Graph-FCA: An extension of formal concept analysis to knowledge graphs

Sébastien Ferré, Peggy Cellier

Graph-FCA: an Extension of Formal Concept Analysis to Knowledge Graphs*

Sébastien Ferré[a], Peggy Cellier[a]

[a] *Univ. Rennes, INSA, CNRS, IRISA, Campus de Beaulieu, 35042 Rennes cedex, France*

Abstract

Knowledge graphs offer a versatile knowledge representation, and have been studied under different forms, such as conceptual graphs or RDF graphs in the Semantic Web. A challenge is to discover conceptual structures in those graphs, in the same way as Formal Concept Analysis (FCA) discovers conceptual structures in tables. FCA has been successful for analyzing, mining, learning, and exploring tabular data, and our aim is to help transpose those results to graph-based data. Previous several FCA approaches have already addressed relational data, hence graphs, but with various limits. We propose Graph-FCA as an extension of FCA where a dataset is a hypergraph instead of a binary table. We show that it can be formalized simply by replacing objects by tuples of objects. This leads to the notion of "n-ary concept", whose extent is an n-ary relation of objects, and whose intent is a "projected graph pattern". In this paper, we formally reconstruct the fundamental results of FCA for knowledge graphs. We describe in detail the representation of hypergraphs, and the operations on them, as they are much more complex than the sets of attributes that they extend. We also propose an algorithm based on a notion of "pattern basis" to generate and display n-ary concepts in a more efficient and more compact way. We explore a few use cases, in order to study the feasibility and usefulness of Graph-FCA. We consider two use cases: workflow patterns in cooking recipes and linguistic structures from parse trees. In addition, we report on experiments about quantitative aspects of the approach.

Keywords: Formal Concept Analysis, Knowledge Graph, Semantic Web, Graph Homomorphism

1. Introduction

Since the dawn of artificial intelligence, graphs have been used to represent knowledge as a set of interlinked entities, namely *knowledge graphs*. Notable formalisms of knowledge graphs are semantic networks, conceptual graphs [30, 4],

*This research is supported by ANR project PEGASE (ANR-16-CE23-0011-08).
Email addresses: ferre@irisa.fr (Sébastien Ferré), cellier@irisa.fr (Peggy Cellier)

and more recently RDF graphs in the Semantic Web [15]. In the last ten years, the number and size of knowledge graphs has exploded with the development of the Semantic Web, and its W3C standards (e.g., RDF, SPARQL). Its open side, called Linked Open Data (LOD), is now made of more than 1000 datasets [28], which contain about 70 billions semantic links (called triples). This effort has been joined by Web giants such as the Google Knowledge Graph or Facebook Graph Search.

Knowledge graphs allow for the representation of complex information as a set of entities interlinked with binary, and possibly n-ary, relationships. A challenge is to discover conceptual structures in knowledge graphs, in the same way as Formal Concept Analysis (FCA) discovers conceptual structures in tables [12]. Indeed, Formal Concept Analysis is concerned with the definition of concepts from factual data, and their organization into a generalization ordering, the concept lattice. It serves many purposes such as knowledge discovery, machine learning, information retrieval, or software refactoring [25]. Several proposals in FCA have been made for the exploration and analysis of graph structures. For instance, a power context family [33] is a form of knowledge graph, but there is a distinct concept lattice for each relation arity. Relational Concept Analysis (RCA) [27], an extension of FCA, shows interesting results on relational data, in particular in software engineering [16]. RCA defines concepts that mix unary and binary relationships, but only in tree-shaped patterns. Moreover, the representation of an intent is not self-contained because it contains relational attributes that refer to other concepts, possibly in a circular way. Graphs have also been used as object descriptions (e.g., for molecules) as an application of pattern structures [10, 21], but we do not consider those as knowledge graphs because graph nodes (e.g., individual atoms) are not formal objects. In all those approaches, only unary concepts are defined, i.e., extents are sets of objects, not relations. Concept lattices of relational structures [20] are an approach based on category theory that shares with Graph-FCA the use of n-ary relations, and the definition of *n-ary concepts* with relations as extents. In fact, many of our theoretical foundations, initially presented in our previous paper [6] and in Sections 3,4,5, had already been established independently in [20]. Meanwhile, the main contributions of this paper are twofold: a formal presentation much closer to standard FCA; and more importantly, an algorithm to generate n-ary concepts, along with a compact representation of them. Section 8 provides some details about the relationships between the two formalizations. Outside of the FCA domain, our work can be related to the mining of *closed patterns* in relational, graph, and network data [13]. We here favor a more declarative formalization that emphasizes the duality between concept intents and concept extents. To the best of our knowledge, our notion of *projected graph pattern* has not been studied in the mining context.

Graph-FCA is an extension of FCA for knowledge graphs. The specifity of Graph-FCA is to extract n-ary concepts from a knowledge graph using n-ary relationships. Graph entities play the role of FCA objects, and graph relationships play the role of FCA attributes. The consequence is that the incidence relation relates tuples of objects (with various arities) to attributes, rather than single

objects to attributes. A key novelty compared to FCA is that an extensional representation is not a set of objects, but a set of *tuples of* objects, i.e. a n-ary relation. The particular case of unary relations corresponds to sets of objects. An intensional representation is defined as a *projected graph pattern* (PGP), i.e. as a graph pattern plus a *projection tuple*. The projection tuple can have any arity, and the graph pattern can mix relationships with various arities. Both extensional and intensional representations are self-contained in the sense that they do not refer to each other or to other concepts. A Graph-FCA concept is a pair (extent, intent) where the extent is an object relation, and the intent is a PGP. In practice, it allows to discover n-ary relational concepts. For instance, in a knowledge graph that represents family members with a "parent" relation, the "sibling" binary concept can be discovered, and described by a PGP like "a pair of persons having a common father and a common mother". This significantly extends previous FCA approaches because power context families do not mix arities in concept definitions, and RCA only defines unary concepts based on unary and binary relations. In fact, PGPs are analogous to Datalog (non-recursive) predicate definitions [3], and to SPARQL queries. It suggests that Graph-FCA could be the basis for discovering or learning n-ary predicate definitions, and for querying knowledge graphs. The former is akin to Inductive Logic Programming (ILP) [23], and for the latter, we have already worked out a solution [7].

The formalization of Graph-FCA was published in [6] whereas algorithmic aspects of Graph-FCA and preliminary experiments were presented in [9]. In this paper, we thoroughly discuss theory and present additional results and experiments.

After some technical preliminaries (Section 2), we formalize knowledge graphs as *graph contexts* (Section 3), and then introduce *projected graph patterns (PGP)*, *object relations*, and mappings from one to the other based on *PGP inclusion* and *PGP intersection* (Section 4). From those definitions, we organize PGPs into a bounded lattice, and object relations into a complete lattice, from which we prove the existence of a concept lattice for each concept arity (Section 5). Section 6 presents an algorithm to extract concepts from a graph context, and we propose a compact representation for the extracted concepts. Section 7 shows two use cases to illustrate the capabilities of Graph-FCA, and reports on experiments about the practical complexity of computing Graph-FCA concepts. Section 8 discusses related work. Finally, we conclude and discuss future work on Graph-FCA (Section 9).

2. Preliminaries: Tuples, Projections, and Formal Concept Analysis

In this section, we recall three mathematical concepts required for the paper: tuples, projections, and Formal Concept Analysis (FCA).

Tuples. A *tuple* is noted $\overline{x} = (x_1, \ldots, x_k)$, where $|\overline{x}| = k$ is its *arity*. To avoid confusion with other kinds of indices, $\overline{x}[i]$ can be used as an alternate notation for x_i. The set of all k-tuples over a domain E is noted E^k. The set of all

tuples is defined by $E^* = \bigcup_{k \geq 0} E^k$. There is only one 0-tuple, denoted by (). We use $1..k$ to denote the set of integers from 1 to k. For any unary function ϕ, the notation $\phi(\overline{x})$ denotes the tuple s.t. $\phi(\overline{x})[i] = \phi(\overline{x}[i])$; and for any binary function ψ, the notation $\psi(\overline{x}, \overline{y})$ denotes the tuple s.t. $\psi(\overline{x}, \overline{y})[i] = \psi(\overline{x}[i], \overline{y}[i])$. Given two k-tuples $\overline{x}, \overline{y}$, the notation $\phi^{\overline{y}}_{\overline{x}}$ denotes the function that maps x_i to y_i, for every $i \in 1..k$, and any other value not in $\overline{x}$ to itself. $\phi^{\overline{y}}_{\overline{x}}$ is only well-defined when $\overline{y}$ is *compatible* with $\overline{x}$, i.e. when for all $i \neq j \in 1..k$, $x_i = x_j \Rightarrow y_i = y_j$. For instance, $\phi^{(r,s,t)}_{(u,v,u)}$ is not well-defined because $x_1 = x_3$ but $y_1 \neq y_3$. However, $\phi^{(r,s,r)}_{(u,v,w)}$ is well-defined, u and w both map to r.

Projections. A *projection* $\pi \in \Pi^l_k$ is used to map a k-tuple to a l-tuple, according to the following formula: $\pi(\overline{x})[i] = \overline{x}[\pi(i)]$, i.e. the i-th element of a projected tuple is the element at index $\pi(i)$. It is defined as a function from the indices of the target tuple $1..l$ to the indices of the source tuple $1..k$. For instance, the projection $\pi_1 = \{1 \mapsto 3, 2 \mapsto 1\}$ maps a 3-tuple to a 2-tuple of values where the first element of the output tuple is the 3rd element of the input tuple, and the second element of the output tuple is the 1st element of the input tuple. A more concise representation of π_1 is $(3, 1)$ as a shorthand for the mapping function $\overline{x} \mapsto (\overline{x}[3], \overline{x}[1])$. The identity projection is denoted by $id_k \in \Pi^k_k$, and the composition of two projections is denoted by $\pi_2 \circ \pi_1$, where $(\pi_2 \circ \pi_1)(\overline{x}) = \pi_2(\pi_1(\overline{x}))$, i.e., $(\pi_2 \circ \pi_1)(\overline{x})[i] = \overline{x}[\pi_1(\pi_2(i))]$. A *permutation* is a bijective projection π, and has an inverse projection π^{-1} that is the inverse permutation. Note that the combined applications of an element-wise function ϕ and a projection π commute, i.e. $\pi(\phi(\overline{x})) = \phi(\pi(\overline{x}))$ for every function ϕ, projection $\pi \in \Pi^l_k$, and tuple $\overline{x} \in E^k$. This generalizes to n-ary element-wise functions ψ when applied to n tuples of same arity k: $\pi(\psi(\overline{x}, \overline{y}, \ldots)) = \psi(\pi(\overline{x}), \pi(\overline{y}), \ldots)$.

Formal Concept Analysis. Formal Concept Analysis [11] is about the mathematical derivation of formal concepts from binary data, called a formal context. A *formal context* is a triple $K = (O, A, I)$, where O is a set of *objects*, A is a set of *attributes*, and $I \subseteq O \times A$ is an *incidence relation* between objects and attributes. Two mappings are defined between sets of objects and sets of attributes: *int* and *ext*. For every set of objects $X \subseteq O$, $int(X) := \{a \in A \mid \forall o \in X : (o, a) \in I\}$ returns the *intension* of X, i.e. the set of attributes shared by all objects in X. For every set of attributes $Y \subseteq A$, $ext(Y) := \{o \in O \mid \forall a \in Y : (o, a) \in I\}$ returns the *extension* of Y, i.e. the set of objects sharing all attributes in Y. The pair (ext, int) forms a Galois connection so that $ext \circ int$ and $int \circ ext$ are closure operators, respectively on sets of objects and sets of attributes. A *formal concept* is a pair $C = (X, Y)$ where $X \subseteq O$, $Y \subseteq A$, $int(X) = Y$, and $ext(Y) = X$. X is called the *extent* of the concept, and is a closed set of objects, while Y is called the *intent* of the concept, and is a closed set of attributes. The set of all formal concepts of a formal context forms a complete lattice $\mathcal{C} = \langle \mathcal{C}, \leq, \wedge, \vee, \top, \bot \rangle$, where:

- $(X_1, Y_1) \leq (X_2, Y_2) : \iff X_1 \subseteq X_2 \iff Y_1 \supseteq Y_2$,

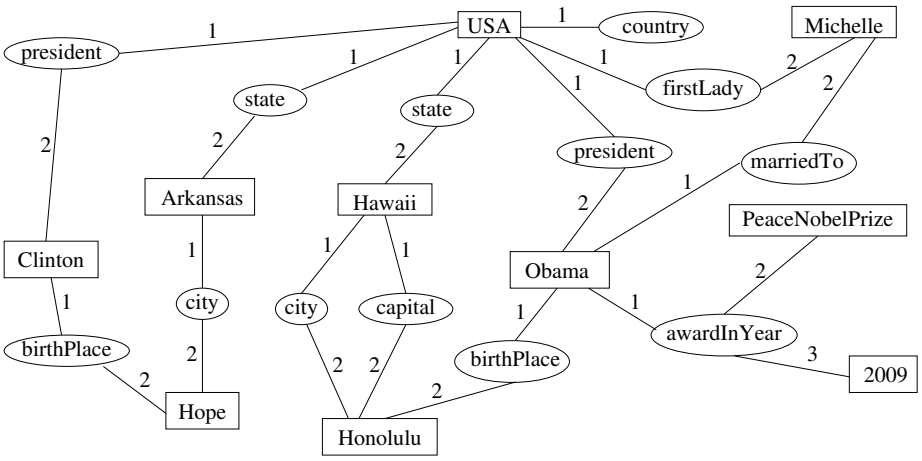

Figure 1: Graphical representation of a graph context about former USA presidents.

- $(X_1, Y_1) \wedge (X_2, Y_2) = (X_1 \cap X_2, int(ext(Y_1 \cup Y_2)))$,
- $(X_1, Y_1) \vee (X_2, Y_2) = (ext(int(X_1 \cup X_2)), Y_1 \cap Y_2)$,
- $\top = (O, int(ext(\emptyset)))$,
- $\bot = (ext(int(\emptyset)), A)$.

3. Knowledge Graphs as Graph-FCA Formal Contexts

In this section we first introduce the notion of formal context in Graph-FCA called *graph context*. We then discuss the differences between named entity nodes and value nodes in this formalization. Finally, we analyze the relations between graph contexts and other knowledge graph representations.

3.1. Graph Context

The first step is to formalize a knowledge graph as a formal context, which we call a *graph context*. The only difference with the classical FCA definition lies in the use of object tuples (O^*) instead of objects (O) in the incidence relation.

Definition 1 (graph context). *A* graph context *is a triple* $K = (O, A, I)$*, where* O *is a set of* objects*,* A *is a set of* attributes*, and* $I \subseteq O^* \times A$ *is an* incidence relation *between object tuples and attributes.*

Figure 1 shows the graphical representation of a small graph context about USA presidents. It uses a graphical notation similar to conceptual graphs, using rectangles for entities, and ellipses for relationships [4]. A graph context is a directed multi-hypergraph, where each node is identified by an object $o \in O$ (e.g.,

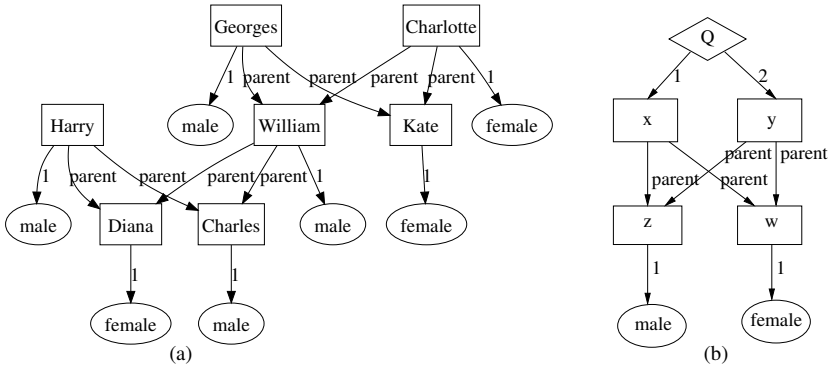

Figure 2: Graphical representation of another graph context about the British royal family.

"Obama", "Hawaii", "2009"), and where each directed hyperedge is an incidence $(\overline{o}, a) \in I$. For the sake of readability, we will use $a(o_1, \ldots, o_k)$ as an alternate notation for incidence $(\overline{o}, a) \in I$. A hyperedge connects a number of objects $\overline{o}$ in a fixed order, and is labelled by an attribute $a \in A$ (e.g., "has president", "is a country"). If $|\overline{o}| = 2$, it is equivalent to a classical labeled edge linking two nodes: e.g., $\mathit{president}(\mathit{USA}, \mathit{Obama})$. If $|\overline{o}| = 1$, it is equivalent to a classical node labelling: e.g., $\mathit{country}(\mathit{USA})$. The advantage of this definition is therefore to treat uniformly classical node labels and edge labels, and to support hyperedges, i.e. n-ary relationships: e.g., $\mathit{awardInYear}(\mathit{Obama}, \mathit{PeaceNobelPrize}, \mathit{2009})$. Hyperedges are directed such that each position in the tuple of objects $\overline{o}$ corresponds to a particular role in the relationship: e.g., $\mathit{state}(\mathit{USA}, \mathit{Hawaii})$ holds while $\mathit{state}(\mathit{Hawaii}, \mathit{USA})$ does not. Note that the numbers that label the edges in the graphical representation correspond to an index in the tuple of objects in the relationship. Nothing forbids to use the same attribute with different arities, but this amounts to have different relationships with the same name[1].

Figure 2 (a) shows another graph context about the British royal family where objects are people, and attributes are about their gender and parenthood. In this figure, the graphical notation of binary edges has been simplified as direct edges between nodes, i.e. the ellipse node for the "parent" relation is replaced by a single edge and the name of the relation is directly put as the label of the edge.

3.2. Representation of Named Entities and Values

In graph contexts, objects only act as junction points between relationships, and can be seen as logical variables. Therefore, two objects that cannot be distinguished based on their relationships to other objects can be considered as the same object, and cannot be used to discriminate other objects. The same situation exists in classical FCA where two objects that are described by the

[1]Similarly to Prolog where predicates are identified by their name *and* arity.

same set of attributes will belong to the same concept, and removing any of them will make no change to the concept lattice. However, objects may carry a significant value (e.g., '2009' in Figure 1) or may refer to a concrete named entity, i.e. everything that can be denoted by a proper name (e.g., 'Obama'). In those cases, it is desirable to put those objects in the set of attributes so that they are taken into account in the description of objects, and hence in the formation of concepts. In classical FCA, this is typically done by adding objects to the set of attributes ($O \subseteq A$), and by adding an incidence from every object to itself ($(o, o) \in I$, for all $o \in O$). The same can be done in Graph-FCA by adding an unary edge $((o), o)$ for each relevant object $o \in O$. By this process, we can distinguish in a knowledge graph the nodes that represent named entities and values, on one hand, and existential variables on the other hand. We can further distinguish between named entities (e.g., people, countries) and values (e.g., numbers, dates). The former have an identity, and it makes sense to represent them by a single object/node. The latter do not have an identity, only a value, so that different occurrences of a same value (e.g., `nbOfChildren '2'` and `nbOfTerms '2'`) should be represented by different objects in order to avoid spurious relatedness (e.g., two persons becoming related because the number of children of one and the number of presidential terms of the other are the same).

3.3. Conversion from/to Other Kinds of Knowledge Graphs

Graph contexts can easily be translated to/from other well-known relation-based representations. For example, a graph context is equivalent to the union of all elements of a Power Context Family [26]. The main limit when translating to other representations is for n-ary relationships with $n \geq 2$. That limit can be addressed by reifying hyperedges as nodes.

For the RDF graphs of the Semantic Web, objects are RDF nodes (URIs, literals, and blank nodes); attributes are URIs (classes, properties, individuals) and literal values (e.g., numbers, dates); and incidences are derived from nodes and triples:

- incidence $((o), c)$ for each triple $(o, \text{rdf:type}, c)$,
- incidence $((o_1, o_2), p)$ for each triple (o_1, p, o_2) where o_2 is not a literal,
- incidence $((o, o_l), p)$ for each triple (o, p, l) where l is a literal and o_l represents its occurrence in the triple,
- incidence $((u), u)$ for each object URI u, and
- incidence $((o_l), l)$ for each occurrence o_l of a literal l.

In relational databases, objects are keys and values, attributes are table names, and incidences are table rows. In Inductive Logic Programming and Datalog, objects are Prolog atoms, attributes are predicates, and incidences are background knowledge facts. In Relational Concept Analysis (RCA) [27], objects are objects, attributes are either context attributes or relation names,

and incidences are either $((o), a)$ when o has attribute a in some context, or $((o_1, o_2), r)$ when (o_1, o_2) are in relation r (see Section 8 for more comparison with RCA).

4. Intensional and Extensional Representations in Graph-FCA

In this section, we introduce the intensional and extensional representations of Graph-FCA. Section 4.1 defines *projected graph patterns* as an extension of the sets of attributes of FCA. Section 4.2 defines *object relations* as an extension of the sets of objects of FCA. Section 4.3 then defines two mappings, one from projected graph patterns to object relations, and another from object relations to projected graph patterns. The two mappings are shown to form a Galois connection in Section 5, and are the basis of Graph-FCA concept lattices. Section 4.4 finally shows that PGPs and object relations are both organized into lattices, one for each arity. The latter subsection is more technical and can be skipped at first reading.

4.1. Projected Graph Patterns as Intensional Representations

A concept intent is an intensional representation that describes everything that a set of objects have in common. As a particular case, the intent of a single object is the description of that object. Depending on the FCA variant, an intensional representation can be a set of attributes, a logical formula [8], or a structure [10]. Therefore, in order to identify Graph-FCA intensional representations, we may start by asking what is an adequate object description in Graph-FCA. An object (e.g., *Obama*) should at least be described by its adjacent hyperedges, i.e. hyperedges $(\overline{o}, a)$ where $\overline{o}$ contains the object (e.g., $president(USA, Obama)$, $awardInYear(Obama, PeaceNobelPrize, 2009)$). Then, if adjacent objects are interlinked (e.g., *USA's president* and *USA's first lady* are *married*), this should also appear in the description. Similarly, the descriptions of adjacent objects (e.g., *Obama being born in Honolulu*) should be included as they indirectly impact what the object is (e.g., *USA having a president born in Hawaii*).

All in all, this implies that the description of an object is the entire knowledge graph, or at least the connected component it belongs to if the knowledge graph is disconnected. Given that knowledge graphs, like the Web, are generally not disconnected, this seems to imply that all objects have the same description! In fact, it is like if all objects were represented by the same knowledge graph, the same world, but each from a different point of view. The different points of view can be interpreted as different phrasings of the same information: e.g., "Obama is the president of USA, and was born in Honolulu", and "Honolulu is the birth place of the president of USA, Obama". Therefore, the description of an object o in a graph context $K = (O, A, I)$ can be defined as the couple (o, I).

It is possible to generalize descriptions from objects to tuples of objects. For example, the description of the 2-tuple (*Honolulu*, *USA*) should contain the properties of each object, and the relationships (direct and indirect) that link

them (e.g., Honolulu "is the capital of a state of" USA, Honolulu "is the birth place of a president of" USA). Similarly to single objects, the description of a tuple of objects $\overline{o}$ can be defined as $(\overline{o}, I)$.

Object descriptions need to be generalized to form concept intents shared by several objects, or several tuples of objects. For instance, we want to describe what *Obama* and *Clinton* have in common, or what (*Honolulu*, *Hawaii*) and (*Houston*, *Texas*) have in common. Those commonalities can be represented by graph patterns, and a natural generalization ordering is graph homomorphisms [14], i.e. mappings from patterns to descriptions that preserve the edge structure. We first define a *graph pattern* as a generalized incidence relation, and then a *projected graph pattern* (PGP) as a generalized description. In the following definitions, we use $\mathcal{V}$ to denote the domain of existential variables, and the set of objects is considered as a subset of those variables ($O \subseteq \mathcal{V}$). We assume a graph context $K = (O, A, I)$ that defines objects and attributes.

Definition 2 (graph pattern). *A* graph pattern $P \subseteq \mathcal{V}^* \times A$ *is a set of directed hyperedges with variables as nodes, and attributes as edge labels.* $V(P)$ *denotes the set of variables occuring in* P*.*

Graph patterns have the same type as incidence relations, only extending the domain of nodes from objects to variables. For instance, the pattern $P_{ex} = \{president(x, y), birthPlace(y, z), Honolulu(z)\}$ describes any situation where "some entity x has as a president another entity y, which has as a birth place a location z called *Honolulu*". In Figure 2 (b), another graph pattern is shown with 4 variables (x, y, z and w) and 3 different attributes (*parent*, *male* and $female$). It describes the situation where "two entities x and y share a father and a mother". Every graph pattern can be seen as a graph context smaller than the original one. We are primarily interested in connected patterns, but disconnected patterns are not excluded.

Definition 3 (projected graph pattern). *A* projected graph pattern *(PGP) is a couple* $Q = (\overline{x}, P)$ *where* P *is a graph pattern, and* $\overline{x} \in \mathcal{V}^*$*, called* projection tuple*, is a tuple of variables.* $|Q| = |\overline{x}|$ *denotes the* arity *of the PGP.* $V(Q)$ *denotes the set of variables occuring in* Q*. We note* $\mathcal{Q}$ *the set of PGPs, and* $\mathcal{Q}_k$ *the subset of k-PGPs, i.e. PGPs having arity* k*. Projections are extended to PGPs:* $\pi(Q) = (\pi(\overline{x}), P)$*.*

A projection tuple can be seen as a tuple of objects abstracted with variables. It also defines a focus on a pattern P. For instance, the PGP $Q_{ex1} = ((x), P_{ex})$ using the above pattern describes "any entity x having a president born in Honolulu"; whereas the PGP $Q_{ex2} = ((y), P_{ex})$ using the above pattern describes "any entity y being a president born in Honolulu". The PGP of Figure 2 (b) $Q = ((x, y), P)$ describes the sibling relation between x and y. Pattern hyperedges can be seen as constraints on variables. A variable that occurs in the projection tuple but not in the pattern is unconstrained, and can take any object as value. A variable that occurs in the pattern but not in the projection tuple is existentially quantified with respect to projected variables. In Q_{ex1},

there must exist an entity y that is the president of x and is born in Honolulu, but which one and how many does not matter. In the same way, in Q, there must exist an entity z that is the father of x and y and an entity w that is their mother but the identities of z and w do not matter.

PGPs can be used to represent the description of objects and object tuples. In those descriptions, the whole incidence of the graph context is used as pattern in order to take into account all relationships between objects, however distant they are from the described objects.

Definition 4 (object (tuple) description). *Given a graph context $K = (O, A, I)$, the* description *of any object $o \in O$ is defined as the PGP $Q_K(o) := ((o), I)$. By extension, the description of any tuple of objects $\overline{o} \in O^*$ is defined as $Q_K(\overline{o}) := (\overline{o}, I)$.*

As said above, PGPs are an extension of the sets of attributes of FCA. For instance, the set of attributes $\{a, c, d\}$ corresponds to the PGP $((x), \{a(x), c(x), d(x)\})$, where x is a variable. Indeed, in FCA, $\{a, c, d\}$ covers all objects that have at least attributes a, c, d. In Graph-FCA, this can be expressed as a node labelled by a, c, d, hence the introduction of the variable x. In general, a set of attributes Y corresponds to the PGP $((x), \{a(x) \mid a \in Y\})$.

PGPs are comparable to non-recursive anonymous predicate definitions, and to SPARQL ASK/SELECT conjunctive queries. For example, the definition of the 'uncle' predicate

$$uncle(x, y) :\Leftrightarrow \exists z.parent(x, z) \wedge brother(z, y)$$

is equivalent to the PGP $((x, y), \{parent(x, z), brother(z, y)\})$. Similarly, the SPARQL SELECT query

```
SELECT ?x ?y WHERE
{ ?x a :Film . ?x :genre :ScienceFiction . ?x :director ?y }
```

is equivalent to the PGP $((x, y), \{Film(x), genre(x, g), ScienceFiction(g), director(x, y)\})$. The SPARQL ASK queries (yes/no questions) correspond to PGPs whose arity is zero ($\overline{x} = ()$).

4.2. Object Relations as Extensional Representations

In our introduction of PGPs as intensional representations, we made a shift from single objects to tuples of objects. That implies that extensional representations in Graph-FCA are sets of tuples of objects. With the constraint that all member tuples have the same arity, we obtain that extensional representations are *object relations*.

Definition 5 (object relation). *An* object relation *is a set $R \subseteq O^k$, for some arity $|R| = k$, of object tuples. We note $\mathcal{R}$ the set of object relations, and $\mathcal{R}_k$ the subset of relations having arity k. Projections are extended to relations: $\pi(R) = \{\pi(\overline{o}) \mid \overline{o} \in R\}$.*

$\mathcal{R}_0$ has only two relations: $\{()\}$ and $\{\}$. It can be seen as the Boolean type, with the two relations meaning "true" and "false" respectively. $\mathcal{R}_1$ has one relation for each set of objects $X \subseteq O$. It therefore corresponds to FCA extensional representations. For instance, the set of objects $\{o_1, o_3, o_4\}$ corresponds to the object relation $\{(o_1), (o_3), (o_4)\}$, simply embedding each object into a 1-tuple. More generally, a set of objects X corresponds to the object relation $\{(o) \mid o \in X\}$. Object relations are comparable to the interpretations of a predicate in classical logic, and to SPARQL query results.

4.3. Extension and Intension in Graph-FCA

We have introduced the intensional representation (PGPs) and the extensional representation (object relations) in Graph-FCA. Now, we extend the notion of "extension of" and "intension of" of FCA (often noted with the prime operator), by defining two mappings: from PGPs to object relations (*extension*), and from object relations to PGPs (*intension*).

4.3.1. Extension: from PGPs to Object Relations

We define a mapping from PGPs to object relations, i.e. from intensional representations to extensional representations. It defines for each PGP its *extension*, i.e. its set of instances in a given graph context. An instance is a tuple of objects whose description $(\overline{o}, I)$ "contains" the PGP modulo a mapping from pattern variables to objects. We first generalize inclusion from sets of attributes to PGPs. PGP inclusion $\subseteq_q$ is based on graph homomorphisms [14]. It is similar to the notion of *subsumption* on queries [5] or rules [23].

Definition 6 (PGP inclusion). *Let $Q_1 = (\overline{x}_1, P_1)$, $Q_2 = (\overline{x}_2, P_2)$ be two PGPs with same arity: $|Q_1| = |Q_2|$. Q_1 is* included in *Q_2, or equivalently Q_2* contains *Q_1, which is denoted by $Q_1 \subseteq_q Q_2$ iff there exists a homomorphism from Q_1 to Q_2, i.e. a mapping $\phi \in V(Q_1) \to V(Q_2)$ s.t. $\phi(\overline{x}_1) = \overline{x}_2$ and $\phi(P_1) \subseteq P_2$.*

$$Q_1 \subseteq_q Q_2 :\Leftrightarrow \exists \phi \in V(Q_1) \to V(Q_2) : \phi(\overline{x}_1) = \overline{x}_2 \wedge \phi(P_1) \subseteq P_2$$

Definition 7 (extension). *Let $K = (O, A, I)$ be a graph context. The* extension *of a k-PGP $Q = (\overline{x}, P)$, denoted by $ext(Q)$, is defined by*

$$ext(Q) := \{\overline{o} \in O^k \mid Q \subseteq_q Q_K(\overline{o})\}$$

The above definitions say that for every occurrence of the pattern P in the graph context ($\phi(P) \subseteq I$), there is an instance of Q ($\phi(\overline{x})$). Conversely, if $\overline{o}$ is an instance of Q, then $Q = (\overline{x}, P)$ must be a generalization of its description $Q_K(\overline{o}) = (\overline{o}, I)$.

For example, in the graph context of Figure 1, the PGP $((x, y), \{USA(u), president(u, x), birthPlace(x, z), state(u, y), city(y, z)\})$ has the following extension: $\{(Obama, Hawaii), (Clinton, Arkansas)\}$. That PGP retrieves the list of USA presidents along with the state of their birth place.

The above definition is compatible with the interpretation of a predicate definition in classical logic: our incidence relation I corresponds to a model, and our homomorphism ϕ corresponds to a variable assignment. It is also compatible with SPARQL query results: our incidence relation I corresponds to a RDF graph, and our homomorphism corresponds to a solution mapping. Finally, it is consistent with classical FCA in the case where only 1-tuples are used, i.e., when $Q = ((x), \{((x), a) \mid a \in Y\})$ for some set of attributes $Y \subseteq A$. Indeed, by casting 1-tuples to their element, and choosing $\phi = \{x \mapsto o\}$, we obtain the classical FCA definition: $ext(Y) = \{o \in O \mid \forall a \in Y : (o, a) \in I\}$.

Note that the mappings ϕ used in PGP inclusion are homomorphisms, and not isomorphisms, because two different variables can map to a single node. This departs from previous work in graph mining and FCA [34, 21] which are based on isomorphisms, but this follows classical logic and SPARQL querying as explained above. The use of homomorphisms has important consequences on the concept lattices, as explained below.

4.3.2. Intension: from Object Relations to PGPs

We define a mapping from object relations to PGPs, i.e. from extensional representations to intensional representations. It defines for each object relation its *intension* as the "PGP intersection" of the description of all tuples $\overline{o} \in R$. PGP intersection $\cap_q$ is defined as a form of graph alignment, where each pair of variables from the two patterns becomes a variable of the intersection pattern. It corresponds to the *categorical product* $\times$ of graphs (see [14], p. 116).

Definition 8 (PGP intersection). *Let ψ be an injective mapping from pairs of variables to variables. The intersection of two k-PGPs $Q_1 = (\overline{x_1}, P_1)$ and $Q_2 = (\overline{x_2}, P_2)$, denoted by $Q_1 \cap_q Q_2$, is defined as $Q = (\overline{x}, P)$, where*

$\overline{x} = \psi(\overline{x_1}, \overline{x_2})$,
$P = P_1 \times P_2 = \{(\psi(\overline{y_1}, \overline{y_2}), a) \mid a \in A, (\overline{y_1}, a) \in P_1, (\overline{y_2}, a) \in P_2, |\overline{y_1}| = |\overline{y_2}|\}$.

Definition 9 (intension). *Let $K = (O, A, I)$ be a graph context. The* intension *of a finite non-empty object relation $R \in \mathcal{R}_k$, denoted by $int(R)$, is defined by*

$$int(R) = \cap_q \{Q_K(\overline{o})\}_{\overline{o} \in R}$$

For example, in the graph context of Figure 1, the object relation $R = \{(Clinton, Hope), (Obama, Honolulu)\}$ has the following intension: $((x, y), \{((x, y), birthPlace), ((USA, x), president), ((USA, z), state), ((z, y), city)\} \cup I)$, where I is the incidence relation of the graph context. The obtained variables are derived from the following alignments: $x = \psi(Clinton, Obama)$, $y = \psi(Hope, Honolulu)$, $z = \psi(Arkansas, Hawaii)$, $USA = \psi(USA, USA)$. Because *USA* is aligned onto itself, the whole graph context is included as a description of USA. The found intension tells us that the two pairs in R relate a USA president to his birthplace, a city in some state of USA. PGP intersection also applies to PGPs that are not object descriptions. For example, the intersection of $Q_1 = ((x_1), \{((x_1, y_1), a), ((x_1, z_1), c), ((y_1, z_1), b)\})$

and $Q_2 = ((x_2), \{((x_2, y_2), a), ((x_2, y_2), c), ((y_2, y_2), b)\}$ is $Q = ((x), \{((x, y), a), ((x, z), c), ((y, z), b)\})$, which is isomorphic to Q_1. Here, both y_1, z_1 are aligned with y_2, hence generating two variables $y = \psi(y_1, y_2)$ and $z = \psi(z_1, y_2)$. Note that Q is not subgraph isomorphic to Q_2. Indeed, under isomorphism, PGP intersection would be the problem of Maximum Common Subgraphs (MCS). A drawback of MCS is that there is generally not a unique solution, so that sets of graph patterns have to be used for intensional representations [21]. Moreover, the MCSs can be less specific. For example, the MCSs of Q_1 and Q_2 patterns are $\{((x, y), a)\}$ and $\{((x, z), c)\}$.

The above definitions of PGP intersection and intension are consistent with classical FCA, where only unary attributes are used. In that case, a single variable can be used, say x, and we can define $\psi(x, x) := x$. Given two sets of FCA attributes, Y_1 and Y_2, the PGP representation of $Y_1 \cap Y_2$ is equal to the PGP intersection of the PGP representations of the two sets: $((x), \{((x), a) \mid a \in Y_1 \cap Y_2\}) = ((x), \{((x), a) \mid a \in Y_1\}) \cap_q ((x), \{((x), a) \mid a \in Y_2)$.

4.4. Lattices of Intensional/Extensional Representations

In the sequel, we define a partial ordering over PGPs for each arity k, and we show that k-PGPs form a bounded lattice. For consistency, we present the lattice of object k-relations, i.e. sets of tuples of size k, based on a set inclusion ordering.

4.4.1. Lattices of k-PGPs

The partial ordering over PGPs should correspond to a generalization ordering over them. Intuitively, a PGP Q_1 is more general than a PGP Q_2 if Q_1 is included in Q_2: $Q_1 \subseteq_q Q_2$ (see Definition 6). Indeed, assume $Q_2 = ((x, y), \{((x), \mathit{country}), ((x, y), \mathit{president})\})$ representing the relationship between countries and their president. Then, $Q_1 = ((x, y), \{((x, y), \mathit{president})\})$ representing the relationship between different kinds of organizations and their president is more general than Q_2 because it relaxes the constraint saying that the organization should be a country. Generalization by constraint relaxation is also found in ILP to define subsumption between learning hypotheses.

Recall that PGP inclusion is defined modulo a homomorphism, which can map two different nodes in Q_1 to a single node in Q_2. The latter corresponds to adding an equality constraint between two entities, which is indeed a specialization. It enables to have

$$((x, y), \{((x, y), \mathit{president}), ((x', y), \mathit{president})\}) \subseteq_q ((x, y), \{((x, y), \mathit{president})\}),$$

by mapping both x, x' to x. Note that the first PGP does not state that y is the president of two organizations, but rather states twice that y is a president, which is equivalent to the second PGP. As the reverse inclusion trivially holds, the two PGPs are equivalent representations of the same thing. We note $Q_1 \equiv_q Q_2$ when $Q_1 \subseteq_q Q_2$ and $Q_2 \subseteq_q Q_1$. PGP inclusion is compatible with inclusion between sets of attributes in FCA. Indeed, as a single variable is involved in

FCA, the homomorphism must be the identity function, and the definition of $Q_1 \subseteq_q Q_2$ boils down to $P_1 \subseteq P_2$.

We prove that $\subseteq_q$ is a preorder, and hence that a partially ordered set is obtained for patterns by considering equivalence classes of PGPs modulo $\equiv_q$.

Lemma 10. *PGP inclusion $\subseteq_q$ is a preorder over PGPs.*

Proof: reflexivity. Given a PGP Q, it suffices to take $\phi = id$ to verify $\phi(P) \subseteq P$ and $\phi(\overline{x}) = \overline{x}$, and hence $Q \subseteq_q Q$.

transitivity. Assume PGPs Q_1, Q_2, Q_3 s.t. $Q_1 \subseteq_q Q_2$ and $Q_2 \subseteq_q Q_3$. Hence, there exists two homomorphisms ϕ_1, ϕ_2 s.t. $\phi_1(P_1) \subseteq P_2$, $\phi_2(P_2) \subseteq P_3$, $\phi_1(\overline{x}_1) = \overline{x}_2$, and $\phi_2(\overline{x}_2) = \overline{x}_3$. Then, it suffices to take $\phi = \phi_2 \circ \phi_1$ to verify $\phi(P_1) = \phi_2(\phi_1(P_1)) \subseteq \phi_2(P_2) \subseteq P_3$, and also $\phi(\overline{x}_1) = \phi_2(\phi_1(\overline{x}_1)) = \phi_2(\overline{x}_2) = \overline{x}_3$. Hence $Q_1 \subseteq_q Q_3$. ■

Before showing that the pre-ordering over k-PGPs forms a bounded lattice modulo $\equiv_q$, for every arity k, we first need to define *PGP union* to act as a supremum. To that purpose, we also need to define the equality constraints that are introduced by duplicates in a projection tuple $\overline{x}$.

Definition 11 (equality constraints). *Let $\overline{x}$ be a projection tuple of arity k. Its set of* equality constraints *is defined by*

$$Eq(\overline{x}) = \{(i, j) \mid i < j \in 1..k, x_i = x_j\}.$$

A set of equality constraints generates an equivalence relation between projection tuple indices, and is confused with it in the following.

For example, the set of equality constraints of the projection tuple (x, y, x) is $\{(1, 3)\}$, and generates two equivalence classes: $\{1, 3\}$, and $\{2\}$. In a PGP $(\overline{x}, P)$, equality constraints come in addition to hyperedge constraints from the graph pattern P. By allowing such duplicates, we allow a single entity to play different roles, like when searching a common PGP between the pairs $(Canberra, Sydney)$ and $(Paris, Paris)$: e.g. $((x, y), \{Country(z), capital(z, x), largestCity(z, y)\})$ that means "a pair of entities that are respectively the capital and the largest city of a same country".

Definition 12 (PGP union). *Let $Q_1 = (\overline{x}_1, P_1)$ and $Q_2 = (\overline{x}_2, P_2)$ be two k-PGPs, using disjoint sets of variables ($V(Q_1) \cap V(Q_2) = \emptyset$). The* PGP union *$Q_1 \cup_q Q_2$ is the k-PGP $Q = (\overline{x}, P)$ verifying*

- *$Eq(\overline{x}) = Eq(\overline{x}_1) \cup Eq(\overline{x}_2)$,*
- *$\overline{x}$ is formed by choosing a distinct variable for each equivalence class of $Eq(\overline{x})$, and by defining each x_i ($i \in 1..k$) as the variable of the equivalence class that contains index i, and*
- *$P = \phi^{\overline{x}}_{\overline{x}_1}(P_1) \cup \phi^{\overline{x}}_{\overline{x}_2}(P_2)$.*

The assumption that PGPs do not share any variable is there to avoid variable capture. It entails no loss of generality because variables can be renamed freely. PGP union corresponds to add both equality and edge constraints of both PGPs, and is logically equivalent to a conjunction. The projection tuple that satisfies all equality constraints is used as a projection tuple of the PGP union, and also to merge variables ($\phi^{\overline{x}}_{\overline{x}_i}$) from the two projection tuples $\overline{x}_1$ and $\overline{x}_2$. For instance for 3-PGPs and the set of equality constraints $Eq(\overline{x}) = \{(1,3)\}$, a valid projection tuple is $\overline{x} = (v_1, v_2, v_1)$, where the 1st and 3rd component use the same variable. Finally, PGP union is compatible with the union of sets of attributes in FCA.

We first prove two lemmas stating that $\cup_q$ and $\cap_q$ are respectively the supremum and infimum of k-PGPs, with query inclusion $\subseteq_q$ as the partial ordering.

Lemma 13. *Let Q_1, Q_2 be two PGPs. Their PGP union $Q_1 \cup_q Q_2$ is their* supremum*.*

Proof: Let $Q = Q_1 \cup_q Q_2$. To prove that $Q = (\overline{x}, P)$ is an upper bound, it suffices to prove that it contains both Q_1 and Q_2. To prove $Q_1 \subseteq_q Q$, it suffices to choose $\phi_1 = \phi^{\overline{x}}_{\overline{x}_1}$, which is well-defined because $Eq(\overline{x}_1) \subseteq Eq(\overline{x})$, and to prove $\phi_1(\overline{x}_1) = \overline{x}$ and $\phi_1(P_1) \subseteq P$. This is easily obtained from the definition of Q. The proof of $Q_2 \subseteq_q Q$ is identical with $\phi_2 = \phi^{\overline{x}}_{\overline{x}_2}$.

To prove that Q is the *least* upper bound (the supremum), we have to prove that every PGP $Q' = (\overline{x}', P')$ that contains both Q_1 (via ϕ_1) and Q_2 (via ϕ_2) also contains Q. From hypotheses $\phi_1(\overline{x}_1) = \overline{x}'$ and $\phi_2(\overline{x}_2) = \overline{x}'$, we obtain that $Eq(\overline{x}_1) \subseteq Eq(\overline{x}')$ and $Eq(\overline{x}_2) \subseteq Eq(\overline{x}')$. Then, we have $Eq(\overline{x}) = Eq(\overline{x}_1) \cup Eq(\overline{x}_2) \subseteq Eq(\overline{x}')$, and hence that $\phi^{\overline{x}'}_{\overline{x}}$ is well-defined. Let $\phi = \phi_1 \cup \phi_2 \cup \phi^{\overline{x}'}_{\overline{x}}$, which is well-defined because the three mappings have disjoint domains (or can be made so by variable renaming in Q_1, Q_2, Q). We can then prove that (a) $\phi(\overline{x}) = \overline{x}'$ and (b) $\phi(P) \subseteq P'$ by observing that $\phi(\phi^{\overline{x}}_{\overline{x}_1}(P_1)) = \phi_1(P_1)$ and similarly for P_2. Hence $Q \subseteq_q Q'$. ■

Lemma 14. *Let Q_1, Q_2 be two PGPs. Their PGP intersection $Q_1 \cap_q Q_2$ is their* infimum*.*

Proof:

To prove that $Q_1 \cap_q Q_2$ is a lower bound, it suffices to prove that Q is included in both Q_1 and Q_2. To prove $Q \subseteq_q Q_1$, it suffices to choose the mapping $\phi_1(x) = (\psi^{-1}(x))[1]$ (recall that ψ is an injective mapping from 2-tuples of variables to variables), and to prove that $\phi_1(\overline{x}) = \overline{x}_1$ and $\phi_1(P) \subseteq P_1$. This is easily obtained from the definition of Q. The proof of $Q \subseteq_q Q_2$ is identical with $\phi_2(x) = (\psi^{-1}(x))[2]$.

To prove that $Q_1 \cap_q Q_2$ is the *greatest* lower bound (the infimum), we have to prove that every PGP Q' that is included in both Q_1 (via ϕ_1) and Q_2 (via ϕ_2) is also included in Q. To that purpose, it suffices to choose $\phi(x') = \psi(\phi_1(x'), \phi_2(x'))$, and to prove that $\phi(\overline{x}') = \overline{x}$ and $\phi(P') \subseteq P$. This can be obtained from the definition of Q, and from the hypotheses $\phi_1(\overline{x}') = \overline{x}_1$, $\phi_1(P') \subseteq P_1$, $\phi_2(\overline{x}') = \overline{x}_2$, and $\phi_2(P') \subseteq P_2$. ■

We now define the empty PGP and the full PGP, and prove that they are respectively the bottom and top of the partial ordering over PGPs.

Definition 15 (empty and full PGP). *For every arity k, the* empty k-PGP *is defined as $\emptyset_q := (\overline{x}, \emptyset)$ such that $|\overline{x}| = k$ and $Eq(\overline{x}) = \emptyset$; and the* full k-PGP *is defined as $\Omega_q := ((x, \ldots, x), \{((x, \ldots, x), a) \mid a \in A\})$.*

The empty PGP puts no constraint, i.e. has no hyperedge, and no duplicate variables in the projection tuple. The full PGP is fully contrained, i.e. its pattern has an hyperedge for all attributes and arities, and it uses the same variable in all positions.

Lemma 16. *The empty k-PGP is the bottom k-PGP, i.e. $\emptyset_q \subseteq_q Q$ for all PGP $Q \in \mathcal{Q}_k$.*

Proof: Let $\emptyset_q = (\overline{x}_1, \emptyset)$ and $Q = (\overline{x}_2, P)$. It suffices to choose $\phi = \phi^{\overline{x}_2}_{\overline{x}_1}$ in the definition of PGP inclusion. ϕ is well-defined because $Eq(\overline{x}_1) = \emptyset$ according to the definition of the empty PGP, and hence $\overline{x_1}[i] = \overline{x_1}[j] \Rightarrow i = j \Rightarrow \overline{x_2}[i] = \overline{x_2}[j]$. ■

Lemma 17. *The full k-PGP is the top k-PGP, i.e. $Q \subseteq_q \Omega_q$ for all PGP $Q \in \mathcal{Q}_k$.*

Proof: Let x be the one variable used in Ω_q. It suffices to choose ϕ s.t. $\phi(y) = x$ for every variable $y \in V(Q)$, and use it in the definition of $\subseteq_q$. ■

From above definitions and lemmas, the following theorem immediately follows.

Theorem 18. *For every arity k, the algebraic structure $(\mathcal{Q}_k, \subseteq_q, \cap_q, \cup_q, \Omega_q, \emptyset_q)$ forms a bounded lattice, modulo $\equiv_q$.*

4.4.2. Complete Lattices of Object k-Relations

The partial ordering over object relations should be consistent with the partial ordering on PGPs if we want to obtain concept lattices. Therefore, it should correspond to a form of generalization at the extensional level. A PGP can be made more general by relaxing constraints, which entails a larger extension. As object relations are sets of object tuples, we simply use set inclusion to partially order them. Given that $\mathcal{R}_k$ is the powerset of O^k, the poset $(\mathcal{R}_k, \subseteq, \cap, \cup, O^k, \emptyset)$ is a complete lattice, with set intersection $\cap$ as infimum, set union $\cup$ as supremum, full relation O^k as top, and empty relation $\emptyset$ as bottom.

5. A Family of Graph Concept Lattices

In the previous section, we introduce for an arity k two partial orderings, one over the k-PGPs and one over the object k-relations. In this section, based on those two orderings, we define the notion of graph concept for a given arity k and the associated graph concept lattice. Then we discuss the relationships between concepts of lattices with different arities. The latter subsection (Section 5.2) can be skipped at first reading.

5.1. Lattices of Graph k-Concepts

In order to prove the existence of a concept lattice for each arity, it suffices to prove that the two mappings between extensional and intensional representations form a Galois connection.

Theorem 19 (Galois connection). *Let $K = (O, A, I)$ be a graph context. For every arity k, the pair of mappings (ext, int) forms a Galois connection between $(\mathcal{R}_k, \subseteq)$ and $(\mathcal{Q}_k, \subseteq_q)$, i.e. for every object relation $R \in \mathcal{R}_k$ and PGP $Q \in \mathcal{Q}_k$,*

$$R \subseteq ext(Q) \iff Q \subseteq_q int(R)$$

Proof: $R \subseteq ext(Q) \iff \forall \overline{o} \in R : \overline{o} \in ext(Q)$
$\iff \forall \overline{o} \in R : Q \subseteq_q Q_K(\overline{o})$ (Definition 7)
$\iff Q \subseteq_q \cap_q \{Q_K(\overline{o})\}_{\overline{o} \in R}$ (Lemma 14)
$\iff Q \subseteq_q int(R)$ (Definition 9) ■

Corollary 20. *From (ext, int) being a Galois connection and from $(\mathcal{R}_k, \subseteq, \cap, \cup, O^k, \emptyset)$ and $(\mathcal{Q}_k, \subseteq_q, \cap_q, \cup_q, \Omega_q, \emptyset_q)$ being lattices, we have the following propositions for every relations $R, R_1, R_2 \in \mathcal{R}_k$, and every PGP $Q, Q_1, Q_2 \in \mathcal{Q}_k$, for any arity k:*

(1a) $Q_1 \subseteq_q Q_2 \Rightarrow ext(Q_1) \supseteq ext(Q_2)$	(1b) $R_1 \subseteq R_2 \Rightarrow int(R_1) \supseteq_q int(R_2)$
(2a) $Q \subseteq_q int(ext(Q))$	(2b) $R \subseteq ext(int(R))$
(3a) $int(R) \equiv_q int(ext(int(R)))$	(3b) $ext(Q) = ext(int(ext(Q)))$
(4a) $int(R_1 \cup R_2) \equiv_q int(R_1) \cap_q int(R_2)$	(4b) $ext(Q_1 \cup_q Q_2) = ext(Q_1) \cap ext(Q_2)$
(5a) $int(\emptyset) \equiv_q \Omega_q$	(5b) $ext(\emptyset_q) = O^k$

From the Galois connection, *graph concepts* can be defined and organized into concept lattices, like in classical FCA, with one concept lattice for each arity k.

Definition 21 (graph concept). *Let $K = (O, A, I)$ be a graph context. A* graph concept *of K is a pair (R, Q), made of an object relation (the* extent*) and a (equivalence class of) PGP (the* intent*), such that $R = ext(Q)$ and $Q \equiv_q int(R)$. The* arity *of a graph concept is the arity of its extent and intent, which have to be equal.*

Theorem 22 (graph concept lattices). *The set of graph k-concepts $\mathcal{C}_k$, partially ordered by $\leq$, which is defined by $(R_1, Q_1) \leq (R_2, Q_2) : \iff R_1 \subseteq R_2 \iff Q_2 \subseteq_q Q_1$, forms a bounded lattice $(\mathcal{C}_k, \leq, \wedge, \vee, \top, \bot)$ where:*

1. $(R_1, Q_1) \wedge (R_2, Q_2) = (R_1 \cap R_2, int(ext(Q_1 \cup_q Q_2))$,
2. $(R_1, Q_1) \vee (R_2, Q_2) = (ext(int(R_1 \cup R_2)), Q_1 \cap_q Q_2)$,
3. $\top = (O^k, int(ext(\emptyset_q)))$,
4. $\bot = (ext(int(\emptyset)), \Omega_q)$.

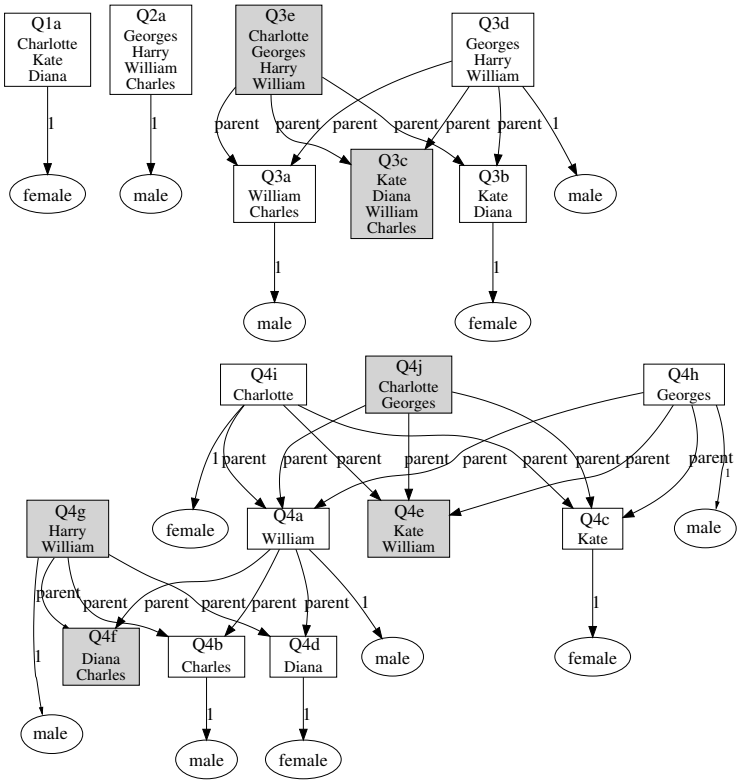

Figure 3: Compact representation of graph concepts about the British royal family.

In the example context of Figure 1, the most interesting graph concept has as extent the set of triples (president, city, state): $\{(Obama, Honolulu, Hawaii), (Clinton, Hope, Arkansas)\}$. Its intent is the PGP $((p, c, s), \{((USA, p), president), ((p, c), birthPlace), ((USA, s), state), ((s, c), city)\} \cup I)$. Other graph concepts are either projections of it (see Section 5.2), concepts with singleton extents (having one tuple), the top concepts (having all tuples), and the bottom concepts (having no tuple).

Figure 3 displays a compact representation of the graph concepts about the British royal family (graph context in Figure 2 (a)). In total, there are 19 unary concepts (top and bottom concepts are not represented in Figure 3). Each node x (in a square box) identifies a unary concept (e.g., Q3e) along with its extent (here, $\{Charlotte, Georges, Harry, William\}$). The concept intent is the PGP $((x), P)$, where P is the subgraph containing node x and all white nodes (called the *pattern core*, i.e. the nodes that appear in all represented concepts). Concept Q3e is concept "child", which, in the graph context, always has a known father and mother, which always have a son. Note that the son maybe either the child's brother or the child himself because homomorphisms need not be injective. Concept Q1a is concept "female person". Concept Q4i

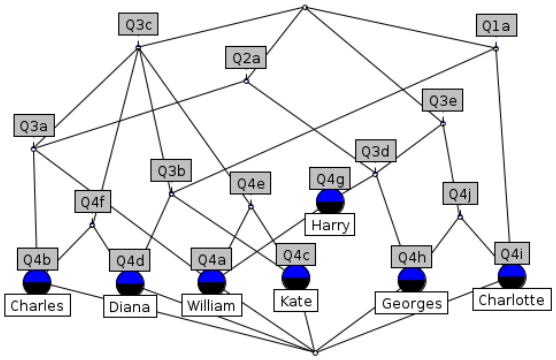

Figure 4: Lattice of the unary concepts about the British royal family.

uniquely characterizes Charlotte in the graph context as being female, having parents, paternal grand-parents, and a brother. Note that there is no concept that uniquely characterizes Harry because his description is a subset of William's description; hence concept Q4g gathering William and Harry. Figure 4 shows the lattice structure of all 19 unary concepts. Binary concept intents are obtained by picking two nodes to form a projection tuple, and by taking the union of the two patterns. For example, concept (Q3a,Q3b) is concept "couple sharing a son". Concept (Q3e,Q3d) defines the relationship "having a brother", if we exclude self-relationships. Ternary and other n-ary concepts are formed likewise. Unlike for unary concepts, the extent of n-ary concepts is not directly accessible on the compact representation. It is a subset of the set product of unary extents.

5.2. Projections Between Concept Lattices

In the previous section, we have shown the existence of a family of graph concept lattices, one for each arity $k \geq 0$. This is analogous to previous work with power context families [33], with the important difference that each k-concept has as an intent a PGP that may combine relationships of different arities. As FCA lattices are generally used as search spaces for knowledge discovery, it is useful to relate concepts from different lattices, i.e. having different arities. To that purpose, we use *projections*, a fundamental operation in relational algebra (see Section 2 for definitions and notations). A projection enables to permute, duplicate, and remove *columns* in relations and PGPs. Our projections differ from those of pattern structures, which are used to simplify graph patterns [10].

We first demonstrate that the set of all concept extents is closed by projection because the projection of the extension of a PGP is the extension of the projection of the PGP.

Lemma 23. *For all $Q \in \mathcal{Q}_k$, and $\pi \in \Pi^l_k$, we have: $\pi(ext(Q)) = ext(\pi(Q))$.*

Proof: $\pi(ext(Q)) = \pi(\{\overline{o} \mid \exists \phi : \phi(\overline{x}) = \overline{o} \wedge \phi(P) \subseteq I\})$
$= \{\pi(\overline{o}) \mid \exists \phi : \phi(\overline{x}) = \overline{o} \wedge \phi(P) \subseteq I\} = \{\overline{o}' \mid \exists \phi : \pi(\phi(\overline{x})) = \overline{o}' \wedge \phi(P) \subseteq I\}$

$= \{\overline{o}' \mid \exists \phi : \phi(\pi(\overline{x})) = \overline{o}' \wedge \phi(P) \subseteq I\} = ext((\pi(\overline{x}), P)) = ext(\pi(Q))$ ■

Lemma 24. *For all $R \in \mathcal{R}_k$, and $\pi \in \Pi_k^l$, we have: $\pi(int(R)) = int(\pi(R))$.*

Proof: Let $R = \{\overline{o}_1, \ldots, \overline{o}_n\}$. $\pi(int(R)) = \pi(\bigcap_{\overline{o} \in R}(\overline{o}, I)$
$= \pi((\psi(\overline{o}_1, \ldots, \overline{o}_n), I \times \ldots \times I)) = (\pi(\psi(\overline{o}_1, \ldots, \overline{o}_n)), I \times \ldots \times I)$
$= (\psi(\pi(\overline{o}_1), \ldots, \pi(\overline{o}_n)), I \times \ldots \times I) = \bigcap_{\overline{o} \in R}(\pi(\overline{o}), I) = \bigcap_{\overline{o} \in \pi(R)}(\overline{o}, I) = int(\pi(R))$ ■

Theorem 25. *Let $\pi \in \Pi_k^l$ be a projection. For every k-concept (R, Q), $(\pi(R), \pi(Q))$ is a l-concept. The latter is called the π-projection of concept (R, Q), denoted by $\pi(R, Q)$.*

Proof: From Lemma 23, we have $\pi(R) = \pi(ext(Q)) = ext(\pi(Q))$, so that $\pi(R)$ is a l-concept extent. From Lemma 24, we have $\pi(Q) = \pi(int(R)) = int(\pi(R))$, so that $\pi(Q)$ is a l-concept intent. ■

For example, let $P = \{((x), country), ((x, y), president)\}$ be a graph pattern relating a country to its president. The PGP $Q = ((x, y), P)$ returning pairs (country,president) can be projected to the PGP $((y), P)$ returning all presidents of a country, or to the PGP $((x), P)$ returning all countries having a president. In the particular case where $k = l$, the two concepts belong to the same lattice. For example, the PGP $((y, x), P)$ is a permutation of Q. Therefore, there may be up to $k!$ permutations of a single k-concept in the same concept lattice. As those permutations are equivalent from the point of view of knowledge discovery, a concept lattice could in principle be made smaller by retaining only one of the permutations. Note that a concept can sometimes be equal to some of its permutations. For example, The query $((x, y), \{((x, z), parent), ((y, z), parent)\})$, which defines the sibling relationship, has the same extension as its permutation (y, x). This equality comes from a symmetry in the PGP.

The existence of a projection between two concepts defines a pre-ordering $\leq_\pi$ on the set of all concepts $\mathcal{C} = \bigcup_{k>0} \mathcal{C}_k$. Indeed, it satisfies transitivity (by composing projections), reflexivity (by using the identity projection), but not antisymmetry (consider a permutation and its inverse). Two concepts are then equivalent ($=_\pi$) if they are a permutation one of the other. The example concept from Section 5, which contains triples (president, city, state), has 6 distinct permutations, and 2×3 projections of arity 2, and 3 projections of arity 1. The fact that all those projections have the same number of instances as the example concept reveals functional dependencies from any column to the others. For instance, the president determines the city and state. The functional dependency from state to president would be violated if two presidents in the concept extent were associated to the same state. This example suggests that the partial ordering $\leq_\pi$ can support the discovery of functional dependencies, and may generalize previous work on multi-valued contexts [1].

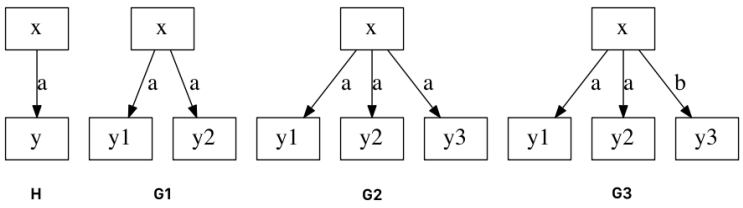

Figure 5: Graphs **H** and **G1** are retracts of graph **G2** but not of **G3**. Graph **H** is the core of graphs **G1** and **G2**. Hence, graphs **H**, **G1**, and **G2** are equivalent.

6. Computation and Presentation of Graph Concepts

The computation of graph concepts is challenging because of the complexity of computing with graphs. The fact that PGP inclusion is based on graph homomorphism rather than on subgraph isomorphism like in most other work on graph patterns [21, 34] is both an advantage and a drawback. The advantage is that every intersection of two PGPs is a PGP, so that it is not necessary to reason on sets of PGPs for computing concept intents. The drawback is that an intersection $Q_1 \sqcap_q Q_2$ may be larger than both Q_1 and Q_2 when an object has several edges with the same attribute: e.g., a parent of several children. Another difficulty is that it is more difficult to get canonical representations of PGPs compared to FCA sets of attributes. Indeed, PGPs may be equivalent although their graph patterns are not isomorphic: e.g., graphs **H**, **G1**, and **G2** in Figure 5. In the following, we first describe the naive version of bottom-up generation of concepts up to some arity k (Section 6.1). We then sketch an algorithm to factorize computations, and to generate a *pattern basis* as a compact representation of the set of concepts (Section 6.2). Finally, the graphical display produced by the implementation of the algorithm is described in Section 6.3.

6.1. Naive Generation of Concepts

The adopted strategy is to generate concept intents in a bottom-up fashion, based on the second half of the Galois connection: $R' := \sqcap_q \{Q_K(\overline{o})\}_{\overline{o} \in R}$. The principle for each arity k is to start from the set of all descriptions of k-tuples of objects $\{Q_K(\overline{o}) \mid \overline{o} \in O^k\}$, and to close it by application of PGP intersection $\sqcap_q$. It is easy to see why this naive generation is far from optimal. For example, in the royal family context, the generation of the unary concepts from the 7 objects already generates $C_7^2 = 21$ PGP intersections, and hence 21 alignements of the incidence relation on itself. It gets exponentially worse with concept arity increasing.

In order to detect when an intent has already been generated, each PGP $Q = (\overline{x}, P)$ (including object tuple descriptions) must be given the canonical representation of its equivalence class. That canonical representation is computed in two steps. First, the minimal *retract* R of the graph pattern P that contains

the projected nodes $\overline{x}$ must be found. Second, the nodes of R are numbered in a canonical way assuming a fixed ordering of attributes. Roughly, a *retract* of a graph G is a subgraph of G that conveys the same information (for details see [14], p. 112). In Figure 5, graphs **H** and **G1** are retracts of graph **G2** but not of **G3**. Indeed, stating several times that x is in a a-relation to something adds nothing to stating it once because variables $y1$, $y2$, and $y3$ can map to the same object in the graph context. On the contrary, **G3** states that x is both in a a-relation and a b-relation, and cannot retract to **H**: edge $((x, y3), b)$ cannot fold onto edge $((x, y2), a)$. A *core* is a minimal retract. In Figure 5, graph **H** is the core of graphs **G1** and **G2**. If there are several cores, any of them can be chosen as they are isomorphic.

6.2. Efficient Generation of a Pattern Basis

We sketch another algorithm for a more efficient generation and a more compact representation of concepts (Algorithm 1). In this section, we focus on *qualitative* performance, and in Section 7, we report and discuss experiments about *quantitative* performance. By *qualitative* performance we mean the orderly generation of concepts avoiding as much as possible duplication both in computation and in presentation of results. First, we choose to only generate concepts whose intent is a *connected PGP*, i.e. a PGP $Q = (\overline{x}, P)$ whose graph pattern P is connected, and whose projection tuple $\overline{x}$ contains only nodes from the graph pattern. Indeed, a disconnected PGP can be decomposed into a set of connected PGPs and empty PGPs. Second, Algorithm 1 computes new graph patterns by *alignment* (categorical product $\times$) of graph patterns without taking into account projection tuples (**Line 6**, details in Section 6.2.1). Concept intents are derived from those graph patterns by combining them with projection tuples (up to some maximum size K) taken from their nodes (**Lines 11-15**, details in Section 6.2.2).

6.2.1. Generation of Graph Patterns

Instead of generating PGPs directly by PGP intersection, we first generate alignments (categorical products) of graph patterns ($P_1 \times P_2$), ignoring at this stage projection tuples (**Line 6**). Thus, the incidence relation is aligned onto itself only once ($I \times I$, when $P = I$). The graph product may have several connected components, and according to our above choice we consider each connected component as an independent pattern P_p. Figure 6 shows the first computed categorical product in the running example. The incidence relation of the graph context is aligned with itself, defining a graph over pairs of people of the British royal family. It is made of 3 non-trivial connected components (P1, P2, P3), where trivial connected components are those made of a single node. P1 includes the incidence relation, and is equivalent to it (up to homomorphism); and P3 is isomorphic to P2. We therefore only consider P2 in the subsequent explanations.

Optimization in case of symmetries. In case of 1-n or n-n relationships (e.g., persons having several children), the generated graph patterns P_p often exhibit

Algorithm 1 Generation of k-concepts for $k \in 1..K$

Require: $K = (O, A, I)$ is a graph context
Ensure: $Concepts$ is the concept set
Ensure: $PatternBasis$ is a basis of patterns grouped by their core

$PatternBasis \leftarrow \emptyset$
$Concepts \leftarrow \emptyset$
$Patterns \leftarrow \{I\}$ // a queue of patterns to process
for all $P \in Patterns$ **do**
  $Patterns \leftarrow Patterns \setminus \{P\}$ // take P off the queue
  $NewPatterns \leftarrow ConnectedComponents(P \times I)$
  $NewPatterns \leftarrow \{removeDuplicateNodes(P_p) \mid P_p \in NewPatterns\}$
  for all $P_a \in NewPatterns$ (not yet seen modulo isomorphism) **do**
    $P_c \leftarrow$ the core of P_a
    $PatternBasis \leftarrow PatternBasis \cup \{P_a @ P_c\}$
    // P_a is added and grouped with other patterns sharing core P_c
    for all $k \in 1..K$, $\overline{x} \in V(P_a)^k$ /* k-tuples over P_a-nodes */ **do**
      $P_r \leftarrow$ the smallest retract of P_a containing $\overline{x}$
      $Q \leftarrow (\overline{x}, P_r), R \leftarrow Q'$ // concept intent and extent
      $Concepts \leftarrow Concepts \cup (R, Q) @ P_a$
    end for
    $Patterns \leftarrow Patterns \cup \{P_a\}$
  end for
end for

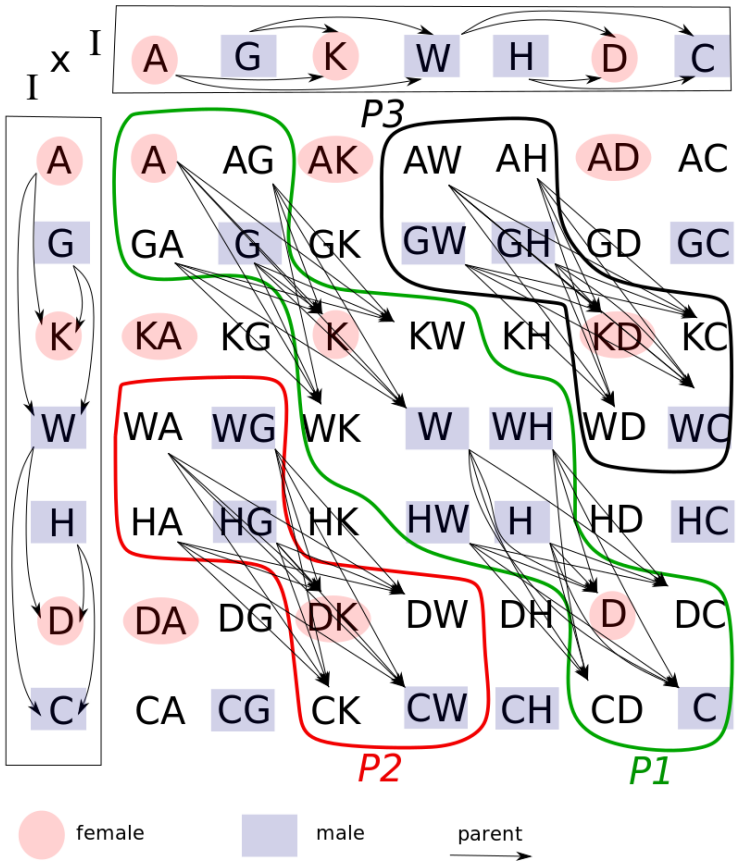

Figure 6: Graphical representation of the alignment (categorical product) $I \times I$ of the incidence relation I of the running example with itself. The graph I can be seen on the left (vertically), and at the top (horizontally). The graph pattern resulting from the product is made of all the rest. The three connected components (with at least 2 nodes) are delimited and named (P1, P2, P3).

duplication in the sense that several nodes play exactly the same role in the pattern, for example, nodes WA and HA in Figure 7 because they have the same relations with the same nodes. Nodes DK and CW are not duplicate nodes because they have a different gender. More formally, two nodes are duplicates in a pattern P if there is an automorphism of P (i.e., an isomorphism to the pattern itself) that maps one of the object to the other, and if their merging into one node defines a retract of P. Duplication can lead to a combinatorial explosion in the size of patterns P_p after several iterations. The optimization (**Line 7**, function *removeDuplicateNodes*()) consists in first partitioning P_p-nodes by grouping those that are equivalent modulo automorphism, and then keeping only one node out of each group. The only consequence is to miss

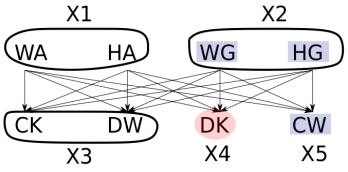

Figure 7: Connected component P2 from Figure 6 redrawn for readability, and with duplicate nodes grouped. Each group is labeled by a variable X_i.

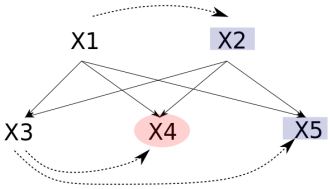

Figure 8: Simplified representation of P2 after merging duplicate nodes. The dashed arrows show the possible retracts of the pattern, e.g. a retract can be obtained by mapping X1 onto X2.

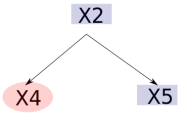

Figure 9: The core of P2 is the smallest retract.

the concept intents that have several projected nodes from the same group. However, those intents could be easily retrieved by duplicating projected nodes in generated intents. For example in Figure 3, the binary concept "parent couple" (Q3c,Q3c) is obtained by duplicating the unary concept "parent" Q3c. Figure 7 shows pattern P2 from Figure 6 where duplicate nodes are grouped, and where each group is labelled with a variable. Figure 8 shows pattern P2 after simplification, i.e. replacing duplicate nodes by variables.

Grouping patterns by core. The core of a graph pattern is its smallest retract, and as such it is its most compact representation modulo homomorphism. It is therefore valuable for visual presentation to compute the core P_c of generated patterns (**Line 9**), and to group patterns in the basis according to their core (**Line 10**). Figure 8 shows the possible retracts of pattern P2 as mappings between variables (dashed arrows), and Figure 9 shows the core of P2.

Iterating with new patterns. The new patterns P_a obtained as explained above become an input for the next stage by being queued in *Patterns* (**Line 16**). In case a generated pattern P_a has already been seen (module isomorphism) in a previous step it is ignored (**Line 8**).

6.2.2. Generation of k-Concepts

Algorithm 1 derives k-concepts from each pattern in the pattern basis (**Lines 11-15**). However, the concepts are not displayed by themselves but as part of a pattern. The computation of concepts enables to identify for each concept what is the relevant part of the pattern for its interpretation.

For each connected graph pattern P_a, and for each projection tuple $\overline{x}$ taken from P_a-nodes, the PGP $(\overline{x}, P_a)$ is a concept intent. Once a projection tuple has been chosen (**Line 11**), the graph pattern P_a can be simplified into its smallest retract P_r containing the nodes in $\overline{x}$ (**Line 12**). From there a concept can be formed by defining $Q = (\overline{x}, P_r)$ as the intent, and $R = Q'$ as the extent (**Line 13**). Finally, that concept can be added to the set of concepts. We keep a record of the pattern in which the concept was formed (@P_a in **Line 14**) in

order to know where the concept can be read in the display of the pattern basis. The retracted pattern P_r gives the subset of P_a-nodes that are relevant for the concept.

6.3. Implementation

We have implemented the above algorithm as a prototype in about 1700 lines of OCaml[2]. It takes as input a graph context, i.e. a set of directed hyperedges. It returns as output a compact representation of the pattern basis, where connected concepts can be derived. As an example, we refer to Figure 3 about the British royal family. That compact representation is presented as a collection of patterns, e.g. 4 patterns in the example. For each pattern, its core nodes are shown in white while other projection nodes are shown in grey. In the example, the 3rd pattern has 3 core nodes (Q3a,Q3b,Q3d), and 2 other nodes (Q3c,Q3e). Each node, e.g. Q3c, also represents the unary concept projected at that node. It is identified by the pattern number (here '3') and a letter identifying the node in the pattern (here 'c'). The graph pattern of the intent of a concept, e.g. Q3c, is the subgraph of the pattern induced by the set of nodes made of: the projected node (here Q3c), the core nodes (here Q3a, Q3b, Q3d), and possibly other nodes between the projected node and core nodes (here none). For clarity, the latter in-between nodes are indicated between brackets after the label of the projected node. N-ary concepts can be derived similarly by picking several nodes. For example, the binary concept (Q4e,Q4j) has the intent whose graph pattern is the subgraph of pattern 4 induced by the 6 core nodes plus the projected nodes Q4e and Q4j.

The prototype has options to control the display of concepts: `-dot` for graphical representation (`.dot` file) or `-txt` for a Prolog-like textual form; `-ext` to label each unary concept by its extent (as a list of objects) or `-supp` to only show the size of the extent (support); `-only-cores` to only show the core nodes; `-minsupp` to only show the concepts with *minsupp* objects in their extent; `-maxsize` to only show the patterns with *maxsize* nodes. For example, Figure 3 shows that the extension of the unary concept Q3c contains Kate, Diana, William, and Charles (i.e., the parents). The prototype also has options to control the computational complexity, at the cost of incomplete results: `-nodiag` to ignore patterns (P_a in Algorithm 1) that contains a node aligned on itself, and hence that includes the whole graph context; `-l` to limit the number of *levels* of intersection. A pattern P_1 produced by the first intersection $I \times I$ has level 1; then a pattern P_2 produced by a subsequent intersection $P_1 \times I$, and that was not produced at level 1, has level 2; and so on. An interesting property of levels is that a concept produced at level l has a support that is at least $l + 1$.

[2]Source code, datasets, and concept sets at `https://bitbucket.org/sebferre/g-fca`

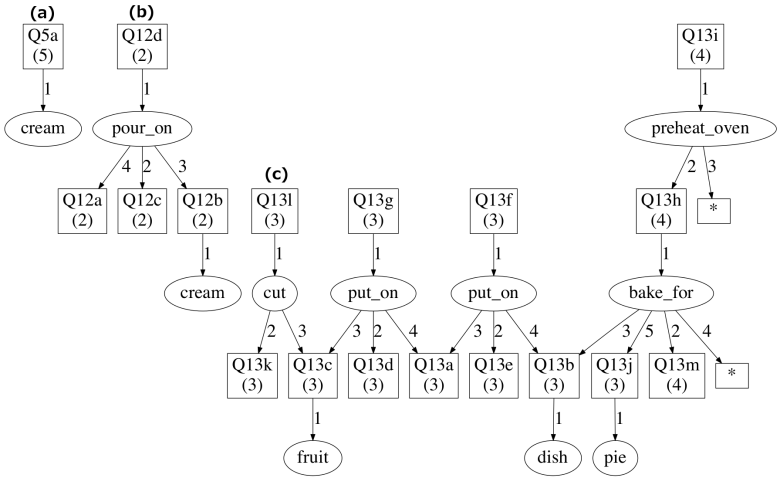

Figure 10: Three concepts extracted from the recipes.

7. Use Cases and Experiments

In this section, two use cases are presented in order to show the interest of Graph-FCA concepts and their PGPs. The first one extracts workflow patterns from cooking recipes, highlighting the benefits of n-ary relationships. The second one extracts linguistic structures from parse trees, comparing two graph modellings. Then some quantitative aspects are discussed, about runtime and number of generated concepts.

7.1. Extraction of Concepts in Recipes

We have conducted experiments on recipes. Four recipes are modelised: chocolate apple pie, strawberry-apple pie, mango-coconut pie and condoeuvre (Rhubarb pie or gooseberry pie). In this example, n-ary relations are used to represent temporal constraints between actions, and entities manipulated by actions. For instance, "put_on" is a quaternary relationship relating (1) start, (2) end, (3) object (e.g., "fruit"), and (4) destination (e.g., "pastry"). All action attributes use a similar schema (e.g., "cut", "bake_for"); other attributes represent types of ingredients or ustensils (e.g., "cream", "dish"). The graph context representing those four recipes is made of 111 nodes and 137 hyperedges. From those four recipes, 43 patterns are extracted in less than 1s. An excerpt is given in Figure 10. Some patterns are very small and very frequent as (a) in Figure 10. They represent ingredients (e.g., sugar, cream) or atomic actions (e.g., "put something on something", "pour on"). Some patterns are larger but less frequent as (b) in Figure 10. They represent refinements of previous patterns or very specific actions (e.g., "pour cream on something"). Finally, some patterns are large and still frequent as (c) in Figure 10. They correspond to the abstraction of many recipes. In this example, the pattern represents

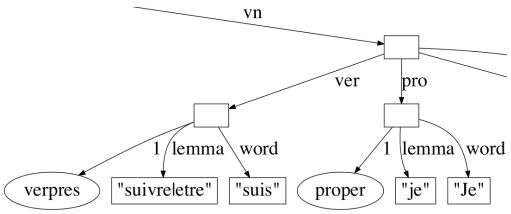

Figure 11: Excerpt of the composition modelling for the phrase "Je suis".

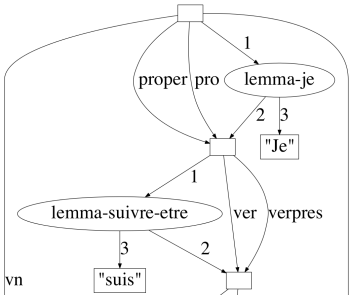

Figure 12: Excerpt of the sequential modelling for the phrase "Je suis".

an abstraction of a pie recipe. It means: "cut the fruit in order to put it on something (often a pastry), which is put on a dish, which is baked, after preheating the oven, in order to obtain a pie". As an example of n-ary concept, the ternary concept (Q13j,Q13a,Q13c) can be used to relate a pie (Q13j) to the kind of base (Q13a) and fruit (Q13c) it is made of, while abstracting over other details of the recipe.

7.2. Extraction of Concepts in Parse Trees

We have also conducted experiments on the poem of the french poet Charles Baudelaire named *"La beauté"*[3]. The goal of those experiments is the extraction of syntactic structures appearing in the text. We use the french chunker of the tool Treetagger [29][4] in order to automatically build the parse tree of the text.

From the computed parse tree, we propose two modellings to represent the poem: the first model represents the parse tree without taking into account the order between words, only the composition relation, whereas the second model explicitly encodes the order between words in addition to the composition relation. Both modellings are presented in the sequel and then the extracted patterns are discussed.

Composition Modelling. In the first representation of the text, edges represent part-of relationships between phrases and words. For instance, Figure 11 shows an excerpt of the graph of the sentence *"Je suis belle, ô mortels !"*. In this representation, the sentence ("s") is described as containing a verbal nucleus ("vn") which in turn contains: (1) a verb ("ver") which has a word ("suis"), a lemma ("suivre" or "être") and another part-of-speech (POS) information ("verpres", i.e. verb in present) ; (2) a pronoun ("pro") which has a word ("je"), a lemma ("je") and another part-of-speech information ("proper", i.e. personal pronoun) ; and two other words (a punctuation and an adjective). Note

[3] In "Les Fleurs du mal". Charles Baudelaire. 1857

[4] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

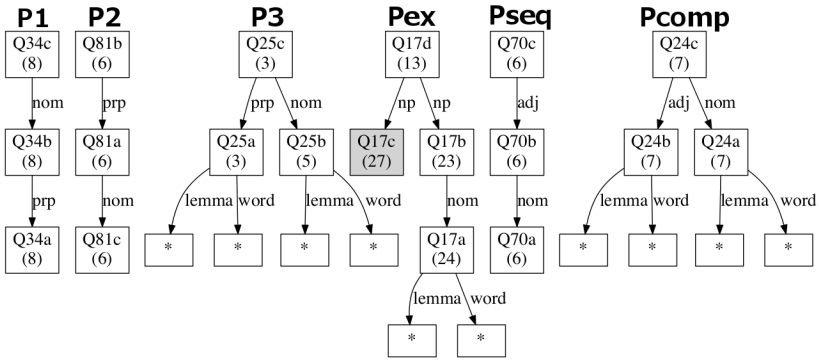

Figure 13: Concepts extracted from the poem.

that the part-of-speech tags ("ver", "vn", "proper", etc) are Treetagger tags, and are here used to label the part-of relationships.

Sequential Modelling. In the second representation, nodes represent positions between words, and edges represent phrases and words between those positions. POS tags and lemmas are used as attributes. For instance, Figure 12 shows an excerpt of the graph of the sentence *"Je suis belle, ô mortels !"*. For instance, the word "Je" is represented by three edges from the top node: attributes "proper", "pro" (POS information), and "lemma-je" (lemma). The fact that edge "vn" overlaps edges "pro" and "ver" represents composition. Note that with this representation where nodes are positions and not words, the interpretation of n-ary concepts is not more difficult than the one with the composition modelling. For instance let us consider a ternary concepts with three nodes: x, y, z; and three edges between them: $det(x, y)$, $noun(y, z)$, $np(x, z)$. It can be interpreted as the concept of noun phrases that contain a determiner followed by a noun such that y is the position locating the boundary between the determiner and the noun. This kind of pattern could be useful to discover grammatical rules from treebank datasets.

Discussion about extracted patterns. When extracting concepts from both representations, we note that more patterns are extracted from the sequential model than from the composition model. For instance, with parameter $maxsize = 10$ (maximum number of nodes per PGP), 284 patterns are extracted in the sequential model instead of 68 patterns for the composition model[5]. Indeed, in the sequential model, the graph structure is rigid, the order between words is really important. When concepts are extracted, the sequential model thus generates more distinct patterns. For instance, let us consider P1, P2 and P3 in Figure 13.

[5]Note that, the extraction of the 284 patterns in the sequential model takes about 4s and the extraction of the 68 patterns in the composition model takes about 20s.

P1 and P2 are extracted from the sequential model whereas P3 is extracted from the composition model. The three patterns represent the association between a preposition and a noun. However the composition modelling generates only one concept when the sequential modelling generates two concepts taking into account the ordering of the two words.

Some structural information about the text can be retrieved in the concepts. For instance, in Figure 13, P_{ex}, extracted from the composition model, exhibits an obvious pattern in the poem, i.e. a noun phrase (np) which contains a noun (*nom*). The size of the extent of unary concepts is given between brackets. Concept (Q17b) can be read as "a noun phrase that contains a noun and that belongs to something". Note that the size of the extension of (Q17b) is 23 objects. Concept (Q17a) can be read as "a noun that belongs to a noun phrase that belongs to something". Note that the size of the extension of (Q17a) is 24 objects. The size of (Q17a) is thus greater than the size of (Q17b), it means that an object in the extension of Q17b contains not only one but two nouns ("un rêve de pierre"). Concept (Q17c) can be read as "a noun phrase that belongs to something that contains a noun phrase that contains a noun". Note that the size of the extension of (Q17c) is 27 objects. This concept is interesting, indeed it exhibits the fact that a noun phrase does not necessarily contain a noun in this text, for instance it can also be a pronoun: *"où"* (where), *"chacun"* (everyone). It also shows that two noun phrases can be found in the same structure.

The information conveyed by the two modellings are not the same, however both representations are interesting. For instance, let us consider the two patterns P_{seq} and P_{comp} in Figure 13 that represent the association of a noun and an adjective. The size of the extension of concepts in P_{seq} is 6 and the size of the extension of concepts in P_{comp} is 7. Indeed, the six phrases that match P_{seq} also match P_{comp}. However the phrase *"toutes choses plus belles"*, which contains an adverb ("*plus*") between the noun and the adjective, only match Q_{comp}. The sequential modelling allows to take into account the order between words, it is more accurate whereas the composition modelling allows for more general patterns. The choice of the more appropriate modelling depends on the task.

7.3. Quantitative Experiments

Finally we have conducted some quantitative experiments on a longer text named "Le Horla" written by the french writer Guy de Maupassant[6]. The main goals of this experiment are: first to compare the runtime of both representations (composition modelling and sequential modelling), second to study the behaviour of the approach (w.r.t. runtime, number of extracted unary concepts and number of extracted patterns) on real graphs.

[6]Text available here: `https://fr.wikisource.org/wiki/Le_Horla_(recueil)/Le_Horla`

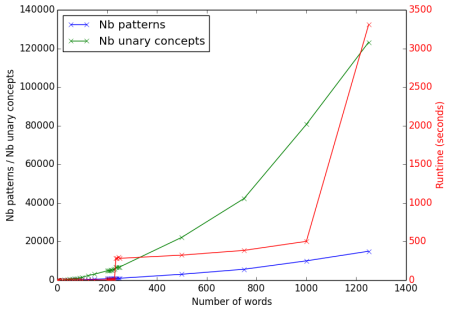

Figure 14: Runtime, number of patterns and number of unary concepts for the sequential representation.

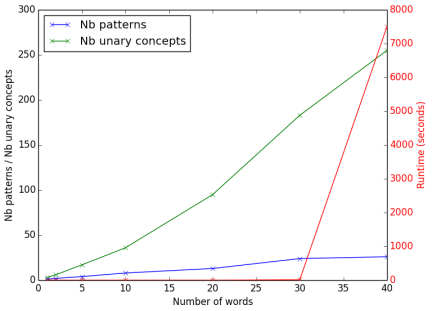

Figure 15: Runtime, number of patterns and number of unary concepts for the compositional representation.

7.3.1. Methodology

We have created subsets of the whole text called $sampling_n$, by taking the n first words of the text. Then we have used Treetagger to build the parse tree of each $sampling_n$. The composition and sequential representations of each $sampling_n$ is computed and finally the Graph-FCA approach is applied on them.

Note that for the computation of the patterns the "-l 2" option is used in order to limit the computation time without missing too many concepts, as we show below. The meaning of the "-l k" option is that the extracted unary concepts are those whose intent that can be obtained from the intersection of maximum $1+k$ object descriptions (see Definition 9). The impact on the number of extracted patterns and extracted unary concepts is really small. Indeed, we have computed them for different values of k. Those experiments have shown that, most of the time, the number of patterns and unary concepts does not increase with k values greater or equal to 3 or 4. In addition, the difference between the number of unary concepts from $k = 2$ to $k = 3$ is about 2% and for the number of patterns is about 1%. For instance for $sampling_{230}$, with $k = 2$ there are $5,778$ extracted concepts whereas with $k = 3$ there are $5,870$ extracted concepts and with $k \geq 4$ there are $5,871$ extracted concepts. In the same way, for $sampling_{230}$, with $k = 2$ there are 879 extracted patterns whereas with $k \geq 3$ there are 884 extracted patterns.

The experiments have been performed on a MacBookPro, with the following features: 3,1 GHz Intel Core i7 processor, 16 Go 2133 MHz LPDDR3 memory and MacOS sierra (version 10.12.6). The Graph-FCA implementation is the one compiled for Mac.

7.3.2. Results and Interpretation

Figures 14 and 15 show three measures as a function of the number of words in the sample, respectively for the composition and sequential models. The

three measures are: (a) total runtime for the generation of the concepts, (b) number of concepts in the compact representation of concepts, and (c) the total number of unary concepts, which is equal to the cumulated number of nodes in those patterns.

Runtime. In the composition model, the runtime increases in a more than exponential way, and we had to stop the measures after 40 words. The explanation for that behaviour is that some nodes have several successors through the same relationship, e.g. a sentence containing up to 4 "verbal nucleus". This implies that for n successors there are n^n distinct homomorphisms, only for the subpattern concerning the node and its successors. In the sequential model, the runtime increases more steadily, and we managed to reach 1250 words. Note that the Graph-FCA context generated for that sample of 1250 words has 2900 nodes and 4754 edges. The better performance of the sequential model can be explained by the more rigid structure of its graph contexts, hence by a much smaller set of homomorphisms. However, two steep increases can be observed in the runtime curve, after 200 words and 1000 words. After inspection of the parse trees of the sentences at those positions, we have identified errors made by Treetagger that produced unusual graph patterns for the sequential model, e.g. a NP constituent starting with another NP constituent. This kind of pattern entails the same kind of combinatorial explosion as in the composition model.

Number of patterns and concepts. The main observation about the number of patterns and the number of concepts is that, in both modelling, the number of patterns is significantly lower than the number of unary concepts, up to 10 times. This demonstrates the usefulness of our compact representation of Graph-FCA concepts. Note that no information is lost in the compact representation because all unary concepts, and even n-ary concepts, can be read directly through patterns.

8. Related Work

Conceptual Graphs. Graph-FCA graph patterns bear much similarity with Conceptual Graphs (CG) [30, 4], and we re-used their graphical notation. We adopted a slightly simpler formalization by not distinguishing between concept types, relation types, and individual markers, which are all modeled with attributes in Graph-FCA for uniformity. The semantics of knowledge graphs (e.g., CG type hierarchies, RDF Schema) is not natively handled but FCA techniques like scaling can easily be applied to Graph-FCA. Those differences are minor, and the benefits of Graph-FCA lies in projected graph patterns (PGPs), PGP intersection, and concept formation from a knowledge graph. For comparison, reasoning with CGs is mostly based on graph homomorphisms, typically between a source graph and a query graph.

Graph mining. Graph-FCA concept formation works differently from graph mining approaches [21, 34, 32] because they generally consist in finding frequent substructures in a collection of graphs (e.g., molecules), and they use subgraph isomorphism instead of homomorphisms.

Inductive Logic Programming (ILP). Graph-FCA also bears some similarity with ILP [23]. A Graph-FCA incidence $(\overline{o}, a)$ can be seen as a logical atom $a(\overline{o})$, the graph context can be seen as the background knowledge (only facts here), and a PGP can be seen as an anonymous clause. PGP inclusion $\subseteq_q$ then corresponds to θ-subsumption, and the categorical product of graph patterns corresponds to the *lgg* (least general generalization) introduced by Plotkin [24]. ILP is generally designed and applied in a supervised setting, learning rules with a fixed head, whereas Graph-FCA is an unsupervised approach that discovers concepts whose intents are PGPs. In ILP terms, it is therefore a way to discover new predicates that can be defined in terms of the background predicates. ILP has also been studied in an unsupervised setting, for the mining of frequent patterns and/or closed patterns [13]. Closed patterns correspond to the concept intents of FCA. Those patterns are generally mined in a *multiple graphs* settings because it gives a well-founded definition of frequency, and hence of closed patterns. Graph-FCA is in a *single graph* setting. The solution that has been proposed so far to have a well-defined notion of closed pattern in the *single graph* setting is based on *range restriction*, i.e. by forbidding the closure of a graph pattern to introduce additional nodes. This is different to our approach that consists into using PGPs, i.e. adding a projection tuple to graph patterns. The difference is best seen on an example. Taking the British royal family context in Figure 2, graph pattern $\{parent(x, y)\}$ is closed under range restriction, while its closure in Graph-FCA is $Q = ((x, y), \{parent(x, y),\ parent(x, f),\ male(f),\ parent(x, m),\ female(m),\ parent(b, y),\ parent(b, f),\ parent(b, m),\ male(b)\}$, which has the benefits to show that in every child-parent relationship the child has a father f, a mother m, and a brother b.

Logical Concept Analysis, Pattern Structures. Previous FCA extensions, Logical Concept Analysis (LCA) [8] and Pattern Structures (PS) [10], have definitions for the Galois connection that look much like those of Graph-FCA (Theorem 19). However, in those extensions, descriptions apply to each object separately, without relationships between objects. In Graph-FCA, the whole knowledge graph serves as a description, not only for individual objects but also for tuples of objects. This allows for describing n-ary relationships between objects, as well as discovering new relationships (concepts) as complex combinations of primitive ones. Now, as hinted in [19], it is technically possible to characterize Graph-FCA concepts with pattern structures. The k-concepts of a graph context $K = (O, A, I)$ correspond to the concepts of the pattern structure $(O^k, (\mathcal{Q}_k, \cap_q), \delta)$, where $\delta(\overline{o}) = Q_k(\overline{o})$. However, it offers a less natural and elegant formalization because an indefinite number of pattern structures are necessary (one for each k), and the whole graph context is duplicated into each description $\delta(\overline{o})$. Moreover, if pattern structures algorithms were to be ap-

plied, it would result in a lot of duplication in computations and representations compared to our algorithm.

Polyadic Concept Analysis. Another extension of FCA to n-ary relationships is *polyadic concept analysis* [31, 2], a generalization of triadic concept analysis [22]. However, the context and concepts are very different in nature. First, a polyadic context is a *single* n-ary relationship between n different domains of entities (e.g., objects, attributes, conditions, ...). For example, a triadic context may relate points of interest (POI), tags, and users where the incidence (Eiffel Tower, building, John) means that John has put tag 'building' on the Eiffel Tower. In contrast, a Graph-FCA context uses several n-ary relationships (Graph-FCA attributes) to relate the entities of a single domain (Graph-FCA objects). Second, a polyadic concept is a tuple of n subsets of the n domains, such that every selection of one entity per subset defines a tuple of entities that belongs to the context. For example, the concept ({Eiffel Tower, Tour Montparnasse}, {building, Paris, sightseeing}, {John, Mary}) says that both John and Mary put tags 'building', 'Paris', and 'sightseeing' to both the Eiffel Tower and Tour Montparnasse. In contrast, a Graph-FCA concept always has two components: an n-ary relation that in general does not belong to the context, and a projected graph pattern.

Concept lattices of relational structures. Graph-FCA shares theoretical foundations with Kötters' work on concept lattices of relational structures [20]. The relationships are not obvious as Graph-FCA is formalized in terms similar to FCA while Kötters' work is formalized in terms of category theory. Our graph contexts correspond to his *relational structures*, our PGPs (and their intersection/union) correspond to his windowed relational structures (and their product/coproduct), and our n-ary concepts roughly correspond to his concepts for some set of n variables. Concept lattices of relational structures have been applied to RDF graphs, where concept intents are presented as the Most Specific Queries (MSQ) of a set of answers [17]. This work sketches a nice representation of unary concepts that bears some similarity with our representation. However it is difficult to give a more precise comparison due to the lack of a general definition.

Relational Concept Analysis. Another FCA extension, Relational Concept Analysis (RCA) [27], also discovers concepts in a graph context. RCA contexts are limited to unary and binary attributes, and RCA concepts are limited to unary concepts. However, the main difference lies in the nature of concept intents: *rooted tree patterns* instead of projected graph patterns. A comparison of Kötters' formalization with RCA [18] shows that the rooted tree patterns correspond to the unfolding of PGPs up to some depth. Therefore, as rooted tree patterns form a subset of PGPs, every extent of a RCA concept is the extent of some unary Graph-FCA concept, whatever the number of RCA iterations. The converse does not hold because cycles (e.g., in concept Q3e in Figure 3) cannot be expressed as rooted tree patterns. Other advantages of Graph-FCA are (1)

a declarative, rather than iterative, characterization of concepts, and (2) a self-contained graph-based representation of concept intents instead of cascading references to concepts.

9. Conclusion and Future Work

In this paper, we have proposed an extension of FCA, Graph-FCA, where the context is a *knowledge graph*, the objects are replaced by tuples of objects, concept intents are *projected graph patterns (PGP)*, and concept extents are *object relations*. The main advantages of Graph-FCA over other FCA extensions are: (a) the support for n-ary relationships between objects, (b) the possibility to discover n-ary concepts, and (c) the representation of concept intents as projected graph patterns, a versatile knowledge representation akin to first-order rules and SPARQL queries. All fundamental results of FCA are retrieved, and classical FCA is the fragment of Graph-FCA where only unary attributes and unary concepts are used.

We have proposed an algorithm to compute graph concepts in knowledge graphs. In particular, we tackle the problem of the generation and representation of the PGPs representing concept intents in a compact way. We also describe two use cases. The first use case, on cooking recipes, shows the interest of Graph-FCA for n-ary relations. The other use case, on textual data, allows to discuss two kinds of modelling (with or without sequentiality). Finally, we have discussed quantitative aspects of the approach with respect to the chosen modelling and show the usefulness of the compact representation of Graph-FCA.

With the two use cases we have seen that PGPs offer expressive patterns that can mix sequentiality, temporality, and composition thanks to n-ary relations. However, the set of extracted concepts can be large, and costly to compute. Further work is to improve the algorithm, the compact representation of graph concepts, and to find a way to facilitate, for a user, navigation among them.

References

[1] Allard, P., Ferré, S., Ridoux, O.: Discovering functional dependencies and association rules by navigating in a lattice of OLAP views. In: Kryszkiewicz, M., Obiedkov, S. (eds.) Concept Lattices and Their Applications. pp. 199–210. CEUR-WS (2010)

[2] Cerf, L., Besson, J., Robardet, C., Boulicaut, J.F.: Closed patterns meet n-ary relations. ACM Transactions on Knowledge Discovery from Data (TKDD) 3(1), 3 (2009)

[3] Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. 1(1), 146–166 (1989)

[4] Chein, M., Mugnier, M.L.: Graph-based knowledge representation: computational foundations of conceptual graphs. Advanced Information and Knowledge Processing, Springer (2008)

[5] Chekol, M.W., Euzenat, J., Geneves, P., Layaïda, N.: SPARQL query containment under RDFS entailment regime. In: Automated Reasoning (IJCAR), pp. 134–148. Springer (2012)

[6] Ferré, S.: A proposal for extending formal concept analysis to knowledge graphs. In: Baixeries, J., Sacarea, C., Ojeda-Aciego, M. (eds.) Int. Conf. Formal Concept Analysis (ICFCA). pp. 271–286. LNCS 9113, Springer (2015)

[7] Ferré, S.: Conceptual navigation in RDF graphs with SPARQL-like queries. In: Kwuida, L., Sertkaya, B. (eds.) Int. Conf. Formal Concept Analysis. pp. 193–208. LNCS 5986, Springer (2010)

[8] Ferré, S., Ridoux, O.: A logical generalization of formal concept analysis. In: Mineau, G., Ganter, B. (eds.) Int. Conf. Conceptual Structures. pp. 371–384. LNCS 1867, Springer (2000)

[9] Ferré, S., Cellier, P.: Graph-fca in practice. In: Haemmerlé, O., Stapleton, G., Faron-Zucker, C. (eds.) Int. Conf. Conceptual Structures (ICCS) - Graph-Based Representation and Reasoning. pp. 107–121. LNCS 9717, Springer (2016), `https://hal.inria.fr/hal-01405491`

[10] Ganter, B., Kuznetsov, S.: Pattern structures and their projections. In: Delugach, H.S., Stumme, G. (eds.) Int. Conf. Conceptual Structures. pp. 129–142. LNCS 2120, Springer (2001)

[11] Ganter, B., Wille, R.: Formal Concept Analysis — Mathematical Foundations. Springer (1999)

[12] Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York. (1999)

[13] Garriga, G.C., Khardon, R., De Raedt, L.: Mining closed patterns in relational, graph and network data. Annals of Mathematics and Artificial Intelligence 69(4), 315–342 (2013)

[14] Hahn, G., Tardif, C.: Graph homomorphisms: structure and symmetry. In: Graph symmetry, pp. 107–166. Springer (1997)

[15] Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)

[16] Huchard, M.: Analyzing inheritance hierarchies through formal concept analysis: A 22-years walk in a landscape of conceptual structures. In: MechAnisms on SPEcialization, Generalization and inHerItance (MASPEGHI). pp. 8–13. ACM (2007)

[17] Kötters, J.: Concept lattices of RDF graphs. In: Int. Work. Formal Concept Analysis and Application (FCA&A). CEUR, vol. 1434, pp. 81–91 (2015)

[18] Kötters, J.: Intension graphs as patterns over power context families. In: Concept Lattices and Applications (CLA). CEUR 1624 (2016)

[19] Kötters, J., Schmidt, H.W.: A database browser based on pattern concepts. Formal Concept Analysis Meets Information Retrieval (FCAIR) p. 47 (2013)

[20] Kötters, J.: Concept lattices of a relational structure. In: Pfeiffer, H., et al. (eds.) Int. Conf. Conceptual Structures for STEM Research and Education, pp. 301–310. LNAI 7735, Springer (2013)

[21] Kuznetsov, S.O., Samokhin, M.V.: Learning closed sets of labeled graphs for chemical applications. In: Kramer, S., Pfahringer, B. (eds.) Int. Conf. Inductive Logic Programming. pp. 190–208. LNCS 3625, Springer (2005)

[22] Lehmann, F., Wille, R.: A triadic approach to formal concept analysis. In: Int. Conf. Conceptual Structures. pp. 32–43. Springer (1995)

[23] Muggleton, S., Raedt, L.D.: Inductive logic programming: Theory and methods. Journal of Logic Programming 19,20, 629–679 (1994)

[24] Plotkin, G.: Automatic Methods of Inductive Inference. Ph.D. thesis, Edinburgh University (august 1971)

[25] Poelmans, J., Ignatov, D.I., Kuznetsov, S.O., Dedene, G.: Formal concept analysis in knowledge processing: A survey on applications. Expert systems with applications 40(16), 6538–6560 (2013)

[26] Prediger, S., Wille, R.: The lattice of concept graphs of a relationally scaled context. In: International Conference on Conceptual Structures. pp. 401–414. Springer (1999)

[27] Rouane-Hacene, M., Huchard, M., Napoli, A., Valtchev, P.: Relational concept analysis: mining concept lattices from multi-relational data. Annals of Mathematics and Artificial Intelligence 67(1), 81–108 (2013)

[28] Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: *et al.*, P.M. (ed.) The Semantic Web (ISWC). pp. 245–260. LNCS 8796, Springer (2014)

[29] Schmid, H.: Probabilistic part-of-speech tagging using decision trees. In: Int. Conf. New Methods in Language Processing (1994)

[30] Sowa, J.: Conceptual structures. Information processing in man and machine. Addison-Wesley, Reading, US (1984)

[31] Voutsadakis, G.: Polyadic concept analysis. Order 19(3), 295–304 (2002)

[32] Washio, T., Motoda, H.: State of the art of graph-based data mining. SIGKDD Explor. Newsl. 5(1), 59–68 (Jul 2003), http://doi.acm.org/10.1145/959242.959249

[33] Wille, R.: Conceptual graphs and formal concept analysis. In: Int. Conf. Conceptual Structures. pp. 290–303. LNCS 1257 (1997)

[34] Yan, X., Han, J.: Closegraph: mining closed frequent graph patterns. In: ACM Int. Conf. Knowledge discovery and data mining (SIGKDD). pp. 286–295. ACM (2003)