Analysis of Co-Occurrence Patterns in Data through Modular and Clan Decompositions of Gaifman Graphs

Marie Ely Piceno         José Luis Balcázar

October 14, 2019

Abstract

Our team has recently demonstrated that clan decompositions of generalized Gaifman graphs provide advantageous hierarchical visualizations of co-occurrence patterns in data. We develop that early intuition by introducing a construction of implication sets, named “clan implications”, and then, explaining these clan decompositions as a variant of closure spaces associated to these implications. Further, we discuss some algorithmic issues that support reasonably efficiently our exploratory data analysis software implementing this approach.

1 Introduction

Recent work of our team has demonstrated that the decomposition of Gaifman graphs has the intuitive potential of providing hierarchical visualizations that explain data in interesting ways; also, some simple generalizations of Gaifman graphs enhance this potential [1]. Specifically, our team has shown that, in a context of diagnostic data from medical practice in a cooperating hospital, the diagrams we can obtain provide visual, interpretable hints about patterns of diagnostics that co-occur in the data [2], [12].

These hierarchical visualizations provide easy-to-grasp explanations of co-occurrence patterns in the given data. This paper provides a fundamental study of the process behind these applications, by explaining how the modular decompositions of standard Gaifman graphs, and also the more general clan decompositions of their variants, fit a specific view of closure spaces that correspond to so-called “modular implications”, resp. “clan implications”, that we introduce here. This allows us to connect our approach to more standard tools in Data Analysis, such as closure and implication mining. We describe as well some algorithmic decisions that support the system that we are currently developing in order to make applicable our proposal to other fields.

*Partially supported by European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme, grant agreement ERC-2014-CoG 648276 (AUTAR); by grant TIN2017-89244-R from Ministerio de Economía, Industria y Competitividad, and by Conacyt (México). We acknowledge unfunded recognition 2017SGR-856 (MACDA) from AGAUR (Generalitat de Catalunya).
2 Preliminaries

Our work builds upon both the known connection between closure spaces and implications, and on several proposals for identifying hierarchical decompositions of graphs, namely, modules and, in a more general way, clans of 2-structures. The theory of 2-structures and their clans will be provided later on, after motivating the need of using them for certain generalizations of Gaifman graphs; initially, however, we will work with standard Gaifman graphs and we will only need modular decompositions. We explain these notions in this section.

2.1 Set-Theoretic Notations

We assume that a given set $\mathcal{U}$ of potential items of interest has been fixed. Datasets will be assumed to come in transactional form, where each transaction is simply a subset of items. Set-theoretic notations are standard; $X \setminus Y$ denotes set difference: the elements of $X$ that do not appear in $Y$. The complement is $\mathcal{U} \setminus X$.

**Definition 1** Two sets $X$ and $Y$ overlap if the sets $X \cap Y$, $X \setminus Y$, and $Y \setminus X$ are all three nonempty. Equivalently, $X$ and $Y$ do not overlap if and only if they are either disjoint, or a subset of one another.

In many cases, items of $\mathcal{U}$ will be vertices of graphs in a fully standard way. In all our graphs, we assume that no self-edge is ever present; also, we do not allow for multiple edges.

2.2 Implications and Closures

Closure spaces on the powerset of the set of items will play an important role in our developments. A closure operator maps each set of items $X$ to a set of items, its “closure” $\overline{X}$; they are characterized by three properties:

- Extensivity: $X \subseteq \overline{X}$
- Idempotency: $\overline{\overline{X}} = \overline{X}$
- Monotonicity: if $X \subseteq Y$ then $\overline{X} \subseteq \overline{Y}$

A set is a closed set if it coincides with its closure, and the closure space is the family of all the closed sets. A basic fact from the theory of closure spaces is that the intersection of closed sets is again closed.

Implications are conjunctions of definite Horn clauses, where a set of clauses sharing the same antecedent are written as a single formula with conjunctions both at the antecedent and the consequent of an implication connective; we often write antecedents and consequents as mere sets, letting the conjunction connective implicit. For sets of items $X$, $Y$, and $Z$, $Z$ satisfies the implication $X \Rightarrow Y$, denoted as $Z \models X \Rightarrow Y$, if either $X \not\subseteq Z$ or $XY \subseteq Z$.

The fact, well-known in logic and knowledge representation, that Horn theories are exactly those closed under bitwise intersection of propositional models leads to a strong connection with closure spaces. It runs as follows: given a set of implications $\mathcal{B}$, the closure $\overline{X}$ of a set $X$ is the largest set $Y$ such that $\mathcal{B}$ logically entails $X \Rightarrow Y$; whereas, if we are given a closure operator, we
can axiomatize it by the set of implications \( \{ X \Rightarrow Y \mid Y \subseteq X, X \subseteq U \} \) or, equivalently, any set of implications that entails exactly this set.

See [15] (or its earlier version https://arxiv.org/abs/1411.6432v2) for references and supporting facts of all our claims so far.

**Definition 2** \( X \) is a strong closure if, for all other closures \( Y, X \) and \( Y \) do not overlap. That is, either \( X \) and \( Y \) are disjoint, or a subset of one another.

### 2.3 Modular Graph Decompositions

The modular decomposition of a graph is a process that consists of decomposing the graph into sets of vertices, nowadays called “modules”. The earliest appearance of the notion seems to be [7] but it has been rediscovered many times and described under many different names [8]. Since modules can be proper subsets of other modules, we can obtain a hierarchical decomposition of the graph. These facts have been studied in a very wide bibliography (see [8] and the references there).

**Definition 3** Given a graph, a set \( X \) of vertices is a module if, for each vertex \( y \notin X \), either every member of \( X \) is connected with \( y \) or every member of \( X \) is not connected with \( y \).

The modules of a graph can be seen, therefore, as subgraphs of the original graph. Note that the set of all vertices is vacuously a module, as are each vertex by itself and the empty set. These are called the trivial modules; we will systematically ignore the empty module.

As an intuition device that will become full-fledged in subsequent sections, we take the somewhat anthropomorphic stance of considering the presence or absence of an edge as the way one vertex “sees another”. Thus, we will often resort to expressions like “an item sees in a different way two other items” when it is connected to one of them and disconnected from the other one; or, in the same case, we may say that “one item can distinguish between other two items”. This intuition is customary in the field of 2-structures where the notion of a “clan”, a generalization of modules defined also below in Section 4, relies on this intuitive device.

Hence, throughout this paper, the presence or absence of edges will convey only the idea of two different ways one item sees another; that is, we give the same interpretation to a graph and to its complement. Indeed, permuting absence and presence of all edges does not change the ways vertices distinguish each other and, therefore, leaves the same set of modules. On the basis of the definition of module, we can state (see [8]):

**Proposition 4**

1. Let \( x \) and \( y \) be elements of a module \( M \): if item \( z \) sees in a different way \( x \) and \( y \) then, necessarily, \( z \in M \).

2. Let sets \( X \) and \( Y \) be non-disjoint: \( X \cap Y \neq \emptyset \). If both are modules, then also \( X \cap Y, X \setminus Y, \) and \( Y \setminus X \) are modules.

A main interest of the notion of module is that all the vertices of a module can be collapsed into a single vertex without ambiguity with respect to how to

---

An English translation is available as [9], where they are called “homogeneous sets”.

---

3
connect it to the rest of the vertices: the new vertex gets connected to \( y \) if all the members of the module were connected to \( y \), and remains disconnected if all the members were disconnected. Clearly, the definition given of module is what is needed for this process to be applied without hesitation about whether the new vertex should or should not be connected to some external vertex \( y \). More generally, the same considerations apply if we consider two disjoint modules: either they are connected, in the sense that all the respective pairs of vertices (one from each module) are, or they are not because no such pair is.

Nothing forbids modules to intersect each other; in that case, though, collapsing one module into a single vertex may affect the other. In order to avoid side effects, it is customary to restrict oneself to so-called “strong modules” [7] (see also [9]): they allow us to obtain a tree-like decomposition.

**Definition 5** A module \( M \) is a strong module of a graph if it does not overlap any other module; that is, for all other modules of the graph \( M' \), either \( M \cap M' = \emptyset \) or they are subsets of one another.

Given a graph, we can focus on its maximal strong modules; it is known that each vertex belongs to exactly one of them [8]. Thus, one or more (or even all) of these maximal strong modules can be collapsed into a single vertex each. The resulting graph is called a “quotient graph”. The coarsest quotient graph is obtained by collapsing all the maximal strong modules.

Then, a tree-like structure arises from the fact that each of these modules, taken as a set of vertices, is actually a subgraph that can be recursively decomposed, in turn, into maximal strong modules, thus generating views of subsequent internal structures given by their respective coarsest quotient graphs. We display the decomposition tree while labeling each node (a strong module) with the corresponding coarsest quotient graph, and connect visually each collapsed vertex to the subtree decomposing the corresponding module. Of course, the root of the “tree” is the coarsest quotient of the whole graph.

**Example 6** Figure 1 shows a first, simple example of modular decomposition. It is easy to check that \( \{a, b, c\} \) forms a strong module: both \( d \) and \( e \) are connected to each of them. The other possible nontrivial modules are \( \{a, b, c, d\} \), \( \{a, b, c, e\} \), \( \{a, b, e\} \), and \( \{b, c, d\} \).
\{d, e\}, \{a, b\}, \{a, c\}, and \{b, c\}; but they are not strong: each of them intersects others. The root is the fully connected coarsest quotient graph where \{a, b, c\}, the single nontrivial maximal strong module, has been collapsed to a single vertex. Each of the three vertices is connected to the module they represent: two are maximal strong but trivial ones, and the largest one decomposes itself again into three trivial modules.

Further examples will appear below. Our depictions depart somewhat from the standard drawings in previous studies of modular decomposition and, instead, follow the customary representations of clan decompositions of 2-structures, described below.

3 A Closure-Based View of Modules

Given a graph, it is possible to describe the conditions for a subgraph being a module in the form of a set of implications. This approach leads to a connection between the tree-like decomposition described above and the theory of closure spaces. Through them, we also connect to data analysis by closure mining. In this section we show the procedure that we propose to obtain these implications and some of the resulting connections.

As already indicated, if \(x\) and \(y\) are elements of any module \(M\) and \(z\) sees them in a different way, then \(z \in M\). With the standard semantics of implications, taking vertices as propositional variables and, hence, subgraphs as models, this is equivalent to: \(M \models xy \Rightarrow z\).

To pursue this idea further, given a graph, we generate a set of implications from it. We call this set the set of modular implications of the graph. In each implication, the antecedent will be a pair of vertices in the graph, and the consequent will be conformed by those vertices that see the vertices in the left side in different ways.

Formally, we need to establish some notation. Given a graph \(G\) and a vertex \(x\) in it, we denote by \(N_G(x)\) the set of immediate neighbors of \(x\) in \(G\), and define the “distinguishing set” of a pair of vertices \(x, y\), as follows:

\[ D_G(x, y) = (N_G(x) \setminus N_G(y)) \cup (N_G(y) \setminus N_G(x)) \setminus \{x, y\}. \]

That is, this set gathers together all the vertices that see one given pair in different ways. Note that it is necessary to remove explicitly \(x\) and \(y\) from the distinguishing set. Indeed, in case they are connected, the absence of self-edges would make each of them qualify to distinguish themselves from the other, whereas the definition of module only searches for distinguishing vertices outside the module. Then, we construct the set of modular implications as follows:

**Definition 7** For graph \(G\) and different vertices \(x, y\) in it, the corresponding modular implication is \(xy \Rightarrow D_G(x, y)\). The set of modular implications for \(G\) is formed by the modular implications corresponding to every pair of different vertices, for which the right-hand side is nonempty: \(D_G(x, y) \neq \emptyset\).

We can state the following:

**Proposition 8** For graph \(G\) and different vertices \(x, y\) in it, in the corresponding modular implication \(xy \Rightarrow Z\) we have \(z \in Z\) if and only if \(z \notin \{x, y\}\) and \(z\)
is connected to exactly one of \( x, y \). Therefore, \( \{x, y\} \) is a module (of size 2) if and only if \( Z = \emptyset \).

The proof is direct from the definition of module.

**Example 9** Let us continue the previous example by showing the set of modular implications from the same graph and the corresponding closure space. From Figure 1 we obtain the following set of modular implications:

\[
\begin{align*}
\text{ad} & \Rightarrow \text{bc} & \text{bd} & \Rightarrow \text{ac} & \text{cd} & \Rightarrow \text{ab} \\
\text{ac} & \Rightarrow \text{bc} & \text{be} & \Rightarrow \text{ac} & \text{ce} & \Rightarrow \text{ab}
\end{align*}
\]

The pairs \( \{a, b\}, \{a, c\}, \{b, c\}, \{d, e\} \), are size-2 modules and would lead to empty right-hand sides in their modular implications; accordingly, these implications are discarded.

In the left side of Figure 2 we find the closure space lattice described by the modular implications obtained from the graph. For convenience in the comparison, we display again at the right the decomposition from Figure 1, and we mark in bold those closed sets that do not overlap any other closed set (strong closures).

The main result of this section is Theorem 10; its proof is in Appendix A

**Theorem 10** The modules of a graph and the closures defined by its set of modular implications are the same sets.

**Corollary 11** Strong modules and strong closures are the same sets.

Thus, the match seen in Figure 2 is not chance. The closures that are not strong do correspond exactly to the additional modules described at the end of Example 6. In the following, and until explicitly indicated otherwise, whenever we talk about closures, we are referring to the closures described by the set of modular implications.
3.1 Module Taxonomy

To maintain consistency within the paper, for some notions we will be using terminology corresponding to the theory of 2-structures instead of the classical terms in modular decompositions. A graph of three or more vertices is called primitive if all of its modules are trivial, while if all of its vertices are connected, or if all of its vertices are disconnected, we say that it is complete. So, as an alternative definition, a graph is primitive if, for any two vertices in the graph, there is a third one that sees them in a different way. Both nonsingleton quotient graphs in Figure 1 are complete, even if the graph itself is not. By convention, graphs of size 1 or 2 are considered complete.

A \( P_4 \) is a graph consisting of a path on 4 vertices with 3 edges; its complement is also \( P_4 \).

**Proposition 12** Primitive graphs (and thus also their complements) always have induced \( P_4 \) subgraphs \( [14] \). Hence, for a primitive module, its corresponding coarsest quotient graph consists of at least four maximal strong modules.

Modules can be primitive or complete themselves as well, according to their corresponding coarsest quotient graphs. Indeed, the terms are most often applied to coarsest quotient graphs, such as those labeling the nodes of a tree decomposition. Specifically, it is a theorem of modular decomposition that the coarsest quotient graphs labeling each node of a decomposition tree, if nontrivial, are all either primitive or complete. Primitive quotient graphs necessarily include an induced \( P_4 \), and graphs whose tree does not show any primitive quotient graph are called “cographs” and have a large number of graph-theoretic and algorithmic properties.

Among other terms, primitive modules are sometimes called neighborhood modules. Modular decomposition theory distinguishes fully disconnected complete graphs (or “parallel” modules) and fully connected ones (or “series” modules), leading to often duplicated arguments and definitions because both cases fulfill the same role. Indeed, recall from Section 2.3 that presence or absence of edges can be swapped with no change in the modular structure: that’s why fully connected and fully disconnected modules are treated similarly. Hence, we prefer the (2-structures-based) view of naming them both “complete”.

3.2 Closures and module types

By Theorem 10 and Corollary 11 we have that modules and closures, strong or not, are the same sets. As we shall see below, the strong modules and their types will be important in our visualization of co-occurrence patterns in data analysis. Thus, we would like to be able to get the type of a strong module, that is, whether the module is complete or primitive, by using just the information in the closure lattice.

Following this idea, we show in this section that the type of a strong module is given by the immediate closure subsets of its respective strong closure. The immediate closure subsets of a set \( X \) are the children of \( X \) in the closure lattice: closed (not necessarily strong) subsets \( Y \) of \( X \) such that no intermediate set \( Z \), \( Y \subset Z \subset X \), is closed.

\(^2\)See also [https://en.wikipedia.org/wiki/Cograph](https://en.wikipedia.org/wiki/Cograph) and the references there.
Theorem 13 The type of the coarsest quotient graph of a module is primitive if and only if the immediate closure subsets of its corresponding closure in the closure lattice are strong closures and they are more than three.

The proof of this theorem is in Appendix A. Let us see some examples to illustrate both cases. In Figure 3, left is a graph (known as the “gem graph”) with a primitive quotient graph into its decomposition. By Proposition 12, a primitive quotient graph of 4 vertices must be indeed $P_4$. We get from it the following set of graph implications: $ab \Rightarrow c, ac \Rightarrow bd, ad \Rightarrow b, ae \Rightarrow bc, bd \Rightarrow d, be \Rightarrow ac, bd \Rightarrow a, cd \Rightarrow a, ce \Rightarrow ad, de \Rightarrow c$. This implication set generates the closure lattice in the center of Figure 3. The node $abcde$ has two children, so its respective module in the decomposition tree is complete, we may see this in the right of Figure 3. The node $abcd$ has four strong closures as children, and indeed its equivalent strong module into the decomposition is primitive as Theorem 13 says.

In Figure 4, there is an example of a graph with only complete quotient graphs into its decomposition. From Figure 4, left we get the set of graph implications: $ac \Rightarrow b, ad \Rightarrow ce, ae \Rightarrow b, bc \Rightarrow a, bd \Rightarrow ce, be \Rightarrow a, cd \Rightarrow a, ce \Rightarrow abc, de \Rightarrow abc$ with $ab$ and $ce$ closed sets. At the center of Figure 4, we have the closure lattice from this set of implications. As we can see, the closed set $abcde$ has two children so its respective quotient graph in the decomposition is a complete module. The closed set $abce$ has three children overlapping, by the Theorem 13 we may deduce its respective coarsest quotient graph in the decomposition is complete as we can see in the Figure 4 right. The last closed set in the closure lattice is $ab$, it has two children so its respective quotient graph is a complete module as we may see in the decomposition tree.
3.3 Data Analysis via Gaifman Graph Decomposition

Gaifman graphs are logical mathematical structures, whose basic notion is pretty simple. Given a first order relational structure \( \{ R_i \}_{i \in I} \) where the values appearing in the tuples of the relations \( R_i \) come from a fixed universe \( U \), its corresponding Gaifman graph has the elements of \( U \) as vertices, and the edges \((x, y)\), for \( x \neq y \), are determined exactly when \( x \) and \( y \) appear together in some tuple \( t \in R_i \) for some \( R_i \). On this graph is, of course, possible to apply the modular decomposition method.

This multirelational potential will be explored in future work; we discuss further this issue in our last section. For the time being, we restrict ourselves to Gaifman graphs of single tables (thus applicable to most of the commonly employed benchmark datasets). To analyze such a table in terms of its Gaifman graph, the graph will have as vertices all the attribute values, and the edges \((x, y)\), for \( x \neq y \), are determined when the attribute values \( x \) and \( y \) appear together in some tuple. Items in transactional data are handled similarly.

Often, in practice, these graphs may have huge amount of vertices and, in order to get a smaller but representative version, we choose to work with those vertices that appear into the transactions more frequently than a determined threshold. Once we have the desired Gaifman graph, it is possible to apply the modular decomposition method that groups the vertices in a hierarchical way according to their co-occurrences. In the Appendix B we give some examples where the Gaifman graphs are defined in this standard manner and also as defined in a variant proposed in earlier papers.

4 2-structures and clan decomposition

The notion of modular decomposition \[7\] is enough to be applied on standard Gaifman graphs, but is insufficient to handle adequately other variations of Gaifman graphs that we will study also; therefore, for the rest of the paper, we will work with a more general notion, namely, 2-structures and their clans \[6\].

**Definition 14** A 2-structure is a complete graph on some universe with an equivalence relation among its edges.

In specific, we work with a special kind of 2-structures, the so called reversible 2-structures \[6\] where all the equivalence classes are symmetric. An equivalence class \( E \) on a set \( U \) is symmetric if for all \( x, y \in U \): \( xEy \iff yEx \). Since in the 2-structures that we use the edges represent co-occurrences, is the same to say that \( y \) co-occurs with \( x \) than \( x \) co-occurs with \( y \). Then, the equivalence classes of co-occurrences are symmetric equivalence classes. Hence, in the following when we talk about 2-structures we will be referring to reversible 2-structures.

It is possible to see a Gaifman graph as a 2-structure where all the absent edges will be in the same equivalence class while the existing edges will belong to the other equivalence class. In this way we get a complete graph with two equivalence classes. The advantage of working with 2-structures is that they allow us to work with more than two equivalence classes.

We extend here the intuitive concept of when a vertex sees in different ways two vertices. In 2-structures we say that an item “sees in a different way” two other items if the edges that connect it with these two are in different equivalence classes.
classes. That is, \( x \) sees in a different way \( y \) and \( z \) if the edges \((x, y)\) and \((x, z)\) are in different equivalence classes. Alternatively, we say \( x \) can distinguish between \( y \) and \( z \). Note that this is consistent with the previous usage when a graph is seen as a 2-structure with two equivalence classes.

The notion now corresponding to “module” has been called always a “clan”. For a 2-structure given by a set of vertices \( U \) and an equivalence relation \( E \) on its edges, we say that the subset \( X \subseteq U \) is a clan, informally, if, for every \( y \in U \setminus X \), \( y \) cannot distinguish the elements of \( X \). Formally:

**Definition 15** Given \( U \) and an equivalence relation \( E \subseteq ((U \times U) \times (U \times U)) \) on the edges of the complete graph on \( U \), \( C \subseteq U \) is a clan when

\[
\forall x \notin C \forall y \in C \forall z \in C ((x, y), (x, z)) \in E.
\]

Thus, two members of a clan cannot be distinguished by anyone outside the clan. As well as the trivial modules, the so called trivial clans are all the singletons \( \{x\} \) for \( x \in U \), as well as \( U \) itself and the empty set. In order to display the decomposition into a tree-like form we look again for non-overlapping clans.

**Definition 16** For a fixed 2-structure on universe \( U \), a clan \( X \subseteq U \) is a strong clan if \( X \) does not overlap any other clan.

Originally, strong clans were called “prime clans”. However, in the context of modular decompositions, the adjective “prime” has received other usages in the literature. We have deemed better to avoid that adjective, so that previous exposure of the reader to either modular decompositions or 2-structures does not result in misunderstandings.

As with modules, strong clans can be collapsed into single vertices without any ambiguity about how the 2-structure looks like after the collapse. The corresponding notion of coarsest quotient 2-structure follows by the same procedure as with modules: collapsing to single vertices the maximal strong clans. Then, each of these clans is decomposed recursively so as to obtain a tree-like decomposition much like those we have already seen with modules.

The internal structure of a clan is also a 2-structure, the sub-2-structure that involves the nodes into the clan. According to the internal 2-structure of its coarsest quotient, each clan can be classified as a complete or a primitive clan. In a complete clan all the edges are in the same equivalence class while in a primitive clan there are nothing but trivial clans.

It is a theorem of the theory of 2-structures that the nodes of the tree decomposition of a reversible 2-structure are all primitive or complete clans. This is the natural generalization of the modular decomposition of graphs. Recall that this paper only employs reversible 2-structures all along.

## 5 Closures from 2-structures

In the same way as we did with Gaifman graphs, it is possible to obtain the set of implications that describe a 2-structure. Again, from the definition of clan: if \( x \) and \( y \) are elements of any clan \( C \) and \( z \) sees them in a different way, then \( z \in C \). This is equivalent to \( C \models ab \Rightarrow c \), taking the standard semantic of implications, the vertices in the graph as propositional variables, and subgraphs as models. We call this kind of implications “clan implications”.

10
Let us see an example of get the sets of implications from a 2-structure and its corresponding closure space. In an illustrative way, we represent the different equivalence classes by different type of lines. From the Figure 5 we obtain the follow graph implications set:

\begin{align*}
ab & \Rightarrow c \\
ac & \Rightarrow b \\
ad & \Rightarrow ce \\
ae & \Rightarrow cd \\
bc & \Rightarrow a \\
bd & \Rightarrow ce \\
be & \Rightarrow cd \\
bd & \Rightarrow abc \\
\end{align*}

The pair \(de\) is a closed set. In Figure 6 we find the closure space lattice (left) described by the graph implications set and the clan decomposition tree (right) of the graph in Figure 5.

Formally, given a graph \(G\) and a vertex \(x\) in it, we denote by \(E_i, G(x)\) the set of vertices connected with \(x\) by edges belonging to the equivalence class \(E_i\) in \(G\), and define the “distinguishing set” of a pair of vertices \(x, y\), as follows:

\[
\mathcal{D}_G(x, y) = \bigcup \left((E_i, G(x) \setminus E_i, G(y)) \cup (E_i, G(y) \setminus E_i, G(x))\right) \setminus \{x, y\}
\]

This set collects together all the vertices that see one given pair in different ways. Then, we construct the set of clan implications as follows:

**Definition 17** For graph \(G\) and different vertices \(x, y\) in it, the corresponding clan implication is \(xy \Rightarrow \mathcal{D}_G(x, y)\). The set of clan implications for \(G\) is formed by the clan implications corresponding to every pair of different vertices, for which the right-hand side is nonempty: \(\mathcal{D}_G(x, y) \neq \emptyset\).

From the definition of clan we can state the following:
Proposition 18 For graph $G$ and different vertices $x$, $y$ in it, in the corresponding clan implication $xy \Rightarrow z$ we have $z \in Z$ if and only if $z \notin \{x, y\}$ and $z$ is connected to each of $x$, $y$ by inequivalent edges. Therefore, $\{x, y\}$ is a clan (of size 2) if and only if $Z = \emptyset$.

5.1 Connection with standard implications and standard closure space

We may extend Theorem 10 and Theorem 13 to clans in a natural way, getting the theorems:

Theorem 19 1. Clans and closures from the graph implications are the same sets. Implying as well that strong clans and strong closures are the same sets.

2. The type of a clan is primitive if and only if its immediate subset clans are strong clans and they are more than two.

The proof of both theorems are similar to the proof of Theorem 10 and Theorem 13 but having clans instead of modules. The difference between Theorem 13 and Theorem 19 is the number of strong nodes that are required because when we work with more than two equivalence classes is possible to have a primitive node with just three internal nodes.

For example, for the primitive coarsest quotient graph that collapses the maximum strong clans $C_0$, $C_1$ and $C_2$ (of course all of them are connected with edges in different equivalence classes) the possible unions of two of their internal items are not closed sets because we will have the clan implication that involves the third one in the right side.

Unless explicitly indicated otherwise, whenever we talk about closures in the following, we refer to the closures described by the set of clan implications.

It is possible to construct the clan decomposition tree from the closure space lattice getting by the implications obtained from the initial 2-structure. If we take from the closure space just the strong closures, they would be the same sets than the strong clans by Theorem 19 and the type of the clans are also obtained using Theorem 19. To give the internal 2-structures of each clan is a more complicated task since we do not have information that allows us to determine the equivalence class for every edge. Some advances along this line will be reported in a future paper.

5.2 Data Analysis via Generalized Gaifman Graphs

As we indicated earlier, there are further variations of Gaifman graphs, namely labeled Gaifman graphs. In a labeled Gaifman graph, the edges are labeled by the multiplicity of the vertices that they connect, that is the number of tuples containing both values, the co-occurrences of the attribute values into the dataset, or by a label obtained from this multiplicity via some sort of discretization process. In practice, the simple strategy of having as many equivalence classes as different multiplicities leads to so many equivalence classes that most vertices distinguish most others, and no nontrivial clan shows up. Thus, we study here two variations of the labeled Gaifman graphs, based on very simple
discretizations, and leave for future work a detailed study of the effect of applying the various existing discretization methods to gather the diverse multiplicity figures into a sensible quantity of equivalence classes.

The two variations we study here were introduced in our earlier work [1]: the linear and the exponential Gaifman graphs. In the linear Gaifman graphs, the equivalence classes are determined according to an interval size while, in the exponential Gaifman graphs, the width of the intervals in which the equivalence classes are determined grow in an exponential way.

Let \( c_{x,y} \) be the co-occurrences of the attribute values \( x \) and \( y \), and let \( i \) be such that represents the equivalence class \( E_i \).

In a linear Gaifman graph the edge \((x, y) \in E_i \), if and only if \( i \) is equal to the ceil function of \( c_{x,y}/n \) being \( n \) the assigned interval size.

While in an exponential Gaifman graph, the edge \((x, y) \in E_i \), if and only if \( i \) is equal to the floor function of \( c_{x,y} \) logarithm base two; unless \( c_{x,y} = 0 \), in this case \( i \) will be zero.

As you can see if \( c_{x,y} = 1 \), \( i \) also will be zero, thus we have to take care on this definition because some times we would not want to lose these edges.

We also combine these approaches with the thresholded version of the Gaifman graph. In this version all those \( c_{x,y} \) below a determined threshold will be consider as they do not occur to often enough and will be disconnected.

It could be possible to work also with upper threshold where those \( c_{x,y} \) greater than it are disconnected, but here we only work with lower thresholds.

All these variations require us to work on a 2-structure since we have more than two equivalence classes. In Appendix C we give some examples of linear and exponential Gaifman graphs decompositions.

### 6 Decomposition algorithm

There are many published algorithms to construct modular or clan decompositions; most of them require dedicated, quite complex data structures that, as indicated in [3], are unlikely to yield practical algorithms. Also, only a few are incremental and, according to [3], a fully dynamic solution only exists for cographs (where no primitive nodes appear in the decomposition) and the general case is open.

We develop the explanation of our algorithm following roughly the same scheme as in the one in [11]. Recall from Section 2.3 that presence or absence of edges can be swapped with no change in the modular structure; that is why fully connected and fully disconnected modules can be treated similarly. In our algorithm, we handle them as complete clans and keep some label that is able to identify the equivalence class of its edges, which we call the color of the clan.

We describe our algorithm by focusing on the effect of adding one more vertex to an existing decomposition tree, starting at the root: a clan, represented as its coarsest quotient graph, so that each node there corresponds to a subtree decomposing the corresponding maximal strong subclan; then, walking towards the leaves recursively. By comparing the root clan with the new vertex, we can divide the nodes into the coarsest quotient of the root clan into three lists: the list of nodes visible with the color of the clan, the list of other visible nodes and the list of nonvisible nodes.
The list of nonvisible nodes will contain all those nodes some of whose internal elements are seen in different ways from the new vertex, thus we cannot assign a single edge color from the new vertex to these nodes. The list of nodes visible as the color of the clan only applies when the root clan is complete, and will contain those nodes whose edges to the new vertex are all in the same equivalence class than the internal edges of the clan; this list is empty for primitive clans. The list of visible nodes contains the rest of the nodes in the clan.

Nonvisible nodes are handled by the “split” function from Algorithm 3. These nonvisible nodes are nontrivial clans otherwise they have a defined color. Basically, the split function goes as deep as is necessary into the clans until finding visible nodes.

As initial cases we have: add the new vertex to an empty tree or to a tree whose root is a clan conformed just by one vertex. In both cases the new vertex is added as an element of the clan and the clan type is complete. If we are not in any of these initial cases we follow the next steps:

1. If the current clan is a complete clan, the new vertex can:
   
   (a) Be one member of the clan, preserving the type of the clan: when all the internal nodes of the current clan are in the list of visible as the color of the clan nodes.
   
   (b) Generate a subclan: when some of the nodes, but not all, are in the list of visible as the color of the clan nodes. Thus, all the nodes that are not seen by the color of the clan are grouped in a clan sibling to them and the new vertex is added to it.
   
   (c) Generate a superior clan: when all of the nodes are in the list of visible nodes and are seen by the same color (different to the color of the clan). Then, the current clan and the new vertex will be children of a superior complete clan.
   
   (d) Be one member of the clan, changing the clan type to primitive: when all the nodes are in the list of nonvisible or/and in the list of visible nodes but not by the same color. Thus:
      
      i. The nodes in the list of visible nodes are grouped into clans according they are seen from the new vertex and,
      
      ii. The nodes in the list of nonvisible nodes are splitted.

2. If the current clan is a primitive clan, the new vertex can:
   
   (a) Enter the subclan of one of the nodes inside of the current clan: when the new vertex sees in the same way than one specific node all the remaining nodes. Thus, the new vertex is added to this clan. (See Theorem 20 in Appendix D for details about this step.)
   
   (b) Generate a superior clan: when all of the nodes are in the list of visible nodes and are seen by the same color. Then, the current clan and the new vertex will be children of a superior complete clan.
   
   (c) Be one member of the clan, preserving the type of the clan: when all the nodes are in the list of nonvisible or/and in the list of visible nodes but not by the same color. Thus, we add the new vertex to the clan and the nonvisible nodes are splitted.
The general algorithm is Algorithm 2 and uses Algorithm 1 and Algorithm 3 to find the place of the new node and to get the decomposition. As indicated, the reader can find additional argumentations and also a fully developed example in Appendix D. An open-source implementation of this approach can be found in https://github.com/MelyPic/Gaifman-graphs.

6.1 Related algorithms

Earlier algorithms differ among them and with ours in their quite diverse terminology for the main notions. Besides that, the main differences between the algorithm on [11] and our algorithm are that the algorithm in [11] is applied only on graphs instead of 2-structures and the use of a parallel structure to the decomposition tree. We are quite confident that the literal application of the algorithm exactly as described there would not work on 2-structures. However, the major structure of our approach is quite similar to the one in that reference.

Another algorithm to be considered is [10], also an incremental algorithm. It treats an additional type of clan, namely linear clans, that appear when decomposing 2-structures that are not necessarily reversible; these are akin to directed graphs. As already stated, we do not take them into account because, in our data analysis, all our 2-structures are reversible, since they rely on undirected Gaifman graphs. In future work we will continue our analysis of these algorithms and check out whether some of their ideas for a faster algorithm in big-Oh terms can be implemented efficiently enough for practical usage. Also, our results in the present paper might allow for a novel approach based on a closure miner: if a very fast test for strong closures can be designed, we might be able to run any of the existing, very efficient closed set miners, filter strong closures from there, and set up the modular decomposition by resorting to Theorem 13; of course we would also need to lift that theorem from modules into 2-structures, which we believe to be a far-from-trivial task.

7 Discussion and perspectives

Along this paper, we have shown, first, that the known notion of modular decomposition of a graph can be understood, in a quite natural way from a perspective of data analysis, as a variant of closure space visualization; then, that this process can be applied to transactional datasets via a known logical construction, namely the Gaifman graph; and, also, that both the theoretical connection and the practical applicability of the decomposition process can be generalized to quantitatively enabled variants of Gaifman graphs through a known generalization of modular decompositions, namely the clan decomposition of 2-structures. We have included a number of developments that relate the taxonomy of modules to the corresponding local structure of the closure space and we have described an incremental algorithm, which extends existing ones, that we are currently using in our software tools in order to compute clan decompositions. There is ample room to study improvements to this algorithm and, as mentioned above, the possibilities of using our results to obtain further algorithms.

The avenues for further research are many and wide. First and foremost, additional examples of the practical relevance of this approach are convenient;
**Algorithm 1** Algorithm to get the strong clan decomposition tree when a vertex is added to a specific clan.

**Input:** A vertex $v$ to be added, a clan $C$ and the 2-structure induced by the vertices on $C$ plus $v$.

**Output:** Strong clan decomposition tree.

$L_n$ = list of nonvisible maximal strong clans in the coarsest quotient of $C$ from $v$,

$L_v$ = list of visible maximal strong clans in the coarsest quotient of $C$ from $v$,

$L_c$ = list of visible, as the color of the clan, maximal strong clans in the coarsest quotient of $C$ from $v$.

**if** $C$ is empty or its coarsest quotient has just one maximal strong clan **then**

Add $v$ to the coarsest quotient of $C$.

Assign the type of $C$ as complete.

**else if** the type of $C$ is complete **then**

if $L_c$ is equal to the maximal strong clans in the coarsest quotient of $C$ **then**

Add $v$ to the coarsest quotient of $C$.

else if $|L_v| > 1$ **then**

Let $C_{aux}$ an auxiliary clan.

for all maximal strong clans $M$ not in $L_c$ do

Remove $M$ from the coarsest quotient of $C$.

Add $M$ to the coarsest quotient of $C_{aux}$.

end for

Add $C_{aux}$ to the coarsest quotient of $C$.

Add $v$ to the clan $C_{aux}$. (Algorithm 1 with $v$, $C_{aux}$ and the respective 2-structure as parameters).

else if $L_v$ is equal to the maximal strong clans in the coarsest quotient of $C$ and all of them are seen in the same way from $v$ **then**

Add $C$ and $v$ to the coarsest quotient of a new complete clan.

else

Change the type of clan $C$ to primitive.

Let $C_i$ an empty set for each different equivalence class $i$ (color).

for all maximal strong clans $M$ in the coarsest quotient of $C$ do

if $M$ not in $L_v$ **then**

Add $M$ to the coarsest quotient of $C_i$, being $i$ the way in which $v$ sees $M$.

else

Remove the $M$ from the coarsest quotient of $C$.

Split $M$ from $v$ (Algorithm 3 with $M$ and $v$ as parameters), add each maximal strong clan in the result to the coarsest quotient of $C$.

end if

end for

for all $C_i$ do

if $C_i$ has more than one maximal strong clan **then**

Remove all $M$ in $C_i$ from the coarsest quotient of $C$.

Add $C_i$ to the coarsest quotient of $C$.

end if

end for

else

if there is one maximal strong clan $M$ in the coarsest quotient of $C$ that sees the remaining maximal strong clans in the same way than $v$ does **then**

Add $v$ to the clan $M$ (Algorithm 1 with $v$, $M$ and the respective 2-structure as parameters).

else if $L_v$ is equal to the maximal strong clans in the coarsest quotient of $C$ and all of them are seen in the same way from $v$ **then**

Add $C$ and $v$ to the coarsest quotient of a new complete clan.

else

Add $v$ to the coarsest quotient of $C$.

for all nonvisible nodes $M$ in $L_n$ do

Remove the $M$ from the coarsest quotient of $C$.

Split $M$ from $v$ (Algorithm 3 with $M$ and $v$ as parameters), add each maximal strong clan in the result to the coarsest quotient of $C$.

end for

end if

end if
Algorithm 2 Algorithm to obtain a strong clan decomposition tree from a 2-structure

Input: A 2-structure.

Output: Strong clan decomposition tree.

Current clan decomposition tree = empty clan.

U = Vertices in the 2-structure

for all \( v \in U \) do

Current clan = root clan of the current clan decomposition tree.

Current clan decomposition tree = Apply the Algorithm 1 with \( v \), the current clan and the respective 2-structure induced by them as parameters.

end for

Algorithm 3 Algorithm to split a clan from a node

Input: A clan \( C \) and a node \( n \).

Output: \( C_v \), the set of maximal strong clans below \( C \) visible from \( n \).

Let \( C_v \) an empty set.

if \( C \) is primitive then

for all \( M \) in the coarsest quotient of \( C \) do

if \( M \) is visible from \( n \) then

Add \( M \) to the maximal strong clans in \( C_v \)

else

Split \( M \) from \( n \) (Algorithm 3 with \( M \) and \( n \) as parameters), add each maximal strong clan in the result to the set of maximal strong clans in \( C_v \).

end if

end for

else

Let \( C_i \) an empty set for each different equivalence class \( i \) (color).

for all \( M \) in the coarsest quotient of \( C \) do

if \( M \) is visible from \( n \) then

Add \( M \) to \( C_i \), being \( i \) the way in which \( n \) sees \( M \).

else

Split \( M \) from \( n \) (Algorithm 3 with \( M \) and \( n \) as parameters), add each maximal strong clan in the result to the set of maximal strong clans in \( C_v \).

end if

end for

for all \( C_i \) do

if \( C_i \) has more than one maximal strong clan then

Add \( C_i \) (as a single clan) to the set of maximal strong clans in \( C_v \)

else

Add the unique \( M \) in \( C_i \) to the set of maximal strong clans in \( C_v \)

end if

end for

Return \( C_v \)
so far, we can offer some interesting applications in [1] and [2], but we hope to add further, equally convincing case studies down the road.

Along this practical line, we have found many cases where the obtained graphs and decompositions are somewhat too large or complex to provide intuition through visualization. We have proposed a few simple strategies to encompass complex substructures upon visualization in [1] (our “Others” nodes in the examples in the Appendices, for instance) but a more systematic study of the ways in which visualizations can become helpful is necessary; we believe that the answers will come from some notion of interactive data analysis process: hence our insistence that the major decomposition algorithms to be used should be incremental.

Admittedly, our proposals for enhancing Gaifman graphs with quantitative information can be considered simple or even naïve, and must be subject to further testing on practical cases and to comparison with additional alternatives that could be designed in the future. For instance, one may focus on the set of integers arising as multiplicities of each of all the pairs of vertices in our quantitative Gaifman graph, and consider this set of integers as a single-dimensional dataset itself; then, on it, bring to bear existing unsupervised discretization methods in order to split the edges into their corresponding equivalence classes: the options explored here amount to sticking to the “equal length bins” discretization (either on the original data or after a logarithmic scaling), and quite a few additional options exist.

Also, many other tunings can be applied to the Gaifman graph before applying the decomposition procedure. For instance, we have started in [12] the exploration of the case where the equivalence class of the 2-structure edge \((x, y)\) depends on the distance between \(x\) and \(y\) along a shortest path of the original Gaifman graph. Other similar parameters such as connectivity (number of disjoint paths, that one can relate to Menger’s theorem) could be applied as well.

The multirelational potential is also to be developed. Of course, if we are given a dataset consisting of several tables, it is a simple matter to apply directly our approach as, indeed, Gaifman graphs were defined from the start so as to apply to any relational structure with possibly several tables: a Gaifman edge would join \(x\) and \(y\) whenever they appear together in some tuple of some table. However, initial explorations in [1] suggest that this naïve application will fall short of providing good results, because a crucial notion in multirelational data, namely foreign keys, is being ignored. How to take foreign keys into account at the time of constructing the Gaifman graph, in a way that provides sensible results in practice, is a question that needs careful exploration, both conceptually and in terms of efficiency (e.g. we could denormalize into a single large table the data through the foreign keys, but this looks like an inefficient implementation even if the process turns out to be practically applicable at all). We believe that a generalization of the “shortest path” variant alluded to above could lead to a working approach.

Finally, it is well-known that, in data analysis tasks, categorical concepts (such as implications) benefit from a relaxation (such as partial implications or association rules) allowing for exceptions, whether they come from varied inputs or even from material errors in coding or transmission. Likewise, we could relax through allowing exceptions the notions of module and clan. The concept we would end up with seems to us very close to (a recursive form of) the notion of
“blockmodeling” employed in social network analysis [4]; this is again a large area where, to start with, a clarification endeavor will be effortful but necessary.

References

[1] José Luis Balcázar, Marie Ely Piceno, and Laura Rodríguez-Navas. Decomposition of quantitative gaifman graphs as a data analysis tool. In Wouter Duivesteijn, Arno Siebes, and Antti Ukkonen, editors, *Advances in Intelligent Data Analysis XVII - 17th International Symposium, IDA 2018, 's-Hertogenbosch, The Netherlands, October 24-26, 2018, Proceedings*, volume 11191 of *Lecture Notes in Computer Science*, pages 238–250. Springer, 2018.

[2] José Luis Balcázar, Marie Ely Piceno, and Laura Rodríguez-Navas. Hierarchical visualization of co-occurrence patterns on diagnostic data. In *32nd IEEE International Symposium on Computer-Based Medical Systems, CBMS 2019, Cordoba, Spain, June 5-7, 2019*, pages 168–173, 2019.

[3] Robert J. MacG. Dawson. The ‘unusual episode’ data revisited. *Journal of Statistics Education*, 3(3), 1995.

[4] Patrick Doreian, Vladimir Batagelj, and Anuska Ferligoj. *Generalized Blockmodeling*. Cambridge University Press, 2005.

[5] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[6] Andrzej Ehrenfeucht, Tero Harju, and Grzegorz Rozenberg. *The Theory of 2-Structures - A Framework for Decomposition and Transformation of Graphs*. World Scientific, 1999.

[7] T. Gallai. Transitive orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1):25–66, 1967.

[8] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[9] Frédéric Maffray and Myriam Preissman. A Translation of Gallai’s Paper: ‘Transitive Orientierbare Graphen’, pages 25–66. In Ramírez Alfonsín and Reed [13], 2001.

[10] Ross M. McConnell. An O(n²) incremental algorithm for modular decomposition of graphs and 2-structures. *Algorithmica*, 14(3):229–248, 1995.

[11] John H. Muller and Jeremy P. Spinrad. Incremental modular decomposition. *J. ACM*, 36(1):1–19, 1989.

[12] Marie Ely Piceno and Laura Rodríguez-Navas. A graphical tool for the interpretation of medical data. In *ACM Celebration of Women in Computing: womENcourage*, 2019.

[13] Jorge L. Ramírez Alfonsín and Bruce A. Reed, editors. *Perfect Graphs*. Wiley-Interscience, 2001.
A Proofs of Theorems 10 and 13

Theorem 10: The modules of a graph and the closures defined by its set of modular implications are the same sets.

Proof 1 (Closures are modules.) Let $X$ be a closure, and suppose that there is some $y$ that can distinguish two arbitrary $x_1, x_2 \in X$; one of the modular implications will be $x_1 x_2 \Rightarrow Y$ with $y \in Y \neq \emptyset$. As $X$ is a closure, it must satisfy all the modular implications, and both antecedents are in $X$, thus $y \in X$. Hence, no $y$ outside $X$ may distinguish two elements inside $X$, which is the definition of module.

(Modules are closures.) Let $X$ be a module. It suffices to show that it satisfies all the modular implications: let $x_1 x_2 \Rightarrow Y$ be one of them. If either $x_1 \notin X$ or $x_2 \notin X$, then $X$ satisfies the implication by failing the antecedent. If $x_1, x_2 \in X$ then, since $X$ is a module, no item outside $X$ can distinguish them; but, according to Proposition 8, $Y$ is the set of items that distinguish them, hence $Y \subseteq X$ and $X$ satisfies the modular implication.

Theorem 13: The type of the coarsest quotient graph of a module is primitive if and only if the immediate closure subsets of its corresponding closure in the closure lattice are strong closures and they are more than three.

Proof 2 Let $M$ be a module by Corollary 11 its corresponding closure in the closure lattice is also $M$, let us suppose its coarsest quotient graph collapses the maximal strong modules $M_i$.

($\Rightarrow$) Let $M$ be a primitive module, that is its corresponding coarsest quotient graph is primitive; we have to prove that there is not any union of $M_i$’s closure subset $F$ such that $M_i \subseteq F \subseteq M$. By Proposition 12, there are at least four $M_i$. $F$ must be a union of $M_i$ since, by the premise, $M_i$ are strong closures so $F$ does not overlap any of them. But if $F$ includes at least two $M_i$, by definition of primitive there must be at least one other $M_i$ distinguishing them; thus $F$ could not be a closure.

($\Leftarrow$) We have to prove that if the immediate closures $M_i$ of a strong closure $M$ are strong closures and more than three, then $M$ is a primitive module.

Suppose for two arbitrary $M_i$, let us say $M_1, M_2$, that all the remaining $M_i$’s can not distinguish between them (we can guarantee the existence of at least one remaining $M_i$ by the premise), so $M_1 \cup M_2$ must be a closure contradicting the fact of $M_1$ and $M_2$ are immediate closed subsets of $M$. Thus, for arbitrary $M_1, M_2$ must be at least one $M_i$ that may distinguish between making $M_i$ maximal strong modules and the corresponding coarsest quotient graph of $M$ primitive, thus $M$ is a primitive module.
B Examples of modular decomposition

We start by reproducing a part of the discussion in \cite{1} of the modular decomposition of the famous Titanic dataset \cite{3}. Then we discuss the analysis of the classical Mushroom dataset, and we consider an alternative way to handle datasets via a variant of Gaifman graph where the difference in the edges, instead of being less than 1 joint occurrence versus 1 or more, resorts to a threshold possibly different from 1.

Among several existing incarnations of the Titanic dataset, we employ a very simple one, prepared by Radford Neal\footnote{http://www.cs.toronto.edu/~delve/data/titanic/desc.html} that, for each of the 2201 people on board the well-known ship, records the traveling class (first class, second class, third class, crewmember), age (discretized into adult or child), sex, and whether or not the person survived. In the dataset Mushroom (also known as “Agaricus - Lepiota”), a number of purported attributes of potential mushrooms of these families are expected to be useful to predict whether each of the 8124 observations would correspond to an edible or a poisonous mushroom \cite{5}. (The data was compiled by mushroom experts out of hypothetical mushroom observations, not actual ones.)

B.1 Decomposing Standard Gaifman Graphs

The modular decomposition of the standard Gaifman graph of the Titanic dataset is depicted in Figure 7. The modules for sex and survival are clear and intuitive: as they are different possible values for the same attribute, they never appear together, but happen to have the same set of neighbors\footnote{Vertices with the same set of neighbors are commonly called “twins” in graph theory: “true twins” if they are connected and “false twins” otherwise, like here; we will see often such cases in our examples.}. Actually, as it turns out, for values corresponding to different attributes, every possibility does appear somewhere, so that the top node is a complete, fully connected quotient graph.
Likewise, one might expect a module with the four alternative values of traveling class, namely, 1st, 2nd, 3rd or Crew. However, the closest such module is a complete, fully disconnected graph that only includes actual passenger classes that, of course, never appear together. Instead, the value Crew appears in the parent “ages” module, a small primitive graph where, of course, being an adult is incompatible with being a child, and both are compatible with all the traveling classes; however, Crew co-occurs only with Adult. Thus we are told that, of course, the crew included no children, a fact that we might overlook in a non-systematic analysis. That is: even if the traveling classes and the “Crew” label are employed as values in the same column, the data tells us, through our decomposition procedure, that they have different semantics.

This small primitive graph is actually $P_4$, as Proposition 12 says: its coarsest quotient graph collapses four maximal strong modules, thus they must be the 4-vertex path. We will be seeing $P_4$ often again below.

We describe next some outcomes of analysis on the very well-known dataset Mushroom. Here and also later, with other datasets, we often restrict ourselves to the $n$ most frequent items, for a reasonable value of $n$ that leads to understandable diagrams; equivalently, we impose a threshold on the item supports. Specifically, we restrict ourselves first to items appearing at least 2000 times. Even then, we do not display the complete decomposition tree, because the space is not enough to show it adequately, but discuss some of its modules. First, of course a number of false twins appear such as grass-living versus wood-living mushrooms. Again a $P_4$ case is shown in Figure 8: if a mushroom has foul odor, then it is a poisonous mushroom, as “foul odor” never appears in the same tuple with “edible”. Figure 9 is a little bit more difficult to interpret but we find there, for example, that bruised mushrooms have smooth stalk surfaces, both below and above, but never exhibit silky surfaces either below or above, and not all smooth stalk surfaces, either below or above, are bruised, since both are also connected to the no-bruises vertex. The induced $P_4$ predicted by the theory is indeed there as well (but somewhat more difficult to pinpoint).

Often, we will apply an additional visual simplification. With larger datasets, the usual outcome of the analysis includes mostly large primitive modules of unclear interpretability. The visualization may become, then, uninteresting. In order to construct helpful visualizations, we encompass sometimes large, complicated substructures into single nodes that we label “Others”, as we will do next.
Figure 9: A second non-trivial module in the Mushroom dataset decomposition.

B.2 Decomposing Thresholded Gaifman Graphs

Sometimes we may be interested in keeping track of quantitative information that the standard Gaifman graph lacks. Perhaps the simplest alternative Gaifman graph to do this is the thresholded Gaifman graph (other alternatives are indicated below). In this variant, those pairs of items that appear together less than a determined threshold are considered as “not frequent enough”: they remain disconnected, so that connections represent frequencies of co-occurrence higher than or equal to the threshold.

For the same version of the Titanic dataset described above, assume that we wish to ignore co-occurrences of pairs of items if they occur together less than 1000 times. In the thresholded Gaifman graph, such items will be kept disconnected. The resulting decomposition tree for that Gaifman graph (slightly simplified by grouping as Others the rest of the vertices as announced above) is shown in Figure 10. In a sense, this figure supports the well-known saying: “women and children first”.

Working with the Mushroom dataset with attributes appearing more than 2000 times and considering as not frequent enough those co-occurrences below 1000 times we obtain the decomposition shown in Figure 11, where we see a handful of very frequent items that co-occur with each other. Naturally, we might like to see more detailed information: in this case, one way to decompose the node Others is to give a lower threshold. Giving 800 as threshold value we may see part of the internal behavior of the Others node; the Figure 12 show this part of the decomposition. For this example, we have that there are more than 800 mushrooms but less than 1000 that are edible and do not have odor, and also there is a similar quantity of mushrooms whose ring stalk surfaces, above and below, are silky.
Figure 10: Titanic thresholded Gaifman graph decomposition, at 1000.

Figure 11: Mushroom thresholded Gaifman graph decomposition, at 1000.

Figure 12: Modified module applying the threshold at 800 on the Mushroom dataset.
C Examples of clan decomposition

C.1 Decomposing Linear Gaifman Graphs

In Figure 13 we find the result to apply the clan decomposition algorithm on the Zoo dataset the linear 2-structure with 10 as interval size. The dataset has seventeen attributes and a total of forty two attribute values of about one hundred animal species. In the decomposition we find two clans which tell us that mammals drink milk and birds have feathers.

We also work with the linear Gaifman graph decomposition of the Votes dataset, that contains information about the votes for each of the U.S. House of Representatives Congressmen on the 16 key votes. The Figure 14 is the result to apply the decomposition on the linear Gaifman graph with 100 as interval size of those key votes values that appear more than 100 times. As you can see we find a clan conform by republicans and the negative of adoption of the budget resolution, since they are around 141 republicans of 168 votes against adoption of the budget resolution.

C.2 Decomposing Exponential Gaifman Graphs

In the Figure 15 we find the result to apply the clan decomposition algorithm on the 2-structure determined by the exponential Gaifman graph of Mushroom dataset taking those attribute values that appear more that 2000 times. As we can see most of the attribute values co-occur each other in different ways
excepting the items \textit{gill\_attachment\_free} and \textit{veil\_color\_white} that have the same behavior with the rest of the items. And also we find that they co-occur very often, around 7900 times considering that we have a total of 8000 rows we may say that they appear in the rows almost always, thus most of the mushrooms do not have gills and have a white veil.

The next example is part of [2], where we apply the clan decomposition method on an exponential Gaifman graph of a hospitalization database. The resulting decomposition is show in Figure 16. The hospitalization database has information about hospitalization for the years 2015-2016, each row has information about diagnostics, treatments and conditions of a patient at a fixed date. There is also information related to sex and age of the patient but we do not take this kind of information into account. There are around eighty thousand rows. Whereas the previous datasets were relational, this one is transactional and serves as an example of “market-basket-style” data.

To the 2-structure of our example we take an exponential Gaifman graph whose vertices are those diagnostics and procedures that appear more than 100 times into the dataset, getting seven attribute values:

650 Normal delivery,
632 Missed abortion,
305.1 Tobacco use disorder,
401.9 Unspecified essential hypertension,
272.4 Other and unspecified hyperlipidemia,
81.54.00 Total knee replacement (left) and
81.54.01 Total knee replacement (right).

In the decomposition in the Figure 16 we find two small clans: Total knee replacement (left and right) clan and Hypertension and hyperlipidemia clan. In the Total knee replacement clan the internal nodes are not related, that is there are not cases where both knees are replacement at once, while in the Hypertension and hyperlipidemia clan its internal nodes are so related, we find around eleven thousand cases of co-occurrences of them. Also, a bigger clan provides us with information as of the approximate frequencies with which the small clans show up together, for instance knee replacements are essentially
separate of all the other ailments, and the root clan tells us that normal delivery does not co-occur with the other items.

D Correctness and examples of Algorithms 1, 2 and 3

Let $M$ be the coarsest quotient of a strong complete clan that collapses $M_i$ maximal strong clans and let $x$ be an item, we established the follow theorems to the $M' = M \cup \{x\}$ coarsest quotient:

**Theorem 20** Let $M$ be the coarsest quotient of a strong primitive clan and let $x$ be an item, into the coarsest quotient $M'$ of the set $M \cup \{x\}$, $x$ could be a maximal strong clan by itself or there is just one maximal strong clan $M_i$ in the coarsest quotient $M$, such that $M_i \cup \{x\}$ is a maximal strong clan.

**Proof 3** Let $M_i$ be a maximal strong clan in $M'$ such that $M_i$ contains $x$, if $M_i \setminus \{x\}$ is a clan in $M'$ then $M_i \setminus \{x\}$ is also a clan in $M$. If this $M_i \setminus \{x\}$ is a proper nonempty subset of some other maximal strong clan $M_j$, thus $M_j \cup \{x\}$ is also a clan in $M'$ contradicting the fact that $M_i$ is a maximal strong clan in $M'$. Likewise, $M_i$ cannot properly contain another maximal strong clan $M_j$. Therefore, the maximal strong clan containing $x$ is either $\{x\}$ of the union of $\{x\}$ with just one maximal strong clan of $M$.

As an example of how the algorithms work, we apply them on the graph of Figure 17.

At the begin we add the vertex $a$ to an empty clan decomposition tree, getting a tree with just one singleton clan. In the next step we add the vertex $b$ to it, having as a result a clan with two maximal strong clans in its coarsest
quotient (a complete clan). Both steps are initial cases, they are shown in Figure 18.

When we add $c$ to this clan, we are in the case 1.(b) since $a$ is seen from $c$ by the color of the clan but $b$ is not. Thus, the maximal strong clans that are not seen from the color of the clan are removed to another clan and $c$ is added to it, in this case we add $c$ to a complete clan with just $b$ in its coarsest quotient (the initial case). All the process is shown in Figure 19.

Now, we have in the root a complete clan and the node $d$ is added to it. We find the new node may not see one of the maximal strong clans, center of Figure 20, the case 1.(d). Thus, the type of the clan is change to primitive and the maximal strong clans that are seen from $d$ in the same way will conform a new coarsest quotient while those nodes that are not seen from $d$ are splitted; in this case we do not have any new coarsest quotient and the maximal strong clan conform by $b$ and $c$ is splitted, getting as result the decomposition tree shown in the right of Figure 20.

In the next step we add the vertex $e$ to a primitive clan root, left of Figure 21. We find that there is one maximal strong clan $b$ in its coarsest quotient, that sees all the remaining maximal strong clans in the same way than the vertex to be added, you can see it in the center of Figure 21, the case 2.(a). Thus, the new vertex is added to it, the resulting clan decomposition tree is shown in the right of Figure 21.

When we add $f$ to the primitive clan on the root of the current decomposition, left of Figure 22, we find again that the node can not see one of the maximal strong clans, the clan conform by $b$ and $e$, as we have a primitive clan we are in case 2.(c), and the elements of the clan are splitted, right of Figure 22.

In the next step, we add $g$ to the primitive root of this decomposition tree, and we find that $g$ sees all of the maximal strong clans in the coarsest quotient of the root clan in the same way, center of Figure 23, case 2.(b). Thus, this root clan and the new vertex will be in the coarsest quotient of a new complete clan, right of Figure 23.

When we add $h$ to the complete clan in the root of the current decomposition, left of Figure 24, we find that $h$ sees in the same way than the color of the clan to all the maximal strong clans in the coarsest quotient of the root clan, center of Figure 24, the case 1.(a). Thus, $h$ is added as another maximal strong clan to the coarsest quotient of the complete root clan, right of Figure 24.

Finally, to add $i$ produces a superior clan, since $i$ sees in the same way all the maximal strong clans of the current complete root clan but different than the color of the clan, center of Figure 25, the case 1.(c). Thus, this root clan and the new vertex will be in the coarsest quotient of a new complete clan, the resulting clan decomposition tree is shown in the right of Figure 25.
Figure 17: Graph to apply the decomposition algorithm

Figure 18: Initial cases

Figure 19: Applying 1.(b)
Figure 25: Applying 1.(c)