Int. J. Approx. Reason. 164 (2024) 109059

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

Concepts of neighbors and their application to instance-based learning on relational data

H. Ambre Ayats, Peggy Cellier, Sébastien Ferré [*,1]

Univ Rennes, INSA Rennes, CNRS, Inria, IRISA - UMR 6074, Rennes, F-35000, France

ARTICLE INFO

Keywords:
Relational data
Knowledge graph
Instance-based learning
Formal concept analysis
Graph-FCA
Concepts of neighbors

ABSTRACT

Knowledge graphs and other forms of relational data have become a widespread kind of data, and powerful methods to analyze and learn from them are needed. Formal Concept Analysis (FCA) is a mathematical framework for the analysis of symbolic datasets, which has been extended to graphs and relational data, like Graph-FCA. It encompasses various tasks such as pattern mining or machine learning, but its application generally relies on the computation of a concept lattice whose size can be exponential with the number of instances. We propose to follow an instance-based approach where the learning effort is delayed until a new instance comes in, and an inference task is set. This is the approach adopted in k-Nearest Neighbors, and this relies on a distance between instances. We define a conceptual distance based on FCA concepts, and from there the notion of concepts of neighbors, which can be used as a basis for instance-based reasoning. Those definitions are given for both classical FCA and Graph-FCA. We provide efficient algorithms for computing concepts of neighbors, and we demonstrate their inference capabilities by presenting three different applications: query relaxation, knowledge graph completion, and relation extraction.

1. Introduction

Nowadays, a lot of information is available in the form of relational data, such as relational databases [1], graph databases [2], or knowledge graphs [3]. In a relational dataset, the information is represented in terms of entities and relationships between those entities. Graphs are a common representation of relational data, using nodes for entities and edges for relationships. The different existing formalisms differ in the kind of nodes and edges they support or not: e.g., labeled edges, n-ary edges, literal values as nodes. The major advantage of relational data is their versatility and representation power. All kinds of structured data can be represented accurately as entity-relation graphs, including tabular data, XML trees, JSON objects, biological sequences or linguistic parse trees. The growing amount of relational data calls for powerful methods to analyze them and learn from them.

Formal Concept Analysis (FCA) [4,5] is a mathematical framework for the analysis of symbolic datasets. The strength of FCA is to encompass in one framework various tasks such as pattern mining, rule mining, machine learning, hierarchical clustering, and information retrieval [6]. In its simplest setting, an FCA dataset is a binary relation between objects and attributes. However, a number of extensions have been proposed to cope with more complex data and settings. Fuzzy Concept Analysis [7] adds truth degrees to the relations between objects and attributes, Temporal Concept Analysis (TCA) [8] adds a temporal dimension to the description

* Corresponding author.
E-mail addresses: ambre.ayats@irisa.fr (H.A. Ayats), peggy.cellier@irisa.fr (P. Cellier), ferre@irisa.fr (S. Ferré).

[1] The authors are listed in alphabetical order. This research is supported by ANR projects SmartFCA (ANR-21-CE23-0023) and AI4SDA (ANR-20-THIA-0018).

https://doi.org/10.1016/j.ijar.2023.109059
Received 30 January 2023; Received in revised form 4 October 2023; Accepted 6 October 2023

of objects, and Pattern Structures [9] allow for custom descriptions instead of sets of attributes. More specifically, several extensions have been proposed for relational data: *Relational Concept Analysis* (RCA) [10], concept lattices of relational structures [11], and more recently Graph-FCA [12]. A major issue in FCA is that the computation cost of the concept lattice can grow exponentially with the size of the data, and becomes even worse with relational extensions of FCA. However, depending on the task, it may not be necessary to compute all concepts and their lattice. Therefore, methods that compute only the relevant concepts w.r.t. the current task have to be developed.

The computation of the whole concept lattice can be compared to the learning of a model from training data in machine learning (*model-based learning*), where the model is then applied to new instances in order to make inferences about them. An alternative paradigm is *instance-based learning*, also known as *lazy learning*. It avoids the costly computation of a global model by delaying the learning effort until a new instance – called the *query instance* – comes in. In this way, only a specialized local model needs to be built in order to perform the desired inference on the query instance. Another advantage of this paradigm is that it supports frequent updates in the training dataset. Indeed, there is no training phase, which is usually long and energy-consuming. The most famous example of instance-based learning algorithm is the *k-Nearest-Neighbors* (kNN) algorithm [13]. It represents both training instances and query instances as points in a vector space. The class of a query instance is predicted from the class of the training instances that are the *nearest neighbors* of the query instance in the vector space. This involves the definition of a *distance* between instances that reflects their *similarity*. The research question we address in this paper is the following: What is a relevant definition of distance (or similarity) in the FCA context? and How such a distance can be used for different tasks by instance-based learning?

In this paper, we propose an instance-based learning method based on FCA that can be applied to various inference tasks. We first define a *conceptual distance* between instances in terms of formal concepts, and then the *concepts of neighbors* of a query instance as a basis for various inference tasks. Those definitions are given for classical FCA and also for the Graph-FCA extension in order to address relational data. The choice of Graph-FCA among the relational extensions is motivated by several advantages of Graph-FCA: (a) it supports n-ary relations; and (b) it can form n-ary concepts, which enables to reason on tuples of objects as instances (not only singleton objects). In addition to a theoretical framework, we provide efficient algorithms for computing concepts of neighbors. We also describe in detail three applications with different kinds of data and different kinds of inferences: approximate query answering, knowledge graph completion, and relation extraction from texts.

This paper brings together work published in several papers, both applied [14–18] and theoretical [19], with different notations. These contributions are here presented in a single formalism, giving them coherence and readability. In addition, this paper presents a novel formalization of the notion of Concepts of Neighbors in the general FCA framework, making it available to classical FCA and transferable to other FCA extensions.

The paper is organized as follows. Section 2 presents the related work. Section 3 gives the theoretical background on FCA and Graph-FCA. Section 4 defines the notions of conceptual distance and concepts of neighbors. Section 5 details the algorithms and implementation for the efficient computation of concepts of neighbors. Section 6 presents three applications of concepts of neighbors. Finally, Section 7 concludes the paper and presents some perspectives.

2. Related work

The notion of concepts of neighbors and its computation methods can be seen under three aspects. First, it is a machine learning approach on relational data for tasks such as classification. More precisely, it is an instance-based learning approach, conceptually similar to *kNN* algorithms. Second, this method is part of the FCA theory, and especially of Graph-FCA, a generalized framework for handling relational data under the FCA theory. Third, concepts of neighbors can be seen as a graph pattern mining method, searching for similarities in the form of graph patterns in graph datasets. This section presents successively existing work related to those three aspects, highlighting the main similarities and differences with concepts of neighbors.

Instance-based learning As far as we know, instance-based learning on relational data has rarely been studied. The most known approach on this subject is Relational Instance-Based Learning (RIBL) [20]: it consists in defining a numerical distance between items in a relational dataset and then applying an algorithm similar to the *kNN* algorithm to classify query instances. A similar approach has been used in [21], this time in the context of the instance-based guided edition of RDF graphs. However, as far as we know, all existing approaches use a numerical distance between objects, whereas Concepts of Neighbors use a symbolic distance based on formal concepts.

Formal concept analysis Because of its relatively low computational cost, the idea of using instance-based learning to perform classification has been considered in FCA-based machine learning [22]: instead of computing the whole concept lattice of a context, it consists into computing only the concepts related to the object to be classified. Therefore, for a single classification, the number of concepts to compute is reduced from exponential to linear. This idea has been applied to relation classification in biomedical texts [23]. This principle has been extended with other techniques such as approximation, random sampling and parallelization to be applied to big data [24,25]. On a different task, information retrieval, the user query is considered as the query instance, and a notion of *cousin concepts* enables to find approximate answers to the user query and to rank them by increasing distance [26]. However, the cousin concepts are found by navigating the concept lattice, which therefore has to be computed beforehand. Our concepts of neighbors can be seen as a generalization of cousin concepts, and we provide a lazy algorithm for their computation. Approximate query answering is one of the applications of concepts of neighbors (see Section 6.1).

Table 1
Example of a formal context where the set of objects is an excerpt of the British royal family.

	man	woman	adult	kid	married
Charles	×		×		×
Charlotte		×		×	
Diana		×	×		×
George	×			×	
Harry	×		×		×
Kate		×	×		×
William	×		×		×

Graph pattern mining The domain of graph pattern mining has been largely explored these last decades, and several types of approaches can be distinguished. The most classic ones, such as AGM [27], FFSM [28] or gSpan [29], are complete approaches, mining all patterns over a given frequency. However, these methods encounter the pattern explosion problem: either the frequency threshold is too high and the approaches return almost no pattern, or the frequency threshold is too low, and an intractable number of patterns (millions or more) is returned. This problem led to more parsimonious graph mining approaches. Some of them such as Gprune [30] rely on the user for specifying constraints on the patterns, others choose to select specific subsets of the frequent patterns, such as SPIN [28] for the maximal patterns and CloseGraph [31] for the closed patterns. More recently, ideas from information theory have been used to drastically reduce the amount of mined patterns: e.g., the *Minimum Description Length (MDL) principle* (e.g., Graph-MDL+ [32]) or the maximum entropy [33]. Concerning knowledge graph mining, specific approaches have been developed, such as SWApriori [34] for complete pattern mining, or rule mining approaches such as AMIE 3 [35] that aims to generate useful rules for tasks such as knowledge graph completion. Unlike all above approaches that generate a global set of patterns, concepts of neighbors are a local set of patterns that are relevant to a query instance. Moreover, the generated patterns are *rooted patterns*, i.e. they have a distinguished node that corresponds to the query instance. All this make them fit for downstream inference tasks.

3. Preliminary definitions

In this section, we give the preliminary definitions used in the rest of the paper. We first present the main notions of Formal Concept Analysis (FCA). Then the adaptation of those notions in the framework of Graph-FCA, an extension of FCA for multi-relational data, is given as defined in [36].

In the following, for any set X, we note $X^* = \bigcup_{i=0}^{\infty} X^i$ the set of tuples of elements in X of any length.

3.1. Formal concept analysis (FCA)

In this section, we present the basic notions of FCA [5].

Definition 1. A **formal 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 objects and attributes. For each object $o \in O$, we define $I(o) = \{a \in A \mid (o, a) \in I\}$ as the *description* of o.

Table 1 gives an example of a formal context. The set of objects is an excerpt of the British royal family and the attributes are some human characteristics, here to be a man, a woman, an adult, a kid, and married. The incidence relation associates each person to his/her characteristics. For instance, in this context, Charles is a man, an adult and married.

Definition 2. We define the **instances** of a set of attributes as the function

$$\begin{aligned} int :\ & \mathcal{P}(A) \to \mathcal{P}(O) \\ & Y \mapsto \{o \in O \mid Y \subseteq I(o)\} \end{aligned}$$

that maps a set of attributes to the set of the objects that have all the attributes in their description.

Definition 3. We define the **properties** of a set of objects as the function

$$\begin{aligned} prop :\ & \mathcal{P}(O) \to \mathcal{P}(A) \\ & X \mapsto \bigcap_{o \in X} I(o) \end{aligned}$$

that maps a set of objects to the set of their common attributes.

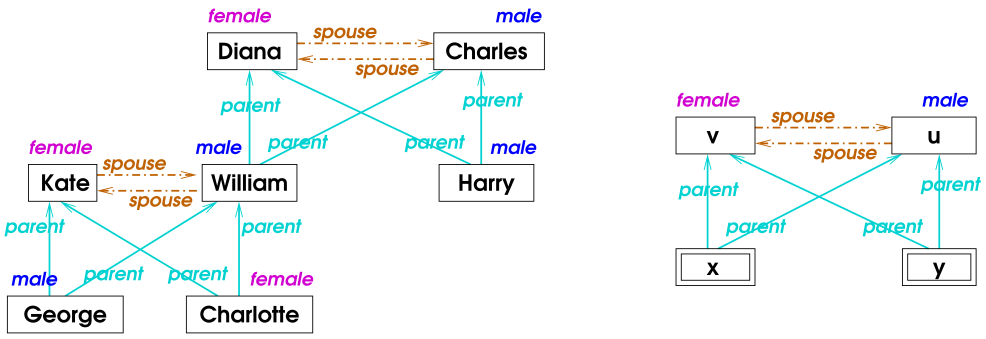

Fig. 1. (a) Graph context representing the British royal family. (b) PGP representing the sibling relationship between x and y.

For instance, in Table 1, $inst(\{woman, adult\}) = \{Diana, Kate\}$ and $prop(\{Charles, William\}) = \{man, married, adult\}$. The pair of functions $(inst, prop)$ forms a Galois connection between sets of objects and sets of attributes, which leads to the definition of formal concepts.

Definition 4. Let $K = (O, A, I)$ be a formal context, $X \subseteq O$ a set of objects, and $Y \subseteq A$ a set of attributes. The pair (X, Y) is a **formal concept** if and only if $inst(Y) = X$ and $prop(X) = Y$. X is called the *extension* of the concept and Y the *intension*.

For example, in Table 1, $(\{Charles, William\}, \{man, married\})$ is not a concept whereas $(\{Charles, William, Harry\}, \{man, married\})$ is a concept.

The main theorem of FCA says that the set of all concepts of a context, when partially ordered by

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

forms a complete lattice. This implies that for every pair of concepts C_1, C_2, there is an infimum concept $C_1 \wedge C_2$ (concept intersection) and a supremum concept $C_1 \vee C_2$ (concept union). There are also a most general concept $\top$ (top concept) and a most specific concept $\bot$ (bottom concept).

3.2. Graph-FCA

Graph-FCA [36,37] is an extension of FCA for knowledge graphs, and more generally for relational data. Indeed, in FCA, objects can be described individually but not the relationships between objects. The equivalent of a formal context for Graph-FCA is called a *graph context*.

Definition 5. 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.

Fig. 1 (a) is a graphical representation of a graph context. It represents an excerpt of the British royal family. The boxes are the objects (e.g. *Charlotte* or *Harry*), the labels next to boxes are unary attributes (e.g. *man*, *woman*), and the arrow labels are binary attributes (e.g. *parent* or *spouse*). It is also possible to have n-ary attributes with $n > 2$ in Graph-FCA, although they are not used in what follows. The unary incidence $((Kate), woman) \in I$ says that Kate is a woman; we call it a node label and also write it $woman(Kate)$ for readability and by analogy to predicate logic. The binary incidence $((George, Kate), parent) \in I$ says that Kate is a parent of George; we call it a labeled edge and also write it $parent(George, Kate)$.

Most models of knowledge graphs can be translated accurately to graph contexts: e.g., the RDF graphs of the Semantic Web [38], conceptual graphs [39], RCA contexts [10]. For simple entity-relation KGs, entities translate to objects, relations to binary attributes, and triples to binary incidences. For translating RDF graphs to graph contexts, we abstract each URI and blank node x into an object o_x, and we add the following incidences to the context:

- incidence $c(o_x)$ for each triple $(x, \texttt{rdf:type}, c)$,
- incidence $p(o_x, o_y)$ for each triple (x, p, y) where y is not a literal,
- incidence $p(o_x, o_l)$ for each triple (x, p, l) where l is a literal and o_l is an object representing its occurrence in this specific triple,
- incidence $u(o_u)$ for each URI u,
- incidence $l(o_l)$ for each occurrence o_l of a literal l.

The sets of objects and attributes can be deduced from this set of incidences. In particular, the set of attributes is made of classes as unary attributes, properties as binary attributes, and URIs and literals as unary attributes. In this representation, each RDF node is represented by an object labeled by its URI or literal – or unlabeled in the case of a blank node. More details can be found in [37].

The purpose of FCA is to discover *patterns* shared by objects. Whereas in classical FCA a pattern is a subset of attributes, in Graph-FCA a pattern is a *projected graph pattern*, which expresses a common graph structure rooted in one or several nodes.

Definition 6. Let $\mathcal{V}$ be an infinite set of variables. A **graph pattern** $P \subseteq \mathcal{V}^* \times A$ is a set of pairs $(\overline{y}, a)$, with $\overline{y}$ a tuple of variables and a an attribute. Each of those pairs can be seen as a n-ary directed incidence (with $n = |\overline{y}|$) labeled by attribute a.

A **projected graph pattern (PGP)** is a pair $Q = (\overline{x}, P)$ where $\overline{x} \in \mathcal{V}^*$ a tuple of variables – called **projected variables** – and P is a graph pattern, such that each incidence of P is transitively connected to at least one element of $\overline{x}$. The **arity** of a PGP is the length of $\overline{x}$. A PGP of arity k is also called a k-PGP. The set of PGPs of arity k over a set of attributes A is denoted by $PGP_k(A)$.

Compared to previously published work, we here make it explicit that PGPs should be connected. Indeed, disconnected incidences would not contribute to PGPs' meaning. This has no incidence on Graph-FCA results.

Fig. 1 (b) represents a 2-PGP defining the "sibling" binary relation, i.e. two persons with the same parents. Projected variables are in double boxes. In practice, PGPs can be seen as queries on the graph context, and we reuse the notation of such queries for the textual representation of PGPs:

$$\begin{aligned}[x, y \leftarrow\ & man(u), woman(v), parent(x, u), parent(x, v),\\ & parent(y, u), parent(y, v), spouse(u, v), spouse(v, u)]\end{aligned}$$

For any set of attributes A and arity k, the set of k-PGPs $PGP_k(A)$ forms a bounded lattice, with a partial ordering $\subseteq_{PGP}$ and an infimum $\cap_{PGP}$: $Q_1 \subseteq_{PGP} Q_2$ means that Q_1 is a more general query than Q_2; and $Q_1 \cap_{PGP} Q_2$ is the least general generalization of the two queries. Besides, we define the *description* of the tuple of objects $\overline{o}$ as the PGP $Q(\overline{o}) = (\overline{o}, P(\overline{o}))$, where $P(\overline{o}) \subseteq I$ is the subset of incidences that are transitively connected to any element of $\overline{o}$. It is the union of the connected components of the graph context containing the elements of $\overline{o}$, rooted in those elements. It plays the same role as $I(o)$ in FCA.

Definition 7. We define the set of **answers** of a PGP Q as the set of answers of Q seen as a query.

$$\begin{aligned} ans:\ & PGP_k(A) \to \mathcal{P}(O^k)\\ & Q \mapsto \{\overline{o} \mid Q \subseteq_{PGP} Q(\overline{o})\}\end{aligned}$$

In the example PGP of Fig. 1 (b), the set of answers over the example graph context is made of the pairs $(Harry, William)$ and $(George, Charlotte)$, and also their inverse because of the pattern symmetry, and also the identity pairs like $(Harry, Harry)$ and $(Charlotte, Charlotte)$ because there is no inequality constraints between variables x and y. The answers of $Q(\overline{o})$ are the set of all tuples of objects that match everything that is known about $\overline{o}$.

Definition 8. We define the **most specific query** of a set of k-tuples of objects as the largest k-PGP according to $\subseteq_{PGP}$ whose set of answers contains R.

$$\begin{aligned} msq:\ & \mathcal{P}(O^k) \to PGP_k(A)\\ & R \mapsto \bigcap_{\overline{o} \in R} Q(\overline{o})\end{aligned}$$

In the example graph context, the most specific query of Charlotte and Kate is that they are women: $msq(\{Charlotte, Kate\}) = [x \leftarrow woman(x)]$. The most specific query of Charlotte and William is $[x \leftarrow P_{sibling}, man(y)]$, where $P_{sibling}$ is the pattern of the above example PGP. It says that Charlotte and William have in common married parents and a brother. As proven in [37], the pair of functions (ans, msq) forms a **Galois connection** between $\mathcal{P}(O^k)$ and $PGP_k(A)$, for any k. This leads to the definition of graph concepts.

Definition 9. Let $K = (O, A, I)$ be a graph context. A **graph concept** of arity k (also called a k-concept) is a pair $C = (R, Q) \in \mathcal{P}(O^k) \times PGP_k(A)$ such that $R = ans(Q)$ and $Q = msq(R)$. R is called the **extension** of the concept, and Q is called the **intension**.

For each arity k, the set of all k-concepts forms a complete lattice where two concepts are ordered, $(R_1, Q_1) \leq (R_2, Q_2)$, iff $R_1 \subseteq R_2$, and iff $Q_2 \subseteq_{PGP} Q_1$. FCA is the special case of Graph-FCA where only unary attributes are used in the graph context, and only 1-concepts are considered. A more extensive presentation of Graph-FCA can be found in the reference journal paper [16]. An implementation[2] is available for the computation of graph concepts and their visualization [40].

4. Concepts of neighbors

Concepts of neighbors define a distance/similarity scheme in terms of FCA concepts, in a literal way because concepts are used directly as a measure of the distance/similarity between two objects. This is in contrast with the usual definitions of distance or

[2] https://bitbucket.org/sebferre/graph-fca/.

similarity that use numerical values, even when applied to discrete structures. Concepts of neighbors also offer a local instance-based view of concepts – with FCA objects playing the role of instances – compared to the global view of concept lattices that is common with FCA.

In this section, we first define concepts of neighbors on standard FCA in order to explain the key ideas on a simple and well known formalism (Section 4.1). We then generalize the definitions to the richer Graph-FCA case in order to accommodate more complex relational data.

4.1. The FCA case

We first define the *conceptual distance* between two objects as the most specific concept that contains both of them.

Definition 10. Let $K = (O, A, I)$ be a formal context, and $u, v \in O$ be two objects in this context. The **conceptual distance** between objects u and v is the concept $\delta(u, v) = (X, Y)$ s.t. $Y = prop(\{u, v\}) = I(u) \cap I(v)$, and $X = inst(Y)$.

Intuitively, the more similar the two objects are, the more specific the conceptual distance is. A more specific concept has fewer objects and more attributes. Having more attributes means having more similarities. The objects in the concept extent can be seen as in between the two objects, and hence fewer objects means a smaller distance. For example, in the formal context presented in Section 3.1, the conceptual distance between Charles and Charlotte has an empty intension and an extension containing all the objects, while the conceptual distance between Charles and Harry has for intension $\{man, adult, married\}$ and for extension $\{Charles, Harry, William\}$. It can be seen that the conceptual distance between Charles and Harry has three attributes in its intension while the conceptual distance between Charles and Charlotte has none. Therefore, Charles is more similar to Harry than to Charlotte. Reciprocally, the conceptual distance between Charles and Charlotte has seven objects in its extension, while the conceptual distance between Charles and Harry has only three objects. Therefore, Charles is more distant to Charlotte than to Harry.

The duality between distance and similarity is here embodied in the duality between the extension and intension of a concept. Actually, the conceptual distance between two objects is at the same time their *conceptual similarity*. The above definition satisfies the properties of a distance if we take the partial order $\leq$ on concepts as distance order, and concept supremum $\vee$ as addition: i.e., for all objects $u, v, w \in O$,

1. (positivity) $\delta(u, u) \leq \delta(u, v)$, ($\delta(u, u)$ represents distance zero)
2. (symmetry) $\delta(u, v) = \delta(v, u)$,
3. (triangular inequality) $\delta(u, v) \leq \delta(u, w) \vee \delta(w, v)$.

Because of the ordering being partial, it is common to have objects, say v and w, that are at incomparable distances from u: i.e., $\delta(u, v) \nleq \delta(u, w)$ and $\delta(u, w) \nleq \delta(u, v)$.

Numerical versions of distance and similarity can be derived from the conceptual distance by using the cardinal of the extension or intension. Given the conceptual distance $\delta(u, v) = (X, Y)$:

- the **extensional distance** $d(u, v) = |X|$ is the cardinal of the extension,
- the **intensional similarity** $sim(u, v) = |Y|$ is the cardinal of the intension.

Note that those numerical versions are degraded versions, as they flatten the partial ordering over conceptual distances to a total ordering. This can lead to consider two objects as being at the same distance, whereas the conceptual distances are completely different.

Definition 11. Let $K = (O, A, I)$ be a formal context, and $u \in O$ be an object. The **concepts of neighbors** of u is the set of all conceptual distances from u to any other object in the context: $C\text{–}N(u) = \{\delta(u, v) \mid v \in O\}$.

For example, in the formal context defined in 3.1, $C\text{–}N(Charlotte)$ contains four concepts. The first one has for intension $\{woman, kid\}$ and for extension $\{Charlotte\}$. Then there are two more general concepts: $(\{Charlotte, Diana, Kate\}, \{woman\})$ and $(\{Charlotte, George\}, \{kid\})$. Finally, there is the most general concept, with an empty intension and the whole set of objects as extension.

Concepts of Neighbors $C\text{–}N(u)$ provide an instance-based view over the context, by partitioning the set of objects according to their conceptual distance with u. Each concept of neighbors $\delta \in C\text{–}N(u)$ induces the subset of objects $\delta.proper := \{o \in O \mid \delta(u, o) = \delta\}$, i.e. the objects that are exactly at distance δ from u. This is a subset of the extension of δ, called the **proper extension** of δ.

4.2. The graph-FCA case

Concepts of neighbors are naturally extended to Graph-FCA, as the latter inherits the definitions and theorems of FCA. Conceptual distances and concepts of neighbors are here graph concepts. A first benefit in Graph-FCA is that the similarity between two objects is not only expressed in terms of common properties but also in terms of common relationships to similar objects, which in turn can have common properties and common relationships to farther objects, and so on. For instance, the

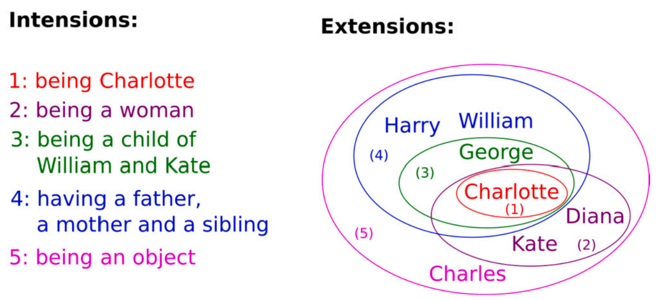

Fig. 2. Concepts of Neighbors of Charlotte. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

conceptual distance between France and Spain could be the concept of countries that speak a Romance language, with PGP $[x \leftarrow country(x), speaks(x, y), language(y), romanceLanguage(y)]$, with Italy and Portugal in the extension. A second benefit is that there are not only concepts for individual objects but also for k-tuples of objects. This implies that it becomes possible to compare such tuples of objects. For instance, the conceptual distance between pairs $(France, Paris)$ and $(Ukraine, Kyiv)$ is the concept whose intent states the "has capital" relationship among other things, with PGP $[x, y \leftarrow hasCapital(x, y), country(x), city(y)...]$, and whose extent contains more $(country, city)$ pairs, e.g. $(Italy, Roma)$.

Definition 12. Let $K = (O, A, I)$ be a graph context, and $\overline{u}, \overline{v} \in O^k$, for some arity k. The **conceptual distance** between the two k-tuples of objects $\overline{u}$ and $\overline{v}$ is the k-concept $\delta(\overline{u}, \overline{v}) = (R, Q)$ s.t. $Q = msq(\{\overline{u}, \overline{v}) = Q(\overline{u}) \cap_{PGP} Q(\overline{v})$, and $R = ans(Q)$.

Like in FCA, this definition satisfies the properties of a distance, and numerical versions of distance and similarity can be derived likewise.

Definition 13. Let $K = (O, A, I)$ be a graph context, and $\overline{u} \in O^k$ be a k-tuple of objects, for some arity k. The **concepts of neighbors** of $\overline{u}$ is the set of all conceptual distances from $\overline{u}$ to any other k-tuple in the context: $C\text{–}N(\overline{u}) = \{\delta(\overline{u}, \overline{v}) \mid \overline{v} \in O^k\}$.

This again induces a partition over the set of k-tuples of objects, where each part is the proper extension $\delta.proper$ of the corresponding conceptual distance δ.

Fig. 2 presents the five concepts of neighbors of Charlotte in the graph context presented in Fig. 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 Charlotte in its extension. Then there are two larger concepts, one containing the children of Kate and William (Charlotte and George), and another one containing the women. Then there is an even larger concept containing 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 all objects as extension. For each concept of neighbors, the objects of the proper extension are displayed in the same color as the concept ellipse. It can be seen that the proper extension of a concept is made of the objects in the extension that are not in sub-concepts. This color-coding materializes the partition of objects by concepts of neighbors. When two objects are in the same proper extension, e.g. Harry and William, this means that they are indistinguishable relative to the chosen object. The Venn diagram shows well the fact that the concepts of neighbors are only partially ordered. For instance, Concept (3) is a sub-concept of Concept (4) but is incomparable with Concept (2). The inclusion of (3) in (4) means that George from Concept (3) is closer to Charlotte than Harry from Concept (4) is: he has a father, a mother and a sibling, like Harry, but he also has the same parents as Charlotte, unlike Harry. The incomparability of (3) and (2) means that George and Diana are similar to Charlotte for incomparable reasons: George has the same parents as Charlotte, while Diana has the same gender as Charlotte. Such distinctions cannot be made with totally ordered distances like numerical ones.

5. Algorithms

We here describe an efficient and anytime algorithm for the computation of concepts of neighbors. The algorithm inputs are a graph context $K = (O, A, I)$, and a k-tuple of objects $\overline{u} \in O^k$, called the *query instance*. The algorithm output is the collection of concepts of neighbors $C\text{–}N(\overline{u})$, with for each conceptual distance $\delta \in C\text{–}N(\overline{u})$ the concept intension $\delta.int$, the concept extension $\delta.ext$, and the proper extension $\delta.proper$. The three types of information play an important role in applications of Concepts of Neighbors (see Section 6). For practical reasons, the algorithm can also take as input a subset of the k-tuples of objects $V \subseteq O^k$, called *candidate instances*, to restrict the set of considered neighbors of the query instance. The extensions and proper extensions of concepts of neighbors are therefore subsets of V.

A first approach to compute concepts of neighbors would be to follow their definition, i.e. to compute the conceptual distance from the query instance $\overline{u}$ to every candidate instance $\overline{v} \in V$. This is the approach followed by work using distance measures on symbolic data [20]. It here requires performing two complex operations for each candidate instance: to compute the most specific query Q between them, and to compute the set of answers R of that most specific query Q. This does not scale to large sets of candidate instances. This is also inefficient because in general each concept of neighbor will be computed many times, $|\delta.proper|$ times to be exact. A second approach would be to start with the concept $\delta(u, u)$ whose intension is the graph description $Q(\overline{u})$ of the query instance, and to explore the space of more general concepts by generalizing the intension. This is the approach followed by query relaxation [14]. It here requires to repeatedly apply minimal generalizations to PGPs, e.g. removing an edge, and to compute the set of answers of the generalized PGPs. If the generalized PGP covers additional instances, then a new conceptual distance has been found, and the additional instances make for its proper extension. The first problem is that the generalization space is combinatorial in the size of the initial PGP, and in the case of knowledge graphs, the initial PGP often contains the whole graph context. Moreover, the set of answers has to be computed for each generalized PGP, and it cannot be derived from previous PGPs as their extension is smaller.

We propose a third approach to compute concepts of neighbors, an approach that factorizes the computation of the most specific queries across the instances, and that factorizes the computation of sets of answers across generalized PGPs. The general idea is to explore the generalization space top-down, starting with the empty PGP, and to massively prune it by maintaining and refining a partition of the instances that converges towards the proper extensions. By exploring the generalization space top-down, i.e. by specializing PGPs, the answers of a specialized PGP Q_2 can be computed incrementally from the answers of its parent PGP Q_1. Indeed, the answers of Q_2 are a subset of the answers of Q_1 because of the Galois connection between PGPs and sets of object tuples. Moreover, for every concept of neighbors, there is a single path in the top-down exploration that leads to its intension, so that the most specific queries are computed only once, whatever the size of the proper extension.

In this section, we first describe the partitioning algorithm that explores the generalization space top-down to compute concepts of neighbors (Section 5.1). We then introduce an essential optimization in the computation of sets of answers by introducing the *lazy join* algorithm (Section 5.2). Those two algorithms have been previously published in [14], in the context of query relaxation. We also present CONNOR, an implementation as a Java library (Section 5.3).

5.1. Iterative partitioning of instances into concepts of neighbors

At any stage of the algorithm, the set of candidate instances is partitioned in a set of *pre-concepts of neighbors* (pre-concepts in short), where each pre-concept is made of a PGP Q_l and its answers R_l like a concept, except that the PGP is not necessarily the most specific query for those answers. Each pre-concept also comes with a subset of answers V_l such that the collection $\{V_l\}_l$ defines a partition of the set of candidate instances, a *match-set* M_l that is the set of answers extended to all variables occurring in the PGP graph pattern, and a set of incidences I_l to be used as PGP specializations. The match-set is useful for the incremental computation of sets of answers of specialized PGPs, and the incidences are useful to control the specialization of PGPs. We define match-sets before formally defining pre-concepts.

Definition 14. Let $K = (O, A, I)$ be a graph context, and $\mathcal{V}$ an infinite set of variables. A **match-set** is a pair $M = (\overline{x}, R) \in \mathcal{V}^k \times \mathcal{P}(O^k)$, for some arity k. It defines a set of mappings from the k variables in $\overline{x}$ to objects of the context: $\overline{x} = dom(M)$ is called the **domain** of the match-set, and $R = rel(M)$ is called the **relation** of the match-set. Match-sets are equipped with two operations from relational algebra [1]:

- the **projection** $\pi_{\overline{y}}\ M$ of a match-set on a sub-tuple $\overline{y}$ of the match-set variables;
- the **(natural) join** $M_1 \bowtie M_2$ of two match-sets.

Definition 15. Let K be a graph concept, $\overline{u}$ be the query instance with arity k, and $V \subseteq O^k$ be the set of candidate instances. A *partition* of the set of candidate instances is a collection $\{C_l\}_l$ of *pre-concepts (of neighbors)*, where each pre-concept is a structure $C_l = (Q_l, R_l, V_l, M_l, H_l)$, s.t.:

- $Q_l = [\overline{u} \leftarrow P_l]$ is a k-PGP that is a generalization of $Q(\overline{u})$, with $\overline{x}_l$ being the tuple of all variables occurring in $\overline{u}$ or in P_l;
- $R_l = ans(Q_l) \cap V$ is the set of answers of Q_l in V;
- $V_l \subseteq R_l$ is a subset of answers such that the collection $\{V_l\}_l$ forms a partition of V;
- $M_l = (\overline{x}_l, ans((\overline{x}_l, P_l)))$ is the *match-set* containing all matchings of the pattern P_l on the graph context;
- $I_l \subseteq I$ is a set of incidences from the description of the query instance that remain available for specializing Q_l.

As we explore the generalization space of the description of $\overline{u}$, we simply use objects from that description as variables in the PGPs. A pre-concept becomes a concept of neighbors when I_l gets empty, i.e. when the pre-concept cannot be specialized further. In this case, the pair (Q_l, R_l) is a concept of neighbors (filtered by V), and V_l is its proper extension. When I_l is not empty, V_l may be a union of proper extensions (lack of discrimination), and Q_l is not necessarily the most specific query of instances in V_l (lack of precision in the conceptual similarity). This implies an overestimate of the distance, and an underestimate of the similarity, for some instances in V_l. The specialization process aims at making pre-concepts converge to concepts of neighbors, going through increasingly precise estimates of distance and similarity.

Initially, there is a single pre-concept, the *initial pre-concept* that uses the empty PGP and that contains all candidate instances.

Definition 16. The *initial pre-concept* is the pre-concept C_{init} s.t.:

Algorithm 1 *Partition*$(K, \overline{u}, V)$.

```
Require: A graph context $K = (O, A, I)$
Require: An arity $k > 0$, a query instance $\overline{u} \in O^k$, and candidate instances $V \subseteq O^k$
Require: An optional timeout
Ensure: A collection of pre-concepts $C$ partitioning $V$ w.r.t conceptual distance to $\overline{u}$
 1: $C \leftarrow \{C_{init}\}$
 2: while no timeout and there is a pre-concept $C_l \in C$ such that $I_l \neq \emptyset$ do
 3:     pick some $(\overline{w}, a) \in I_l$ that is connected to $\overline{u}$ in $Q_l$
 4:     $Q_i \leftarrow [\overline{u} \leftarrow P_l \cup \{(\overline{w}, a)\}]$
 5:     $M_i \leftarrow M_l \bowtie M_{(\overline{w},a)}$
 6:     $R_i \leftarrow rel(\pi_{\overline{u}}\, M_i)$
 7:     $V_i \leftarrow V_l \cap R_i$
 8:     $V_j \leftarrow V_l \setminus V_i$
 9:     $I_{ij} = I_l \setminus \{(\overline{w}, a)\}$
10:     $C \leftarrow C \setminus \{C_l\}$
11:     $C \leftarrow C \cup \{C_i\}$, if $V_i \neq \emptyset$, where $C_i = (Q_i, R_i, V_i, M_i, I_{ij})$
12:     $C \leftarrow C \cup \{C_j\}$, if $V_j \neq \emptyset$, where $C_j = (Q_l, R_l, V_j, M_l, I_{ij})$
13: end while
```

- $Q_{init} = [\overline{u} \leftarrow \emptyset]$ is the empty PGP ($\overline{x}_{init} = \overline{u}$);
- $R_{init} = ans(Q_{init}) \cap V = V$ is the set of all candidate instances;
- $V_{init} = V$ is the set of all candidate instances;
- $M_{init} = (\overline{u}, V)$;
- $I_{init} = I$ is the set of all incidences from the graph context.

Each pre-concept C_l s.t. $I_l \neq \emptyset$ may be split in two pre-concepts C_i and C_j by using an incidence $(\overline{w}, a) \in I_l$ to discriminate among instances V_l those that match the incidence from those that do not.

Definition 17. The specialization of a pre-concept C_l by an incidence $(\overline{w}, a) \in I_l$ leads to two new pre-concepts C_i and C_j that replace C_l in the partition, and that are defined as follows (the definitions of R and M follow from the definition of Q):

$$\begin{aligned} Q_i &= [\overline{u} \leftarrow P_l \cup \{(\overline{w}, a)\}] & Q_j &= Q_l \\ V_i &= V_l \cap R_i & V_j &= V_l \setminus R_i = V_l \setminus V_i \\ I_i &= I_l \setminus \{(\overline{w}, a)\} & I_j &= I_l \setminus \{(\overline{w}, a)\} \end{aligned}$$

Pre-concept C_i is a specialization of C_l, by the addition of an incidence to the query pattern. This leads in general to a smaller set of answers R_i, and hence a smaller subset of candidate instances V_i. Pre-concept C_j is an update of concept C_l, where the candidate instances V_j is the part of V_l that is not selected by V_i. In both pre-concepts, the chosen incidence is discarded so that it is not considered any more for the specialization of the new pre-concepts. After a specialization, we still have a partition of V because $\{V_i, V_j\}$ is a partition of V_l. In general, we get a finer partition with two smaller parts, but it is also possible that one of the parts is empty, in which case the partition is unchanged. For instance, if the chosen incidence is disconnected from $\overline{u}$ in Q_l, then it has no effect on answers and $V_i = V_l$. Therefore, only connected incidences should be chosen for effective specialization. However, even connected incidences can lead to $V_i = V_l$, depending on regularities in data.

According to the above definition, the cost of specializing a pre-concept looks very small. It amounts to add an incidence to a PGP, to perform basic set operations on sets of instances, and to remove an element from the set of incidences. However, the computation of V_i and V_j requires the set of answers $R_i = ans(Q_i)$ of the specialized PGP. This computation can be made incremental by relying on the match-set of pre-concepts. The match-set M of a graph pattern P is equal to the join of the match-sets of all incidences $(\overline{w}, a) \in P$:

$$M = \bowtie_{(\overline{w},a)\in P} M_{(\overline{w},a)}$$
$$\text{where } M_{(\overline{w},a)} = (\overline{w}, ans([\overline{w} \leftarrow a(\overline{w})]))$$

As the join operator is associative and commutative, the match-set of the specialized PGP can be computed incrementally from the parent PGP.

$$M_i = M_l \bowtie M_{(\overline{w},a)}$$

Finally, the set of answers is simply the projection of the match-set on the projected variables.

$$R_i = rel(\pi_{\overline{u}}\, M_i)$$

Algorithm 1 details the partitioning algorithm. Given a graph context, a query instance, and a set of candidate instances, it starts with a single pre-concept, the initial pre-concept (Definition 16). It then iteratively applies specialization steps (Definition 17) to pre-concepts in order to refine the partition. The process runs until no specialization is possible or a timeout has been attained. This timeout is optional, but it has the advantage to make the algorithm anytime (more on this below). Fig. 3 shows an execution trace as

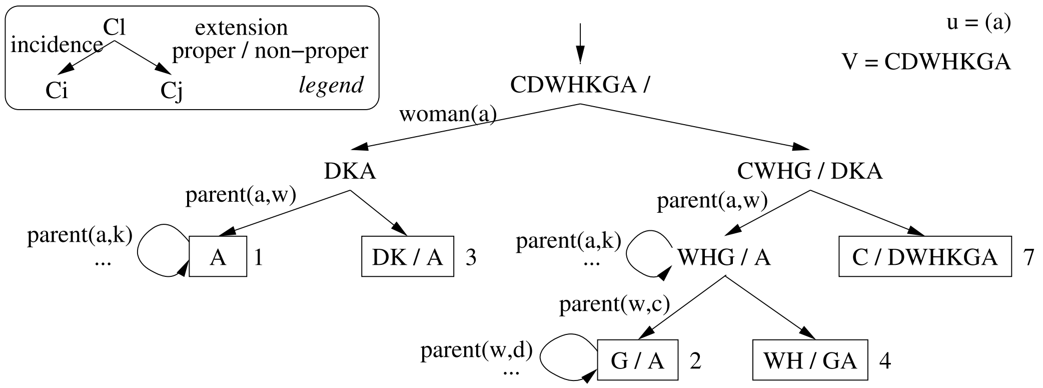

Fig. 3. Trace of the partition algorithm applied on the graph context in Fig. 1 with $\overline{u} = Charlotte$ and $V = O$. Pre-concepts are shown via their extension, with the proper extension on the left and the remainder on the right, if any. Objects are abbreviated by their initial, except for Charlotte (A). The same abbreviations in lowercase are used as variables in the incidences used for pre-concept specialization. Boxed pre-concepts at the leaves are the concepts of neighbors, and the number at their right is the extensional numerical distance.

a binary tree of pre-concepts. It represents the computation of the concepts of neighbors of Charlotte (see Fig. 2): $\overline{u} = Charlotte$ and $V = O$. The initial pre-concept contains the seven persons in the context, from Charles (C) to Charlotte (A). The first specialization uses the incidence $woman(Charlotte)$, separating the women (Diana, Kate and Charlotte) on the left from the men on the right. Those women remain in the extension of the concept on the right but no more in the proper extent. The concept on the left is further split in two pre-concepts: woman with a parent on the left (A), and woman without a parent on the right (DK). Pre-concept (A) cannot be split further, although incidences can still be added to it. From pre-concept (DK), all remaining incidences lead to an empty extension because Diana and Kate have nothing else in common with Charlotte. The boxed pre-concepts at the leaves are the results of the algorithm. Their intent is the set of incidences that label the path from the root to this pre-concept. They coincide with the concepts of neighbors shown in Fig. 2.

Discussion The algorithm may be incomplete in the sense that some concepts of neighbors may be missed, hence moving some instances at a greater conceptual distance than they are actually. The cause is that the generated graph patterns P_l are constrained to be subsets of I, they cannot be arbitrary graph patterns that have a matching occurrence in I. Suppose u is the query instance, and that I contains $\{p(u,w), a(w), b(w)\}$. The algorithm can generate the PGPs $Q_{ab} = [u \leftarrow p(u,w), a(w), b(w)]$ ("has a p that is a a and a b"), $Q_a = [u \leftarrow p(u,w), a(w)]$ and $Q_b = [u \leftarrow p(u,w), b(w)]$, but not the PGP $Q_* = [u \leftarrow p(u,w_1), a(w_1), p(u,w_2), b(w_2)]$ ("has a p that is a a, and has a p – the same or another – that is a b"). Note that Q_* is more general than Q_{ab} and more specific than Q_a and Q_b. If some candidate instance v matches Q_* but not Q_{ab}, then it will belong to the concept of either Q_a or Q_b, depending on which of $a(w)$ or $b(w)$ is chosen first. In order to recover completeness, we should allow for the duplication of a variable in a graph pattern (e.g., $w \rightsquigarrow w_1, w_2$), which would actually make the search space infinite. This simplification only entails an approximation in the computation of the conceptual distances, and it brings not only a benefit in terms of tractability, but also a benefit in terms of interpretability because the intensions of concepts of neighbors are subgraphs of the description of the query instance.

The partitioning algorithm always terminates because the set I_l decreases at each split in both concepts C_i and C_j. Moreover, the number of pre-concepts is bounded by the number of candidate instances at any step because every candidate instance belongs to the proper extension of a single pre-concept, by construction. This is remarkable because the search space is exponential in the number of incidences. Each candidate instance $\overline{v} \in V$ is forced to move down along only one path, so that among the many paths that goes from Q_{init} to $\delta(\overline{u}, \overline{v})$, only one path is actually followed. Despite those good properties, the full partitioning can still take a lot of time in the case of large graph contexts. This is why we make our algorithm anytime by adding a timeout parameter. This is justified because a valid partition of candidate instances by a collection of pre-concepts is available at all time. If the algorithm is stopped before its completion, one simply gets a coarser partition, and an overestimate of the conceptual distances to each instance. Previous experiments [14] have shown that the algorithm has the good property to output more than half of the concepts in a small proportion of the total runtime.

5.2. *Lazy join of match-sets*

The partitioning algorithm of the previous section has good properties w.r.t. the exploration of the search space, but it hides a bottleneck in the computation of match-sets. Suppose the current pre-concept C_0 where $Q_0 = [u \leftarrow film(u)]$ ("is a film"), and where the match-set M_0 has a matching for each unique film. Specializing C_0 by adding an actor to the film (incidence $actor(u, w_1)$) leads to the PGP $Q_1 = [u \leftarrow film(u), actor(u, w_1)]$ and to the match-set $M_1 = M_0 \bowtie M_{actor(u,w_1)}$, which has a matching for each (film, actor) pair. The bottleneck comes when adding an incidence $actor(u, w_i)$ for all actors of the query instance, as this entails the successive joins $M_0 \bowtie M_{actor(u,w_1)} \dots \bowtie M_{actor(u,w_n)}$. Assuming there are N films and n actors per film, each join results in a n-fold increase of the match-set, and the join chain results in a n^n-fold increase. For 1000 films related to 10 actors each, it amounts to 10^{13} matchings in

Algorithm 2 $LazyJoin(T, i, i^*, D^*, M^*, \Delta^*)$.

```
Require: a match-tree $T$, a current incidence $i$ in $T$ labeled with $(D, M, \Delta)$,
    and a new incidence $i^*$ labeled with $(D^*, M^*, \Delta^*)$
Ensure: two sets of variables $\Delta^+, \Delta^-$
 1: $\Delta^+ \leftarrow \emptyset$;   $\Delta^- \leftarrow \emptyset$
 2: for all $i_c \in children(i)$, labeled with $(D_c, M_c, \Delta_c)$ do
 3:     $\Delta_c^+, \Delta_c^- \leftarrow LazyJoin(T, i_c, i^*, D^*, M^*, \Delta^*)$  // recursive call on each child node
 4:     $\Delta^+ \leftarrow \Delta^+ \cup \Delta_c^+$;   $\Delta^- \leftarrow \Delta^- \cup \Delta_c^-$
 5:     $M \leftarrow M \bowtie \pi_{\Delta_c} M_c$, if $\Delta_c$ or $M_c$ was modified  // update of local join
 6: end for
 7: if $D \cap \Delta^* \neq \emptyset$ then  // if this node defines a variable of the new element
 8:     if $i^*$ not yet inserted in $T$ then  // insert new node, if not yet inserted
 9:         $\Delta^- \leftarrow \Delta^- \cup (\Delta^* \setminus D)$;   $M \leftarrow M \bowtie \pi_{\Delta^*} M^*$;   $parent(i^*) \leftarrow i$
10:     else
11:         $\Delta^+ \leftarrow \Delta^+ \cup (\Delta^* \cap D)$
12:     end if
13: end if
14: $\Delta^+ \leftarrow \Delta^+ \setminus \Delta^-$
15: $\Delta^- \leftarrow \Delta^- \setminus \Delta^+$
16: $\Delta \leftarrow \Delta \cup \Delta^+ \cup \Delta^-$  // update $\Delta$
17: return $\Delta^+, \Delta^-$
```

M_n! The bottleneck lies in the exponential growth of the size of match-sets and their computation time. It is actually possible to do better because the expected end result is the set of answers $R = \pi_{\overline{u}} M$, whose size is bounded by the number of candidate instances.

We here describe a compact representation of a match-set M, called a *match-tree*, that is made of several local joins instead of the global join. It supports the incremental computation of match-sets assumed by the partitioning algorithm, and it performs joins in a local and lazy way to keep the representation as compact as possible.

A match-set M_l results from the set of incidences P_l. A match-tree is based on a tree structure over those incidences.

Definition 18. A *match-tree* is a rooted n-ary tree T where each node is an incidence $i = (\overline{w}, a)$ and is labeled by a tuple (D, M, Δ) where[3]:

- $D \subseteq \overline{w}$ is a subset of the variables used by incidence i;
- M is the *local match-set* s.t. $\overline{w} \subseteq dom(M)$;
- $\Delta \subseteq dom(M)$ is the subdomain that is useful to the node's ancestors.

For each incidence, D is the set of variables *defined* by the incidence when added to a pre-concept. For example, starting from $[u \leftarrow film(u)]$, incidence $actor(u, w_1)$ *defines* the variable w_1. The local match-set M is a local join, i.e. a join over a subset of the incidences inserted so far. Joining all local match-sets would result in the global join. In the example, for incidence $actor(u, w_1)$, M is $M_{actor(u,w_1)}$ alone, i.e. a primitive match-set. Finally, Δ tells on which variables the local match-set can be projected without loosing information for the computation of the set of answers R. In the example, $\Delta = \{u\}$, only the films having an actor matter, not the particular actors.

In order to use match-trees in place of match-sets in the partitioning algorithm, we have to define the following operations on them: (a) the initial match-tree T_{init} to be used in the initial pre-concept, (b) the join between a match-tree T_l and a new incidence whose result must be a match-tree T_i, and (c) the projection of a match-tree on variables $\overline{u}$ to get the set of answers R_i.

The *initial match-tree* T_{init} is used in the initial pre-concept in place of the initial match-set. It has a single root node that is a pseudo-incidence $i = \top(\overline{u})$ and that is labeled with $(\overline{u}, M_{init}, \overline{u})$.

The line $M_i \leftarrow M_l \bowtie M_{(\overline{w},a)}$ doing the incremental join in Algorithm 1 is replaced by

$$T_i \leftarrow LazyJoin(T_l, \top(\overline{u}), (\overline{w}, a), D^*, M^*, \Delta^*)$$

where *LazyJoin* is defined by Algorithm 2. It is based on a recursive traversal of the match-tree (lines 2-3) starting at the root $\top(\overline{u})$, inserting the new incidence $i^* = (\overline{w}, a)$ at an appropriate place (line 9), and updating local match-sets accordingly (line 5). The new incidence is labeled as follows, where for recall $\overline{x}_l$ is the tuple of all variables in the current pattern P_l:

$$D^* = \overline{w} \setminus \overline{x}_l, \quad \Delta^* = \overline{w} \cap \overline{x}_l, \quad M^* = M_{(\overline{w},a)}.$$

Considering the variables Δ^* of the new incidence that are already in the match-set, the new incidence i^* is inserted as a child of the first visited node whose defined variables D contains at least one variable in Δ^* (lines 7-9). The common variables between D and Δ^* provide a connection between the current pattern and the new incidence. If some variables in Δ^* are not defined at the insertion node, they are returned and propagated through Δ^- as missing variables for join (line 4, 9, 17), and they are propagated from the

[3] By abuse of notation, we allow tuples of variables to be used as sets of variables.

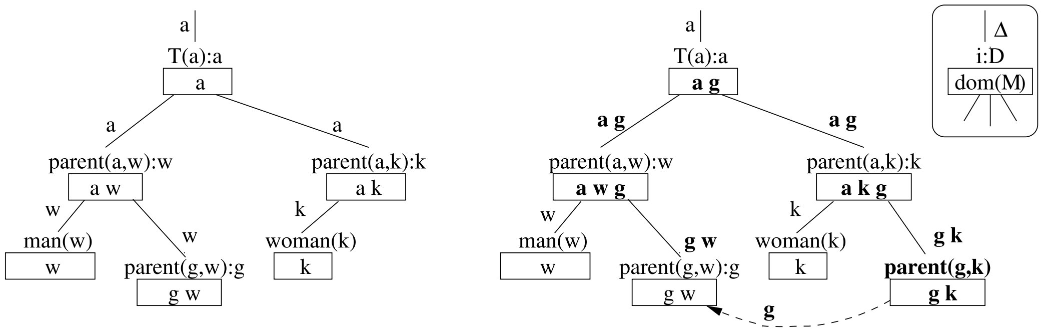

Fig. 4. Match-tree before and after lazy join with incidence $i^* = parent(g, k)$.

incidences that define them through Δ^+ as available variables for join (line 4, 11, 17). Those missing and available variables are added to the Δ^+/Δ^- of incidences upward (line 16), except when they meet each other (lines 14-15).

Finally, the set of answers R corresponding to a match-tree T is $rel(\pi_{\overline{u}}\ M_\top)$, where $M_\top$ is the local match-set of the root.

Fig. 4 shows the effect of the lazy join of the new incidence $i^* = parent(g, k)$ at the pre-concept with proper extension WHG in Fig. 3. The input match-tree is on the left, and the output match-tree is on the right. The input match-tree corresponds to the PGP

$$[a \leftarrow man(w), woman(k), parent(a, w), parent(a, k), parent(g, w)],$$

and it is hence the result of 5 successive lazy joins, each introducing an incidence. Changes (shown in bold on the right side) are propagated from the two incidences that define g and k (see D), and the new leaf is inserted under one of the two nodes as a child (here, under the node defining k). The insertion of the other incidences, which led to the match-tree on the left, changes only one path in the match-tree for each incidence because they do not introduce a cycle. In Algorithm 2, the computation of Δ^+, Δ^- is used to correctly handle cycles. They are sets of variables of the new incidence i^* that are not in the match-set of its parent (g in the example), and hence need to be joined with distant nodes in the match-tree. Δ^- propagates up the tree branch of the parent (incidence $parent(a, k)$ in Fig. 4), and Δ^+ propagates up the tree branch of distant incidences (incidence $parent(g, w)$). When they meet at their common ancestor (incidence $\top(a)$ here), the distant join can be done, and their propagation stops.

Discussion In the example on films, we observe that each new element $i^* = actor(u, w_i)$ only entails the computation $M \bowtie \pi_u\ M_{i^*}$, i.e. the intersection between the current set of films, and the set of films having an actor. The final match-tree therefore contains a match-set at the root whose size is N, and n match-sets at the leaves whose size is Nn (one for each actor). The total size is therefore in the order of Nn^2 matchings instead of Nn^n. For 1000 films related to 10 actors each, it amounts to 10^5 instead of 10^{13}! Moreover, the match-sets have 1 or 2 variables in their domain instead of $(n+1)$ for the global join.

5.3. Implementation

The main implementation of those algorithms is CONNOR, a Java library for the computation of concepts of neighbors. This library is a free and open-source software,[4] based on Apache Jena,[5] a well-known Java library for representing and reasoning on the RDF graphs of the Semantic Web. As presented in Section 3.2, there is a correspondence between RDF graphs and Graph-FCA contexts with only unary and binary relations. This is the case handled by CONNOR: graph contexts are represented as RDF graphs and handled with the Jena library. This library gives a comprehensive interface for the handling of graph contexts (with the class `ConnorModel`) and concepts of neighbors (class `ConceptOfNeighbour`), and provides classes encapsulating the algorithms for the efficient computation of concepts of neighbors (classes `ConnorPartition` and `ConnorThread`). A full presentation of CONNOR with usage examples can be found in [15].

This implementation takes into account the domain knowledge expressed as RDF Schema (RDFS) axioms [38], by applying the algorithms on the saturated version of the RDF graph. For instance, if there is an incidence $c(o)$ where c is an RDFS class, and d is known as a superclass of c in the RDF Schema axioms, then $d(o)$ is also considered as an incidence. This is possible because RDFS can only entail a finite set of facts [41].

6. Applications

This section presents different applications of Concepts of Neighbors. As shown above, the computation of concepts of neighbors is a generic method for computing distances between entities in relational datasets. The potential applications are therefore numerous and varied. As of today, three different applications have been developed and are detailed in this section: (1) query relaxation to find

[4] Accessible here: https://gitlab.inria.fr/hayats/CONNOR.

[5] https://jena.apache.org/.

approximate answering in KGs, (2) knowledge graph completion to infer missing facts in KGs, and (3) relation extraction to populate KGs from text.

6.1. Query relaxation

Query relaxation has been proposed as a way to find *approximate answers* to user queries [42]. Approximate answers are useful when the user query has too few answers or no answers at all. The lack of answers can have various reasons. Either the user has insufficient knowledge about the data schema, or the data is incomplete or noisy, or the query is too stringent. Query relaxation consists in applying transformations to the user query in order to relax constraints, and make it more general so that it produces more answers. A major issue in query relaxation is that the number of relaxed queries grows in a combinatorial way with the number of relaxation steps and the size of the query.

This section is a summary of a previous work [14], with formal notations made consistent with the previous sections. We first give a specific related work on the query relaxation task. Then we show how Concepts of Neighbors can be applied to improve the efficiency of query relaxation. Finally, we present experiments showing that, despite the higher efficiency, our approach does not trade quality for efficiency as it produces the same results as query relaxation.

6.1.1. Related work

The existing approaches for query relaxation consist in enumerating relaxed queries up to some edit distance, and to evaluate each relaxed query, from the more specific to the more general, in order to get new approximate answers [43–45]. The main issue with such an enumeration-based approach is that the number of relaxed queries grows in a combinatorial way with the edit distance, and the size of the query. Moreover, many relaxed queries do not yield any new answer because they have the same answers as more specific relaxed queries. Huang et al. [44] use a similarity score in order to have a better ranking for the evaluation of relaxed queries. Hurtado et al. [43] optimize the evaluation of relaxed queries by directly computing their proper answers. New SPARQL clauses, `RELAX` and `APPROX` [45], have also been proposed to restrict relaxation to a small subset of the query, and hence reduce the number of relaxed queries. However, this requires from the user to anticipate where relaxation can be useful. The above approaches [43,44] put some limitations on the relaxation steps that can be applied. Triple patterns can be generalized by relaxation according to RDFS inference but generally can not be removed from the query. URIs and literals cannot always be replaced by variables, which limits the generalization capabilities. Other approaches [46,47] present powerful relaxation frameworks, but they do not evaluate their efficiency or only generate a few relaxed queries.

6.1.2. Application of concepts of neighbors to query relaxation

Query relaxation takes as input an RDF graph and a query Q, and returns a (partially) ranked list of approximate answers. We have seen above how an RDF graph can be mapped to a graph context, mapping RDF nodes to objects, classes and properties to attributes, and triples to incidences. Existing work on query relaxation almost exclusively considers conjunctive queries, which are equivalent to our projected graph patterns (PGP). The problem of query relaxation can therefore be formulated in the framework of Graph-FCA. Given a graph context $K = (O, A, I)$ and a PGP $Q = [\overline{x} \leftarrow P]$, an approximate answer of Q in K is a tuple of objects $\overline{v}$ that is an answer of a generalized query Q', i.e. $Q' \subseteq_{PGP} Q \wedge \overline{v} \in ans(Q')$. By Definition 7 of ans, this is equivalent to say $Q' \subseteq_{PGP} Q \wedge Q' \subseteq_{PGP} Q(\overline{v})$, which implies that $Q' \subseteq_{PGP} Q \cap_{PGP} Q(\overline{v})$. Among the many possible relaxed queries Q', one prefers the least relaxed query, hence the most specific relaxed query $Q' = Q \cap_{PGP} Q(\overline{v})$.

We can make a concrete instance out of the query $Q = [\overline{x} \leftarrow P]$ by making a new object out of each variable, and a new incidence out of each pattern element, so that we get an augmented graph context where the projected variables $\overline{x}$ become a new instance $\overline{u}$. Now, the relationship between the query, the relaxed query and the approximate answer becomes that of a conceptual distance: $\overline{u}$ is the query instance that represents the relaxed query, $\overline{v}$ is a candidate instance that represents an approximate answer, and the intension of the conceptual distance $int(\delta(\overline{u}, \overline{v})) = Q(\overline{u}) \cap_{PGP} Q(\overline{v})$ represents the relaxed query (the most specific one for that approximate answer). As a consequence, by computing the concepts of neighbors of the instantiation $\overline{u}$ of the query, we obtain the approximate answers as their proper extensions. The partial ordering on concepts of neighbors provides a pre-order on approximate answers. Approximate answers coming from the same proper extension are considered as equally good, and approximate answers coming from more specific concepts are considered as better. A total ordering can be obtained by combining extensional or intensional numerical similarity on concepts and a global ranking on instances. For each approximate answer, the concept intension tells us which relaxations have been applied to the query.

Compared to existing approaches of query relaxation where the time delay before finding the first approximate answers is not bounded, the partitioning algorithm computing concepts of neighbors can provide a ranking of approximate answers from the beginning. This ranking simply gets more and more accurate with runtime. Another advantage is that the number of considered relaxed queries is bounded by the number of instances – and in general much lower – rather than exponential in the size of the query. Moreover, the computation of their sets of answers is shared in part between the relaxed queries.

6.1.3. Experiments

We here report experiments that compare the efficiency of Concepts of Neighbors to existing approaches for query relaxation. All details can be found in [14]. We compare four algorithms: two baseline algorithms, RelaxEnum and NodeEnum, and two variants of our algorithm, Partition and PartitionLJ. RelaxEnum is the classical approach of query relaxation, that enumerates relaxed queries and computes their answers. NodeEnum does the opposite by enumerating RDF nodes, and computing the least common subsumer

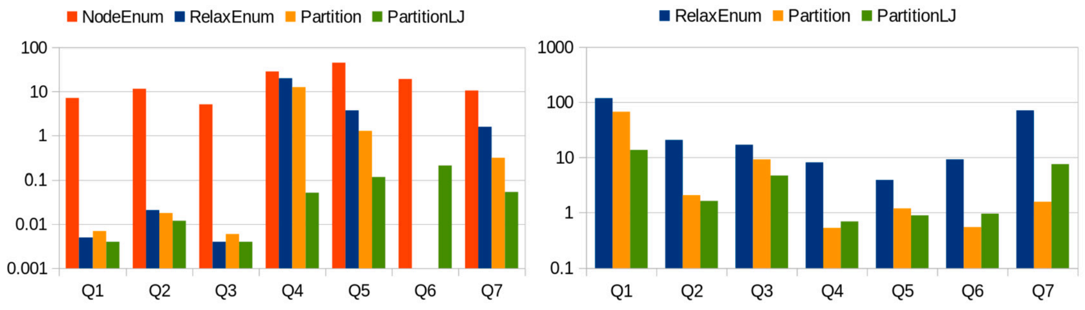

Fig. 5. Runtime (seconds, log scale) per algorithm and per query for MONDIAL (left) and LUBM10 (right).

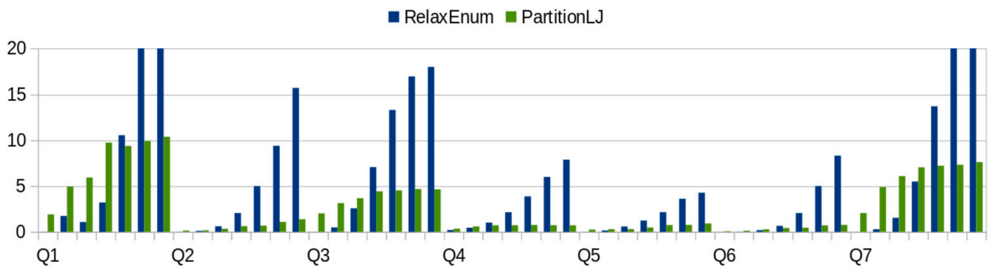

Fig. 6. Runtime (seconds) per algorithm and per LUBM10 query for increasing maximum relaxation distance (1 to 7). Full height bars represent runtimes over 20 seconds.

between the query and the node description [48]. PARTITION and PARTITIONLJ are our two variants, and differ in that only the latter uses lazy joins for computing concept extensions. We consider two execution modes: NOLIMIT for complete computations; and MAXRELAX for a limit to relaxation distance (not applicable to NODEENUM). For experiments, we use sets of queries on a few datasets of different size: 7 queries having 5 to 21 elements in their pattern on a geographical dataset (MONDIAL [49], 12k triples), 7 queries having 3 to 7 elements in their pattern [44] on two synthetic datasets about universities (LUBM10, 1.3M triples; LUBM100, 13M triples [50]).

Results in NoLimit mode Fig. 5 compares the runtime (log scale) of all algorithms on all queries of MONDIAL and LUBM10 in NOLIMIT mode. A few runtimes are missing for NODEENUM on LUBM10, and for RELAXENUM and PARTITION on Q6 of MONDIAL because they are much higher than other runtimes. RELAXENUM is sensitive to the query complexity, in particular to multivalued properties. NODEENUM looks insensitive to the query complexity, but does not scale well with the number of nodes. PARTITION and PARTITIONLJ are always more efficient – or equally efficient – and the use of lazy joins generally makes it even more efficient. On LUBM10, PARTITIONLJ is typically one order of magnitude more efficient than RELAXENUM. It can also be observed that PARTITIONLJ scales well with data size because from MONDIAL to LUBM10, a 100-fold increase in number of triples, the median runtime also follows a 100-fold increase, from 0.01-0.1 s to 1-10 s. This linear scaling is verified on LUBM100 (not shown) where the runtimes are all 10 times higher. We want to emphasize that it is encouraging that the full relaxation of a query over a 1.3M-triples dataset can be done in 4 s on average.

Results in MaxRelax mode In practice, one is generally interested in the most similar approximate answers, and therefore in the relaxed queries with the smaller relaxation distances. It is therefore interesting to compare algorithms when the relaxation distance is bounded. Fig. 6 compares RELAXENUM and PARTITIONLJ on LUBM10 queries for increasing values of maximal relaxation distance, here from 1 to 7 relaxations. As expected, the runtime of RELAXENUM grows in a combinatorial way, like the number of relaxed queries, with the relaxation distance. On the contrary, the runtime of PARTITIONLJ grows more quickly for 1–4 relaxations, and then almost flattens in most cases. The flattening can be explained by several factors. The main factor is that most relaxed queries are redundant and are pruned by the partitioning algorithm. Another factor is that query evaluation is partially shared between different queries. That sharing is also the cause for the higher cost with 1–4 relaxations. In summary, PARTITIONLJ scales very well with the number of relaxations, while RELAXENUM does not. This is a crucial property for similarity search where many relaxation steps are necessary.

Number of relaxed queries The efficiency of PARTITION and PARTITIONLJ is better understood when comparing the number of concepts of neighbors (C-N) to the number of relaxed queries (RQ). Over the 7 LUBM10 queries, there are in total 59 C-Ns out of 2899 RQs, hence a 50-fold decrease.

6.2. Knowledge graph completion

The open nature of KGs often implies that they are incomplete, and a lot of work have studied the use of machine learning techniques to complete them. The task of *knowledge graph completion*, aka. *link prediction* [51], consists in predicting a missing edge between two entities: e.g., predicting that the director of the film *Avatar* is James Cameron. In Graph-FCA term, given a graph context $K = (O, A, I)$ representing a knowledge graph, a binary attribute $r \in A$ representing a *relation*, and an object $u \in O$ representing an *entity*, the objective is to predict the missing entity in an incomplete incidence $r(u, ?)$ or $r(?, u)$. Because of the symmetry between the two cases, we only consider in the following the case $r(u, ?)$ where the *head entity* u is fixed and the *tail entity* v is to be predicted.

This section is a summary of a previous work [16] on the exploitation of Concepts of Neighbors for knowledge graph completion. Indeed, the partitioning of the KG entities into concepts of neighbors provides a valuable basis for different kinds of inferences, like the inference of the missing node of an incomplete edge. One advantage is that inference can be performed without training a model beforehand, so that the proposed approach can easily cope with dynamic knowledge graphs. Another advantage is that explanations can be provided for each inference, based on the intent of concepts of neighbors. Finally, the experimental results show competitive performance w.r.t. state-of-the-art approaches in link prediction.

6.2.1. Related work

Nickel et al. [51] have identified two kinds of approaches that differ by the kind of model they use: *latent feature models*, and *graph feature models*. The former is by far the most studied one. Latent feature models learn *embeddings* of entities and relations into low-dimensional vector spaces, and then compute the truthfulness score of edges $r(u, v)$ by combining the embeddings of the two entities and the embedding of the relation: e.g., TransE [52], RGNN [53], ConvE [54]. Graph feature models, also called *observed feature models*, make inferences directly from the observed edges in the KG: e.g., Random walk inference [55], Horn clauses AMIE 3 [35], AnyBURL [56]. Latent feature models have the best performance, but they do not provide explanations for their inferences, unlike graph feature models. The key novelty of Concepts of Neighbors is that they offer an instance-based approach rather than a model-based approach. This implies that there is no need for a training phase, and that all the learning effort is done at inference time.

6.2.2. Application of concepts of neighbors to link prediction

Given an incomplete incidence $r(u, ?)$, we compute the concepts of neighbors (C-Ns) of head entity u. From there, we infer a ranking of candidate entities for the tail entity v. In fact, as r is not involved in the computation of C-Ns, many target relations can be inferred at nearly the same cost as a single relation. Indeed, the main cost is in the computation of C-Ns. Moreover, the latter is easily controlled because the computation algorithm is any-time (see Section 2).

The principle of C-N-based inference is to generate a ruleset for each concept of neighbors $\delta_l \in C{-}N(u)$. Those rules are similar in nature to those of AMIE+ or AnyBURL except that only rules that are matched by the head entity u are generated. Indeed, the bodies of generated rules are intensions of C-Ns, which are subsets of u's description ($int(\delta_l) \subseteq_{PGP} Q(u)$). From the concept intension $int(\delta_l) = Q_l = [x \leftarrow P_l]$, with $\overline{x}_l$ the tuple of variables in P_l, we generate two kinds of inference rules ρ:

1. **by-copy rules**: $\rho := P_l \rightarrow r(x, v)$, for each $v \in O$;
2. **by-analogy rules**: $\rho := P_l \rightarrow r(x, y)$, for each $y \in \overline{x}_l, y \neq x$.

Inference by copy The first kind of rules (**by-copy rules**) predicts that when an entity matches P_l as x, it is related to the entity v through the relation r. As u matches P_l as x by definition of concepts of neighbors, the rule infers that u is related to v. In formula, the set of inferred entities is simply $V_\rho = \{v\}$. Of course, the strength of the inference depends on the support and confidence of the rule ρ, which are defined as follows.

$$supp(\rho) = |ans([x \leftarrow P_l, r(x, v)])| \quad conf(\rho) = \frac{|ans([x \leftarrow P_l, r(x, v)])|}{|ans([x \leftarrow P_l])| + \lambda}$$

Like this is done in AnyBURL [56], we use a constant $\lambda \geq 0$ as an additive Laplace smoothing, in order to favor rules with larger support. The principle of *inference by copy* is that the tail entities v to be predicted for u can be copied from the tail entities of u's neighbors. For example, if incidence $parent(Charlotte, Kate)$ is missing in the example graph of Fig. 1, it can be inferred from some of her closest neighbors: *"From one hand, George is similar to Charlotte because he has William as a father; from the other hand George's mother is Kate; so Charlotte's mother is expected to be Kate too"* (support $= 1$, confidence $= 1/(2 + \lambda)$).

Inference by analogy The second kind of rules (**by-analogy rules**) predicts that when a pair of entities match P_l as x and y, they are related through r. As u matches P_l as x by definition of concepts of neighbors, the rule infers that u is related through r to any entity v that matches P_l as y when $x = u$. In formula, the set of inferred entities is $V_\rho = \{v \mid (u, v) \in ans([x, y \leftarrow P_l])\}$. Like above, the strength of the inference depends on the support and confidence of the rule, whose definitions are similar to by-copy rules except that they involve pairs of entities instead of entities.

$$supp(\rho) = |ans([x, y \leftarrow P_l, r(x, y)])| \quad conf(\rho) = \frac{|ans([x, y \leftarrow P_l, r(x, y)])|}{|ans([x, y \leftarrow P_l])| + \lambda}$$

The principle of *inference by analogy* here relies on the observation that "u is to v as x is to y". By observing that pairs (x, y) that match pattern P_l often satisfy $r(x, y)$, one can predict the missing entity in $r(u, ?)$ to be any entity v such that pattern P_l is matched

Table 2
MRR performance of representative approaches on link prediction benchmarks.

	WN18	WN18RR	FB15k	FB15k-237
#triples	151k	95k	592k	310k
ConvE	0.942	0.460	0.745	0.316
ComplEx-N3	0.950	**0.480**	**0.860**	**0.370**
AnyBURL	0.950	**0.480**	0.830	0.300
C-N	**0.969**	0.469	0.849	0.296

for $x = u$ and $y = v$. For example, assume in the example graph that George and Charlotte are only known to have for father William, and that Kate is only known to be William's spouse, i.e. incidences $parent(George, Kate)$ and $parent(Charlotte, Kate)$ are missing. Here, inference by copy for the parents of Charlotte would only produce Charles and Diana, using William and Harry as similar entities (see C-N 4 in Fig. 2). The intension of the conceptual similarity of Charlotte with William and Harry is

$$\begin{aligned} Q_{WH} &= [x \leftarrow P_{WH}] \\ P_{WH} &= parent(x,y), man(y), spouse(y,z), spouse(z,y), woman(z), \end{aligned}$$

saying that *"they have a father married to a woman"*. From there, the following by-analogy rule can be generated: $P_{WH} \rightarrow parent(x,z)$. This rule states that *"any female spouse (wife) of a male parent (father) is a parent"*. Applying this rule to Charlotte (and equivalently to George) leads to the inference of $parent(Charlotte, Kate)$ because Kate is indeed the wife of William, who is the father of Charlotte. It is noteworthy here that Kate is predicted to be a parent, although she never appears as a parent in the incomplete graph. This is not possible with inference by copy.

Aggregation of inferences Given an incomplete triple $r(u, ?)$, the output of a link prediction system is a ranking of entities v. Rankings of entities are evaluated with measures such as Hits@N (defined as 1 if the correct entity is among the first N entities, 0 otherwise), and MRR (Mean Reciprocal Rank, the average of the inverse of the rank of the correct entity, in range $[0,1]$). The question that remains to be addressed is how to aggregate inferences from all above rules into a global ranking. Indeed, the known entity u leads to multiple concepts of neighbors, each concept of neighbors δ_l generates multiple rules, and each rule ρ infers a set V_ρ of candidate entities v. The same candidate v may be inferred by several rules, possibly generated from different concepts. The idea is to combine the measures of rules to give a score to each candidate v, in order to get a global ranking. We reuse the **Maximum Confidence (MC)** score introduced in AnyBURL [56]. In short, the score of an entity v is the list of the confidences of the rules that inferred it, in decreasing order.

6.2.3. *Experiments*

The approach based on Concepts of Neighbors has been compared to state-of-the-art approaches – both latent-based and rule-based – on classical benchmarks for link prediction: WN18, WN18RR, FB15k, FB15k-237. The main results are shown in Table 2, and all details about the experiments are available in [16]. ComplEx-N3 (latent-based) clearly outperforms other approaches on all datasets except WN18 where C-N outperforms other approaches on the three measures (MRR, Hits@1, Hits@10). C-N comes second on FB15k, and remains close to the best approaches on WN18RR. On FB15k-237, the MRR delta is -0.074 with ComplEx-N3, but only -0.004 with AnyBURL, the best rule-based approach. It is noteworthy that C-N is competitive with AnyBURL because, whereas AnyBURL rules are computed in a supervised manner (knowing the target relation r), C-N concepts are computed in an unsupervised manner (i.e., only knowing the head entity u). This implies that the concepts of neighbors of an entity can be computed once, and used for many different inference tasks, e.g. predicting links for several target relations.

The MONDIAL geographical dataset [49] was used to evaluate how complementary by-copy and by-analogy rules are. Although by-analogy rules appear more powerful than by-copy rules with an MRR equal to 0.424 vs 0.286, it is beneficial to combine the two as this raises the MRR to 0.455.

We report on an example of inference in the MONDIAL geographical dataset [49]. The Islay island is correctly predicted to belong to Inner Hebrides (score = 0.34 0.34 0.31) by two by-analogy rules (supp = 54, conf = 0.34) and one by-copy rule (supp = 11, conf = 0.31). The five subsequent predictions are other UK islands that were inferred by the same first and second by-analogy rules, and by another third rule with lower confidence. The top-1 rule is the following one.

$$\begin{aligned} & locatedIn(x,u), locatedIn(z,u), belongToIslands(z,y), \\ & locatedInWater(x,v), (v = AtlanticOcean) \\ & \rightarrow belongToIslands(x,y) \end{aligned}$$

It is a by-analogy rule that says that an island x belongs to islands y if x shares a location u (here, UK) with another island z that belongs to y, and x is located in the Atlantic Ocean. Apart from the specialization to the Atlantic Ocean, this rules makes sense because if two islands are located in a same place, e.g. a same country, there is a good chance that they belong to the same islands.

6.3. Relation extraction

Relation extraction is a classical task in machine learning for natural language processing (NLP). This task consists into predicting which type of relation – if any – links two given entities (a subject and an object) in a natural language text. For example, in the sentence "*The University of Rennes is French*", with *University of Rennes* tagged as subject and *French* as object, the objective is to identify that the relation *nationality* links the subject to the object.

This section gathers the results of two previous studies [17,18]. It presents a method based on concepts of neighbors for relation extraction, and shows positive results of this approach, being competitive with the state of the art and presenting interesting interpretability properties. First, the related work about the relation extraction task is presented. Second, we explain how concepts of neighbors can be used for relation extraction. Finally, we summarize the experiments conducted and the results obtained, pointing its results both on the quantitative aspect and the advantages in terms of interpretability. Note that the dataset used in the experimental part is in English, but this work is relatively independent of the language, and could be applied to other languages, as long as the language structure allows for tools similar to those used to exist, and as long as those tools have been developed.

6.3.1. Related works

Initially, as for many NLP tasks, the first approaches proposed for relation extraction used handmade rules applied to sentences parsed thanks to handmade grammars. As the design of those rules and grammars were time-consuming for a hardly reliable result, those methods were abandoned to the profit of supervised machine learning approaches, using annotated corpora. For more details, see the review [57]. During the last decade, with the important progresses in deep learning, neural network-based approaches appeared: convolutional neural networks [58], then LSTM [59], and graph convolution networks [60,61]. Today, the state of the art is dominated by transformer-based pre-trained language models, such as BERT and its variations [62–64]. Despite their impressive results, those approaches, as most deep-learning approaches, act as black boxes and lack interpretability.

Most relation extraction datasets and approaches use a negative class, or *no_relation* class, collecting all the task instances (i.e., pairs of named entities) for which no relation exists between the subject and the object. Therefore, the relation extraction task can be divided in two steps: a *relation detection* step, charged to determine if a given instance expresses a relation, and a *relation classification* step, in which the instances expressing a relation are classified among different relation types. Such a two-step approach is introduced in [65]. As the Concepts of Neighbors approach is based on the computation of similarities between instances, and as the instances of the negative class have no reason to be similar one to each other, the method developed below is suited for relation classification, and should be coupled with a relation detection module to perform proper relation extraction, as presented in [18].

6.3.2. Application of concepts of neighbors to relation classification

The task of performing relation classification on natural text sentences with concepts of neighbors can be divided into three steps. First, the textual data have to be transformed into relational data. Then, Concepts of Neighbors have to be computed for each instance to be classified. Finally, a ranking method has to be defined to infer a prediction from the set of concepts of neighbors.

Sentence modeling The first step is to model each instance of the relation extraction dataset as a graph. An instance is a sentence that is annotated with the position and type of two entities (subject and object), and that is labeled with the expected relation type between the two entities. The modeling presented in this section has both syntactic and semantic features, and relies on an NLP toolkit (such as Stanford CoreNLP [66]) and on WordNet [67], a hierarchical lexical database for English. The modeling is made as an RDF graph, because this formalism is flexible, permits type relaxation, and the CONNOR library (see 5.3) has been developed for computing concepts of neighbors in RDF graphs.

The sentence is first processed with an NLP pipeline (composed of a tokenizer, a lemmatizer, a part-of-speech tagger, a dependency tree parser and a named entity recognizer) in order to extract linguistic knowledge. Then, the result of this processing is used to create a syntactic modeling of the sentence. An RDF entity with a unique id is created for each token and each named entity of the sentence. Then the dependency relations between those tokens and named entities are added in order to represent the syntactic tree of the sentence. Finally, RDF types are added to the model for representing the part-of-speech tags, the lemmas, the named entity positions and types. Fig. 7 shows the syntactic modeling of the sentence *"The University of Rennes is French"*.

To this syntactic modeling, we add a semantic layer, using *WordNet*. To do so, an RDF ontology is built on the lemmas of the verbs and nouns: for each lemma, a supertype is created for each synset containing this lemma, and for each synset, a supertype is created for each of its hypernym synsets. This allows for type relaxation over the lemmas, and therefore identification of semantically-close lemmas. Fig. 8 represents an extract of the type hierarchy that is generated by the lemmas *university*, *school*, *religion* and *faith*. It can be seen that the lemmas *university* and *school* can both be relaxed into the synset *educational institution*, and that those four lemmas can all be relaxed into the synset *institution*.

By combining the syntactic modeling of sentences and the semantic relaxation mechanism over the lemmas, we obtain a syntactic and semantic modeling of natural language sentences as RDF graphs in which each token and named entity is identified by a unique RDF entity for which the computation of concepts of neighbors is possible.

Use of concepts of neighbors For a given instance, the objective is to discover which are the other instances of the training dataset that are the more similar, and then predict a relation type from them. As an instance is a (subject, object) couple, we use binary concepts of neighbors, whose extensions are sets of similar (subject, object) couples.

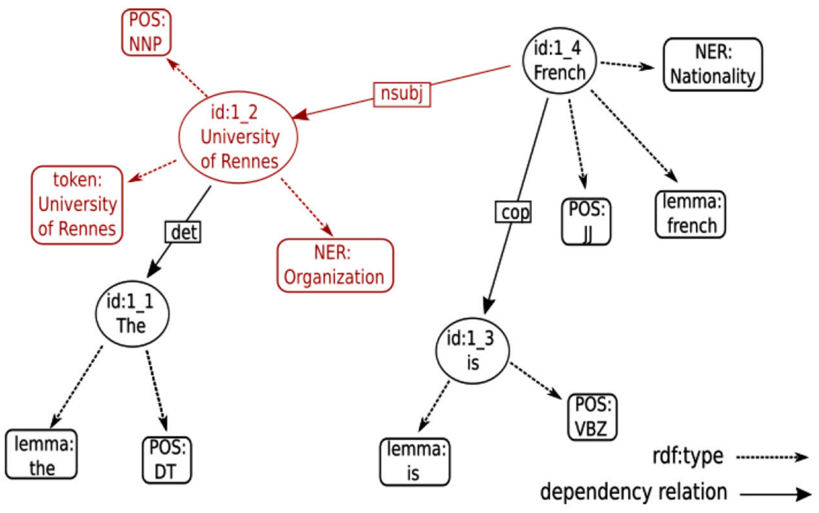

Fig. 7. Syntactic modeling of the sentence "The University of Rennes is French".

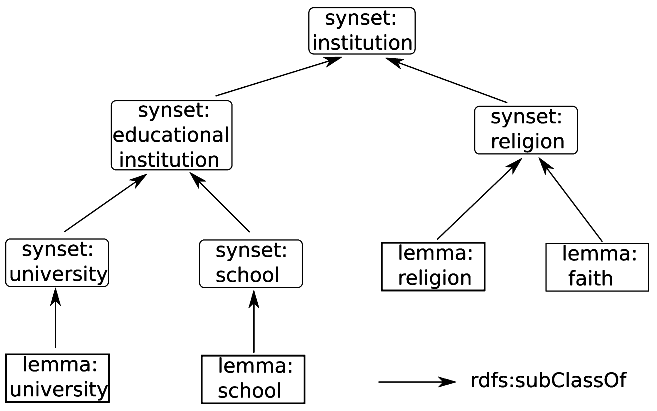

Fig. 8. Example of type hierarchy over lemmas generated with WordNet.

In addition, a mechanism similar to the RECENT paradigm presented in [64] is used: before computing the concepts of neighbors of a (subject, object) couple, we deduce from its subject and object types the relation types that are compatible with this query instance. Then we simply limit the set of candidate instances to the training instances labeled with a compatible relation type.

Scoring method for relation classification For each instance, a set of rules predicting the different relation types is constructed from its set of concepts of neighbors: for each relation type r and for each concept of neighbors δ, we consider a rule $R_{r,\delta}$ having for head the relation type and for body the intension of the concept, and we compute its confidence in the classical way:

$$conf(R_{r,\delta}) = \frac{|\{(s,o) \in \delta.ext \mid r(s,o)\}|}{|\delta.ext|}$$

Then, similarly to Section 6.2, the MC scoring method is used on those confidences to sort the possible predictions.

From relation classification to relation extraction As explained before, the method presented above is suited for relation classification but not for relation extraction, and therefore it must be combined with a relation detection module to perform full relation extraction. In [18], this method is combined with LUKE [63], a transformer-based pre-trained language model fine-tuned on the relation detection task in order to obtain a two-step relation extraction method.

6.3.3. *Experiments*

This section successively presents the dataset used to evaluate this approach, the comparison points used during those evaluations, and the results of these evaluations.

Dataset The following experiments were conducted on TACRED [60], a standard widely-used dataset for Relation Extraction proposed by Stanford University and composed of 100,000 instances (split between train, development and test instances) classified

Table 3
Accuracy on the relation classification task, compared to baseline.

Timeout (s)	10	20	30	60	120	300	600	1200
Ours	82.0	82.1	82.7	82.9	83.4	**83.6**	**83.6**	**83.6**
Baseline	80.4							

Table 4
F-score for several Relation Extraction methods on TACRED.

Method	F1 score
LUKE [63]	**72.7**
BERT-LSTM-Base [68]	67.8
Ours	*66.9*
C-GCN [60]	66.4
GCN [60]	64.0

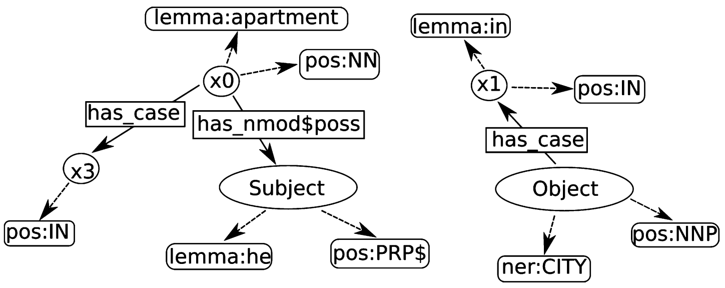

Fig. 9. An explanation for the inference of relation *per:city_of_residence*.

among 41 relation types and a *no_relation* negative class. Issued from the TAC KBP challenge, those instances consist into sentences in English, mainly issued from newspaper, annotated by crowdsourcing. In order to reflect real-world usage, this dataset contains 79.5% negative examples. For evaluation on the relation classification sub-task, only the positive examples are considered, while the whole dataset is used for experiments on the relation extraction task.

Comparison points For relation classification, as the experiments are made on a subset of TACRED, the comparison is made with a naive baseline. For a given example, this baseline simply predicts the most probable relation type in the dataset according to the types of the subject and object.

For relation extraction, the comparison is made with pre-existing deep-learning relation extraction systems. Two of them use transformer-based pre-trained language models (LUKE [63] and BERT-LSTM-Base [68]), while GCN and C-GCN use graph convolution networks over the dependency tree [60].

Results On the relation classification task, several experiments were conducted. As explained in Section 5.1, our algorithm is *anytime*, and therefore experiments were made using different timeouts. Table 3 presents the accuracy scores for both our approach (with a timeout range from 10 seconds to 20 minutes) and the baseline. It can be seen that, whatever is the timeout, our approach outperforms the baseline. In addition, this shows that for a timeout over 120 seconds, the accuracy saturates: the score of 83.6% seems to be the best achievable score here, whatever the timeout.

On the relation extraction task, experiments were conducted by combining our relation classification method with a relation detection module, as presented previously. The results obtained are presented in Table 4. It can be seen that, even if our approach cannot compete with state-of-the-art transformer-based deep learning approaches, it outperforms approaches such as GCN or C-GCN, which rely on a similar representation of the text.

However, the main asset of this approach for relation extraction is not quantitative but qualitative, and rely on interpretability: for each positive prediction, we can obtain the list of rules from which this prediction was made, and use it as an explanation of the prediction. Let us take for example the sentence *“Sollecito has said he was at* ***his*** *own apartment in* ***Perugia****, working at his computer.”* Our system predicts that between the subject *his* and the object *Perugia*, a relation is detected and is of type *per:city_of_residence*. If we look to the rules of high confidence that predicted this relation type, we find a rule of body presented in Fig. 9. This rule body can be interpreted as:

- The subject has lemma *he* and is the possessor of an apartment;
- The object is the name of a city in which there is something.

We can effectively claim that such a rule seems reasonable to predict with a good confidence relation *per:city_of_residence*.

7. Conclusion

In this article we have presented the Concepts of Neighbors, an FCA-based approach for instance-based learning on relational data. This approach, based on Graph-FCA (an extension of FCA for relational data), provides a symbolic notion of distance between entities (or tuples of entities), and therefore allows for inference tasks similarly to *k-Nearest-Neighbors*. An efficient anytime algorithm has been developed for the computation of concepts of neighbors, based on the progressive partitioning of the set of potential neighbors into concepts and on a lazy version of the join operator. CONNOR, a Java implementation of this method on RDF graphs, has been presented, and is available as open-source code. Finally, we have shown the versatility of concepts of neighbors by presenting three applications on three different tasks. Two applications work on knowledge graphs, while the other works on graph representations of texts.

In future works, other kinds of applications of Concepts of Neighbors on Graph-FCA can be considered. For instance, in the semantic web field, Concepts of Neighbors can be used on knowledge graph entities for knowledge graph alignment or entity clustering. In the NLP domain, the syntactic-semantic modeling of text as RDF graphs, combined to the Concepts of Neighbors, could be used for other tasks such as semantic similarity. More generally, this method could be extended to other relational data types, such as molecular graphs. In addition, the concepts of neighbors approach can be adapted to other extensions of FCA, and therefore this instance-based learning approach could be applied to any data that could be formalized in an FCA framework.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

A link to a public software repository is given in the manuscript.

Acknowledgements

The authors wish to thank Nicolas Fouqué (CNRS) for his work on the first Java implementation of the algorithms for the computation of Concepts of Neighbors, as well as Pierre Nunn (ENS Rennes) for his experimental and development work on enhancing these algorithms.

References

[1] E.F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (6) (1970) 377–387.
[2] R. Angles, C. Gutierrez, Survey of graph database models, ACM Comput. Surv. 40 (1) (2008) 1:1–1:39.
[3] C. Gutierrez, J.F. Sequeda, Knowledge graphs, Commun. ACM 64 (3) (2021) 96–104.
[4] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts, in: I. Rival (Ed.), Ordered Sets, Springer, Netherlands, Dordrecht, 1982, pp. 445–470.
[5] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, 1999.
[6] S. Ferré, M. Huchard, M. Kaytoue, S.O. Kuznetsov, A. Napoli, Formal concept analysis: from knowledge discovery to knowledge processing, in: A Guided Tour of Artificial Intelligence Research: Volume II: AI Algorithms, Springer International Publishing, 2020, pp. 411–445.
[7] K.-M. Yang, E.-H. Kim, S.-H. Hwang, S.-H. Choi, Fuzzy concept mining based on formal concept analysis, Int. J. Comput. 2 (3) (2008).
[8] K.E. Wolff, Temporal concept analysis, in: International Conference on Conceptual Structures (ICCS), 2001, pp. 91–107.
[9] B. Ganter, S.O. Kuznetsov, Pattern structures and their projections, in: Conceptual Structures: Broadening the Base, in: Lecture Notes in Computer Science, Springer, 2001, pp. 129–142.
[10] M. Rouane-Hacene, M. Huchard, A. Napoli, P. Valtchev, Relational concept analysis: mining concept lattices from multi-relational data, Ann. Math. Artif. Intell. 67 (1) (2013) 81–108.
[11] J. Kötters, Concept lattices of a relational structure, in: International Conference on Conceptual Structures (ICCS), Springer, Berlin, Heidelberg, 2013, pp. 301–310.
[12] S. Ferré, P. Cellier, Graph-FCA in practice, in: International Conference on Conceptual Structures (ICCS), 2016, pp. 107–121.
[13] D.W. Aha (Ed.), Lazy Learning, Springer International Publishing, 1997.
[14] S. Ferré, Answers partitioning and lazy joins for efficient query relaxation and application to similarity search, in: The Semantic Web, 2018, pp. 209–224.
[15] H. Ayats, P. Cellier, S. Ferré, CONNOR: exploring similarities in graphs with concepts of neighbors, in: ETAFCA 2022 - Existing Tools and Applications for Formal Concept Analysis, 2022.
[16] S. Ferré, Application of concepts of neighbours to knowledge graph completion, Data Sci. 4 (1) (2021) 1–28.
[17] H. Ayats, P. Cellier, S. Ferré, Extracting relations in texts with concepts of neighbours, in: International Conference on Formal Concept Analysis, 2021.
[18] H. Ayats, P. Cellier, S. Ferré, A two-step approach for explainable relation extraction, in: Advances in Intelligent Data Analysis, 2022, pp. 14–25.
[19] S. Ferré, Concepts de plus proches voisins dans des graphes de connaissances, in: Journées Francophones D'Ingénierie des Connaissances, 2017, pp. 163–174.
[20] T. Horváth, S. Wrobel, U. Bohnebeck, Relational instance-based learning with lists and terms, Mach. Learn. 43 (1) (2001) 53–80.
[21] A. Hermann, S. Ferré, M. Ducassé, An interactive guidance process supporting consistent updates of RDFS graphs, in: International Conference on Knowledge Engineering and Knowledge Management, vol. 7603, 2012, pp. 185–199.
[22] S.O. Kuznetsov, Machine learning and formal concept analysis, in: Concept Lattices, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2004, pp. 287–312.

[23] A. Leeuwenberg, A. Buzmakov, Y. Toussaint, A. Napoli, Exploring pattern structures of syntactic trees for relation extraction, in: Int. Conf. Formal Concept Analysis, vol. 9113, 2015, pp. 153–168.
[24] S.O. Kuznetsov, Fitting pattern structures to knowledge discovery in big data, in: International Conference on Formal Concept Analysis, vol. 7880, 2013, pp. 254–266.
[25] S.O. Kuznetsov, Scalable knowledge discovery in complex data with pattern structures, in: Pattern Recognition and Machine Intelligence, in: Lecture Notes in Computer Science, 2013, pp. 30–39.
[26] V. Codocedo, I. Lykourentzou, A. Napoli, A semantic approach to concept lattice-based information retrieval, Ann. Math. Artif. Intell. 72 (1) (2014) 169–195.
[27] A. Inokuchi, T. Washio, H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, in: Principles of Data Mining and Knowledge Discovery, 2000, pp. 13–23.
[28] J. Huan, W. Wang, J. Prins, Efficient mining of frequent subgraphs in the presence of isomorphism, in: IEEE International Conference on Data Mining, 2003, pp. 549–552.
[29] X. Yan, J. Han, gSpan: graph-based substructure pattern mining, in: IEEE International Conference on Data Mining, 2002, pp. 721–724.
[30] F. Zhu, X. Yan, J. Han, P.S. Yu, GPrune: a constraint pushing framework for graph pattern mining, in: Advances in Knowledge Discovery and Data Mining, 2007, pp. 388–400.
[31] X. Yan, J. Han, CloseGraph: mining closed frequent graph patterns, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 286–295.
[32] F. Bariatti, P. Cellier, S. Ferré, GraphMDL+: interleaving the generation and MDL-based selection of graph patterns, in: Annual ACM Symposium on Applied Computing, 2021, pp. 355–363.
[33] M. van Leeuwen, T. De Bie, E. Spyropoulou, C. Mesnage, Subjective interestingness of subgraph patterns, Mach. Learn. 105 (1) (2016) 41–75.
[34] R. Ramezani, M.A. Nematbakhsh, M. Saraee, Mining association rules from semantic web data without user intervention, J. Comput. Secur. 7 (1) (2020) 81–94.
[35] J. Lajus, L. Galárraga, F. Suchanek, Fast and exact rule mining with AMIE 3, in: European Semantic Web Conference, in: Lecture Notes in Computer Science, 2020, pp. 36–52.
[36] S. Ferré, A proposal for extending formal concept analysis to knowledge graphs, in: Int. Conf. Formal Concept Analysis, vol. 9113, Springer International Publishing, Cham, 2015, pp. 271–286.
[37] S. Ferré, P. Cellier, Graph-FCA: an extension of formal concept analysis to knowledge graphs, Discrete Appl. Math. 273 (2020) 81–102.
[38] P. Hitzler, M. Krötzsch, S. Rudolph, Foundations of Semantic Web Technologies, Chapman and Hall/CRC, 2009.
[39] J.F. Sowa, Conceptual structures: information processing in mind and machine, in: Association for Computing Machinery, Addison-Wesley Pub., Reading, MA, 1983.
[40] S. Ferré, P. Cellier, Modeling complex structures in graph-FCA: illustration on natural language syntax, in: Existing Tools and Applications for Formal Concept Analysis, 2022, p. 1.
[41] F. Goasdoué, I. Manolescu, A. Roatiş, Efficient query answering against dynamic rdf databases, in: Int. Conf. Extending Database Technology, ACM, 2013, pp. 299–310.
[42] T. Gaasterland, Cooperative answering through controlled query relaxation, IEEE Expert 12 (5) (1997) 48–59.
[43] C.A. Hurtado, A. Poulovassilis, P.T. Wood, Query relaxation in RDF, J. Data Semant. X (2008) 31–61.
[44] H. Huang, C. Liu, X. Zhou, Approximating query answering on RDF databases, World Wide Web 15 (1) (2012) 89–114.
[45] R. Frosini, A. Calì, A. Poulovassilis, P.T. Wood, Flexible query processing for SPARQL, Semant. Web 8 (4) (2017) 533–563.
[46] P. Dolog, H. Stuckenschmidt, H. Wache, J. Diederich, Relaxing RDF queries based on user and domain preferences, J. Intell. Inf. Syst. 33 (3) (2008) 239.
[47] S. Elbassuoni, M. Ramanath, G. Weikum, Query relaxation for entity-relationship search, in: Extended Semantic Web Conference, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, pp. 62–76.
[48] S. Colucci, F.M. Donini, E. Di Sciascio, Common subsumbers in RDF, in: AI*IA 2013: Advances in Artificial Intelligence, 2013, pp. 348–359.
[49] W. May, Information Extraction and Integration with Florid: the Mondial Case Study, Technical Report 131, Universität Freiburg, Institut für Informatik, 1999.
[50] Y. Guo, Z. Pan, J. Heflin, LUBM: a benchmark for OWL knowledge base systems, in: International Semantic Web Conference, vol. 3, 2005, pp. 158–182.
[51] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, in: IEEE, vol. 104, 2015, pp. 11–33.
[52] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Advances in Neural Information Processing Systems, 2013, p. 9.
[53] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: European Semantic Web Conference, in: Lecture Notes in Computer Science, 2018, pp. 593–607.
[54] T. Dettmers, P. Minervini, P. Stenetorp, S. Riedel, Convolutional 2D knowledge graph embeddings, in: AAAI Conference on Artificial Intelligence, vol. 32, 2018, pp. 1811–1818.
[55] N. Lao, T. Mitchell, W.W. Cohen, Random walk inference and learning in a large scale knowledge base, in: Conf. of Empirical Methods in Natural Language Processing, 2011, pp. 529–539.
[56] C. Meilicke, M.W. Chekol, D. Ruffinelli, H. Stuckenschmidt, Anytime bottom-up rule learning for knowledge graph completion, in: International Joint Conference on Artificial Intelligence, 2019, pp. 3137–3143.
[57] R. Grishman, Twenty-five years of information extraction, Nat. Lang. Eng. (2019) 677–692.
[58] T.H. Nguyen, R. Grishman, Relation extraction: perspective from convolutional neural networks, in: Workshop on Vector Space Modeling for Natural Language Processing, 2015, pp. 39–48.
[59] Y. Xu, L. Mou, G. Li, Y. Chen, H. Peng, Z. Jin, Classifying relations via long short term memory networks along shortest dependency paths, in: Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2015, pp. 1785–1794.
[60] Y. Zhang, P. Qi, C.D. Manning, Graph convolution over pruned dependency trees improves relation extraction, in: Conference on Empirical Methods in Natural Language Processing, 2018, pp. 2205–2215.
[61] F. Wu, T. Zhang, Simplifying graph convolutional networks, in: International Conference on Machine Learning, 2019, p. 11.
[62] X. Wang, Y. Zhang, Q. Li, Y. Chen, J. Han, Open information extraction with meta-pattern discovery in biomedical literature, in: ACM Int. Conf. on Bioinformatics, Computational Biology, and Health Informatics, ACM Press, 2018, pp. 291–300.
[63] I. Yamada, A. Asai, H. Shindo, H. Takeda, Y. Matsumoto, LUKE: deep contextualized entity representations with entity-aware self-attention, in: Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 6442–6454.
[64] S. Lyu, H. Chen, Relation classification with entity type restriction, in: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, Association for Computational Linguistics, 2021, pp. 390–395, Online.
[65] C. Mallart, M. Le Nouy, G. Gravier, P. Sébillot, Active learning for interactive relation extraction in a French newspaper's articles, in: Recent Advances in Natural Language Processing, 2021, pp. 886–894.
[66] C.D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S.J. Bethard, D. McClosky, The Stanford CoreNLP natural language processing toolkit, in: Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2014, pp. 55–60.
[67] G.A. Miller, WordNet: An Electronic Lexical Database, MIT Press, Cambridge, MA, 1998.
[68] P. Shi, J. Lin, Simple BERT Models for Relation Extraction and Semantic Role Labeling, Apr. 2019.