph, S. (eds.) Proceedings of the 12th International Workshop FCA4AI (co-located with ECAI 2024). CEUR Workshop Proceedings, vol. 3911, pp. 47–58. CEUR-WS.org (2024)
12. Formica, A.: Ontology-based concept similarity in Formal Concept Analysis. Information Sciences **176**(18), 2624–2641 (2006)
13. Ganter, B., Kuznetsov, S.O.: Pattern Structures and Their Projections. In: Proceedings of the International Conference on Conceptual Structures (ICCS). pp. 129–142. LNCS 2120, Springer (2001)
14. Ganter, B., Wille, R.: Formal Concept Analysis. Springer (1999)
15. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: an Introduction to Cluster Analysis. Wiley (2009)
16. Kovács, L.: Conceptual clustering with application on FCA context. Expert Systems With Applications **245**, 123013 (2024)
17. Krupka, M.: On complexity reduction of concept lattices: three counterexamples. Information Retrieval **15**(2), 151–156 (2012)
18. Kuznetsov, S.O.: Clustering with Axialities. In: Napoli, A., Rudolph, S. (eds.) Proceedings of the 12th International Workshop FCA4AI (co-located with ECAI 2024). CEUR Workshop Proceedings, vol. 3911, pp. 59–66. CEUR-WS.org (2024)
19. van der Merwe, D., Obiedkov, S.A., Kourie, D.G.: AddIntent: A New Incremental Algorithm for Constructing Concept Lattices. In: Eklund, P.W. (ed.) Proceedings of the Second International Conference on Formal Concept Analysis (ICFCA). pp. 372–385. Lecture Notes in Computer Science 2961, Springer (2004)
20. Monjardet, B.: Metrics on partially ordered sets—a survey. Discrete Mathematics **35**(1), 173–184 (1981)
21. Murtagh, F., Contreras, P.: Algorithms for Hierarchical Clustering: an Overview. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **2**(1), 86–97 (2012)
22. Pérez-Suárez, A., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A.: A review of conceptual clustering algorithms. Artificial Intelligence Review **52**(2), 1267–1296 (2019)
23. Siff, M., Reps, T.W.: Identifying Modules via Concept Analysis. IEEE Transactions on Software Engineering **25**(6), 749–768 (1999)
24. Viaud, J., Bertet, K., Demko, C., Missaoui, R.: Subdirect Decomposition of Contexts into Subdirectly Irreducible Factors. In: Ojeda-Aciego, M., Baixeries, J., Sacarea, C. (eds.) Proceedings of the International Workshop on Formal Concept Analysis and Applications (FCA&A, co-located with ICFCA 2015). CEUR Workshop Proceedings, vol. 1434, pp. 49–64. CEUR-WS.org (2015)
25. Viaud, J., Bertet, K., Missaoui, R., Demko, C.: Using congruence relations to extract knowledge from concept lattices. Discrete Applied Mathematics **249**, 135–150 (2018)