CONNOR: Exploring Similarities in Graphs with Concepts of Neighbors

H. Ambre Ayats, Peggy Cellier, Sébastien Ferré

CONNOR: Exploring Similarities in Graphs with Concepts of Neighbors

Hugo Ayats, Peggy Cellier, and Sébastien Ferré

Univ Rennes, INSA, CNRS, IRISA
Campus de Beaulieu, 35042 Rennes, France
{hugo.ayats,peggy.cellier,sebastien.ferre}@irisa.fr

Abstract. Since its first formalization, the Formal Concept Analysis (FCA) field has shown diverse extensions of the FCA paradigm. A recent example is Graph-FCA, an extension of FCA to graphs. In the context of Graph-FCA, a notion of concept of neighbors has been introduced to support a form of nearest neighbor search over the nodes of a graph. Concepts of neighbors have been used for diverse tasks, such as knowledge graph completion and relation classification in texts. In this paper, we present CONNOR, a Java library for the computation of concepts of neighbors on RDF graphs.

1 Introduction

Introduced in the early 1980's, Formal Concept Analysis (FCA) enables to analyze a collection of objects and properties as a hierarchy of concepts. Several extensions of FCA have been proposed to handle more complex objects and properties, such as multi-relational data: e.g., Relational Concept Analysis (RCA) [8] or Graph-FCA [3]. In the context of Graph-FCA, the notion of concepts of neighbors [5] has been introduced as a lazy learning approach to explore similarities between tuples of nodes in a graph.

Inspired by the work of Kuznetsov on Pattern Structures for classification [7], concepts of neighbors are only those concepts that result from the intersection of a given node (or tuple of nodes) with other (tuples of) nodes, in order to support a decision process about the given node. Compared to computing all concepts, this decreases the worst case number of concepts from exponential to linear. A parallel can also be made with the work of Codocedo *et al.* on cousin concepts [2], but instead of exploring concepts related to a given concept, we explore a more specific kind of concepts related to a given object. This can be seen as a form of conceptual *k Nearest Neighbors* (kNN) algorithm on graph nodes, and like for the kNN approach, possible applications are numerous. So far, concepts of neighbors have been applied to knowledge graph completion [5], query relaxation [4], and relation classification in texts [1].

This paper introduce CONNOR, a Java library for the computation of concepts of neighbors in the context of the semantic web. Based on the widely-used Apache Jena library, this library introduces data structures for representing Graph-FCA notions, concepts of neighbors included, and implements an efficient algorithm for computing concepts of neighbors on RDF graphs.

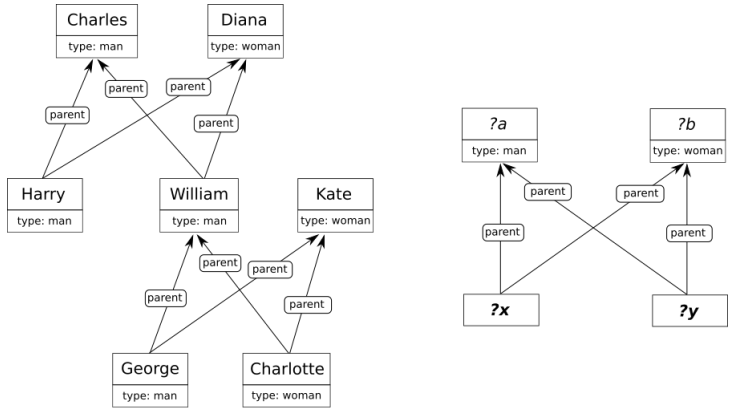

Fig. 1: (a) Context graph representing the British royal family
(b) PGP representing the sibling relationship between x and y

2 Theoretical Background

This section introduces definitions related to the concepts of neighbors. Details can be found in the reference journal paper [6].

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

Figure 1 (a) shows the graph context representing a fragment of the family tree of the British royal family as a graph context. The boxes are the objects (e.g. *Charlotte* or *Harry*), the types inside boxes are unary attributes (e.g. *man*, *woman*) and the arrow labels are the binary attributes (e.g. *parent*).

Definition 2. *A **graph pattern** $P \subseteq \mathcal{V}^* \times A$ is a set of labeled directed edges and labeled nodes with variables as nodes and attributes as labels.*

*A **projected graph pattern (PGP)** is a couple $Q = (\overline{x}, P)$ where P is a graph pattern and $\overline{x} \in \mathcal{V}^*$ a projection tuple. The **arity** of a PGP is the length of $\overline{x}$. A PGP of arity k is also called a k-PGP.*

Figure 1 (b) represents a 2-PGP defining the "sibling" binary relation (two persons sharing a male parent and a female parent). Projection variables are in bold. In practice, PGPs can be seen as queries on the graph context. In this paper we call **set of answers** of a k-PGP the set of the k-tuples of objects that are the answers of the PGP considered as a query.

A PGP **inclusion relation** (noted $\subseteq_Q$) can be defined to express the fact that a k-PGP Q_1 is more specific than another k-PGP Q_2 (i.e. by renaming the variables of Q_2 we can obtain a PGP having the same projection tuple than Q_1

and having its graph pattern included in the graph pattern of Q_1). In this case, we denote $Q_2 \subseteq_Q Q_1$. If $Q_1 \subseteq_Q Q_2$ and $Q_2 \subseteq_Q Q_1$, they are said equivalent ($Q_1 \equiv_Q Q_2$). The **most specific query** for a given answer set is the query (under equivalency relation) that is more specific to every other queries having this set as answer set.

Definition 3. *A **graph concept** of arity k (also called k-concept) is a pair (R, Q), R being a set of k-tuples of objects (called **extension**) and Q being a k-PGP (called **intension**), such that:*

- *R is the set of answers of Q;*
- *Q is the most specific k-PGP having R as set of answers (under the equivalence relation).*

For example, let us consider (R, Q) where Q is the PGP presented in Figure 1 (b) and R is the set of couples: $\{(Harry, William), \ldots, (Harry, Harry), (George, Charlotte), \ldots, (Charlotte, Charlotte)\}$. This concept represents the notion of sibling: the intension is a PGP describing the couple of persons having a same mother and a same father, and the extension all the couples of entities respecting this pattern.

Definition 4. *The **conceptual distance** between two k-tuples of objects $\overline{o}_1$ and $\overline{o}_2$ is the k-concept $\delta(\overline{o}_1, \overline{o}_2) = (R, Q)$ where the extension Q is the most specific k-PGP such that the intension R contains $\overline{o}_1$ and $\overline{o}_2$.*

It can be proven that this conceptual distance has properties similar to a numerical distance, such as positivity, symmetry and triangle inequality, by using the concept extension inclusion as a partial order and a notion of conceptual supremum as addition. The **extensional distance** $d(\overline{o}_1, \overline{o}_2) = |\delta(\overline{o}_1, \overline{o}_2).ext|$ can be used as a (degraded) numerical distance to evaluate the dissimilarity between o_1 and o_2 as the number of k-tuples between them.

Definition 5. *The **concepts of neighbours** of a k-tuple $\overline{o}$ is the set of k-concepts $C\text{-}N(\overline{o}) = \{\delta(\overline{o}', \overline{o}) | \overline{o}' \in O^k\}$.*

The **proper extension** of a concept $\delta.propExt$ for $\delta \in C\text{-}N(\overline{o})$ is the set of k-tuples of its extension that are not in the extension of a more specific concept. The set of proper extensions forms a **partition** of O^k, and therefore the set of extensions forms a hierarchical clustering of O^k.

Figure 2 presents the five concepts of neighbors of Charlotte in the graph context presented in Figure 1 (a). On the right, the extensions are presented as a Venn diagram, and on the left the intensions are expressed in plain English for simplicity. The first concept has for intension the whole graph centered on Charlotte and has only Chalotte in its extension. Then there are two larger concepts, one describing the children of Kate and William (Charlotte and George), and another one describing the women. Then there is an even larger concept describing the people having a father, a mother and a sibling (Harry, William, Charlotte, and George). Finally, there is the top concept, having an empty intension and O as extension.

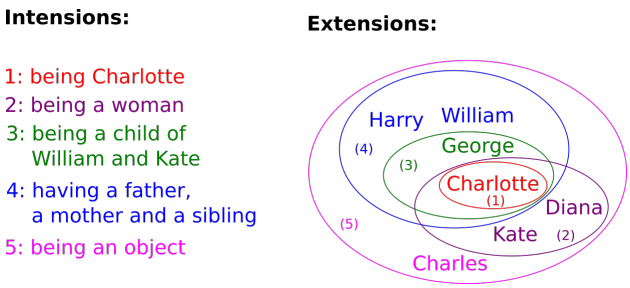

Fig. 2: Concepts of Neighbors of Charlotte

3 CONNOR: Computation of CONcepts of NeighbORs with a Java Library

In this section we present CONNOR, a Java library for the computation of the Concepts of Neighbors. This library is a free and open-source software[1], based on Apache Jena[2], a well-known Java library for the Semantic Web. As the Semantic Web is based on the RDF standard, and as RDF graphs only represent labelled directed multigraphs, the library only handle graph contexts with unary relations (node labels) and binary relations (labelled directed edges). This library implements data structures for manipulating concepts of neighbors, as well as an algorithm for their computation. Table 1 presents an example code for the computation of the concepts of neighbors of Charlotte in the graph context presented in Figure 1 (a).

The CONNOR library is structured into four main classes, which are detailed below.

`ConnorModel` This class encapsulates an RDF model that represents a graph context. There are two ways to create a model using this class. The first way consists in creating an empty model and adding triples one by one. The second way consists in creating a model from a pre-existing RDF model. Section (1) of Table 1 shows an example of creation of `ConnorModel` from a RDF graph.

`ConceptOfNeighbors` As its name tells, this class represents a concept of neighbors, and hence a graph concept too. As expected, an object of this class is characterized by its intension (decomposed into two attributes : the list of the projection variables and the graph pattern), its extension and its proper extension. Those elements can be accessed through the methods `getProjectionVars()`, `getIntensionBody()`, `getExtension()` and `getProperExtension()`. The creation of objects of this class is handled by the `ConnorPartition` class, and should not be done by library users.

[1] Accessible here: `https://gitlab.inria.fr/hayats/CONNOR`
[2] `https://jena.apache.org/`

```
// (1) Creation of the model
ConnorModel model = new ConnorModel(rdfModel);
// (2) Creation of the target
List<String> target = new ArrayList<>({"http://example.
    org/royal/Charlotte"});
// (3) Creation of the init table
Var var1 = Var.alloc("Neighbor_0");
Table table = new Table(new List<Var>({var1}));
rdfModel.listSubjects().forEachRemaining(subject ->
        table.addBinding(new Binding(var1, subject));
);
// (4) Creation of the partition
ConnorPartition partition = new ConnorPartition(model,
    target, table, 0);
// (5) Creation of the thread and computation of the C-N
AtomicBoolean cut = new AtomicBoolean(false);
ConnorThread thread = new ConnorThread(partition, cut);
thread.start();
thread.join(2 * 1000); // 2000ms = 2s
cut.set(true);
// (6) Print of the results
System.out.print(partition.toJson());
```

Table 1: Example of usage of CONNOR

`ConnorPartition` This class is central to the computation of concepts of neighbors. It takes its name from the fact that, as presented above, the proper extensions of the concepts of neighbors form a partition of the set of objects. This class contains all the information needed for the computations of concepts of neighbors, such as of course the graph context (represented by a `ConnorModel`), the concepts of neighbors once the computation is done (represented by a collection of `ConceptOfNeighbors` objects), but also the tuple of objects (called *target*) for which we want to compute the concepts of neighbors. A `ConnorPartition` object can be translated into JSON for serialization and further processing. An exemple of serialization in JSON is given by Section (6) of Table 1.

A key aspect of this class is that it implements an *anytime* algorithm for the computation of concepts of neighbors. Presented in [4], this algorithm starts with the top concept and, by successively trying to add elements to the intension, refines it in more specific concepts that still include the target. This way, the algorithm can be interrupted at each refinement step.

To use this class, the base constructor of this object takes as argument a `ConnorModel` object, the target (represented by a list of URIs), the partition domain and a `maxDepth` parameter. An example of usage is given by Section (4) of Table 1. The partitioning domain is called `initializationTable`, which is

is a `Table` object which represents a set of tuples of entities. The role of this argument is to stipulate which set of tuples should appear in the extensions of the concepts. An example of construction of such a table is given in Section (5) of Table 1. Concerning the `maxDepth` parameter, its role is to set, if desired, a limit to the depth of the intension from the elements of the target. If set to zero, no limit is applied.

The class method to call in order to launch the computation of concepts of neighbors is called `fullPartitioning(cut)`. Taking a mutable Boolean named `cut` as a parameter, it refines the partition until convergence or until `cut` is switched to `true`.

`ConnorThread` This class encapsulates the computation of concepts of neighbors using a `ConnorPartition` in a Java thread, so that the main thread just needs to launch this thread and switch `cut` to `true` when desired. An example of usage is given by Section (5) of Table 1.

4 Conclusion

This paper introduces CONNOR, a Java library for the computation of concepts of neighbors on RDF graphs. After having introduced the theoretical notions related to Graph-FCA (an extension of FCA to graphs) and to the concepts of neighbors, this paper presents the main classes of CONNOR, and details their working through an example code for the computation of concepts of neighbors.

References

1. Ayats, H., Cellier, P., Ferré, S.: A Two-Step Approach for Explainable Relation Extraction. In: Int. Symposium on Intelligence Data Analysis. pp. 14–25 (2022)
2. Codocedo, V., Lykourentzou, I., Napoli, A.: A contribution to semantic indexing and retrieval based on FCA - An application to song datasets. International Conference on Concept Lattices anf Their Applications p. 12 (2012)
3. Ferré, S.: A Proposal for Extending Formal Concept Analysis to Knowledge Graphs. In: International Conference on Formal Concept Analysis. vol. 9113, pp. 271–286. Springer International Publishing, Cham (2015)
4. Ferré, S.: Answers Partitioning and Lazy Joins for Efficient Query Relaxation and Application to Similarity Search. In: The Semantic Web. pp. 209–224. Cham (2018)
5. Ferré, S.: Application of Concepts of Neighbours to Knowledge Graph Completion. Data Science Pre-press(Pre-press), 1–28 (2020)
6. Ferré, S., Huchard, M., Kaytoue, M., Kuznetsov, S.O., Napoli, A.: Formal Concept Analysis: From Knowledge Discovery to Knowledge Processing. A Guided Tour of Artificial Intelligence Research: Volume II: AI Algorithms pp. 411–445 (2020)
7. Kuznetsov, S.O.: Fitting Pattern Structures to Knowledge Discovery in Big Data. In: Int. Conf. on Formal Concept Analysis. vol. 7880, pp. 254–266 (2013)
8. 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)