Defining and computing Least Common Subsumers in RDF

Defining and Computing Least Common Subsumers in RDF☆

S. Colucci[a,*], F.M. Donini[b], S. Giannini[a], E. Di Sciascio[a]

[a] *Politecnico di Bari, Via Orabona 4, 70125 Bari, Italy*
[b] *Università della Tuscia, Via S. Carlo 32, 01100 Viterbo, Italy*

Abstract

Several application scenarios in the Web of Data share the need to identify the commonalities between a pair of **RDF** resources. Motivated by such needs, we propose the definition and the computation of Least Common Subsumers (LCSs) in **RDF**. To this aim, we provide some original and fundamental reformulations, to deal with the peculiarities of **RDF**. First, we adapt a few definitions from Graph Theory to paths and connectedness in **RDF**-graphs. Second, we define *rooted* **RDF**-graphs (r-graphs), in order to focus on a particular resource inside an **RDF**-graph. Third, we change the definitions of LCSs originally set up for Description Logics to r-graphs. According to the above reformulations, we investigate the computational properties of LCS in **RDF**, and find a polynomial-time characterization using a form of graph composition. This result remarkably distinguishes LCSs from Entailment in **RDF**, which is an NP-complete graph matching problem. We then devise algorithms for computing an LCS. A prototypical implementation works as a proof-of-concept for the whole approach in three application scenarios, and shows usefulness and feasibility of our proposal. Most of our examples are taken directly from real datasets, and are fully replicable thanks to the fact that the choice about which triples are selected for the computation is made explicit and flexible.

1. Introduction

A *Common Subsumer* (CS) of two formulas ϕ_1, ϕ_2 is another formula ϕ_3 that is logically entailed by both ϕ_1 and ϕ_2. A *Least* Common Subsumer (LCS) is, intuitively, a most specific CS of ϕ_1, ϕ_2—for logics in which the LCS is unique, it entails every other CS of ϕ_1, ϕ_2. Intuitively, a CS expresses some *logical commonalities* of ϕ_1, ϕ_2, and an LCS expresses all of them.

LCSs have been introduced in 1992 [3] in Description Logics (DLs) [4] and adopted in disparate application domains, including inductive learning, bottom-up construction of knowledge bases, information retrieval, among others. All such domains include tasks which share the need to infer and/or compute commonalities between domain items, that may be formally modeled as concepts in DLs knowledge bases.

Apparently, no work has been done so far to support the adoption of such a reasoning service in what is known to be the biggest available and addressable knowledge base: the Web of Data [5]. Although combining different datasets, the Web of Data may in fact be considered as a unique data source in which resources are all modeled in one language, **RDF** [6], whose semantics allows for logical reasoning over them. To the best of our knowledge, the definition and computation of LCSs in **RDF** has never been addressed in the literature, even though the search for commonalities between pairs of **RDF** resources turns out to be useful in several tasks, and has been studied only with statistical or algebraic tools. Hence, the development of a Knowledge Representation service computing an LCS (or even a CS) of two resources in **RDF** could be a useful tool for building Semantic Web applications.

In spite of the apparently low expressiveness of **RDF**, the language has some special features which make it not comparable to any DL. For this reason, developing an idea presented in a preliminary work [1], we propose a novel definition of LCSs, which is able to manage **RDF** assertions and extracts the informative content hidden in resource descriptions. Although we focus on Simple Entailment [6], our definition of LCS can be used also for stronger entailment regimes adopted for interpretation of **RDF** assertions. Moreover, in our im-

☆ This work is an almost completely rewritten version of [1, 2]. Most results are unpublished.

* Principal Corresponding Author

Email addresses: simona.colucci@poliba.it (S. Colucci), donini@unitus.it (F.M. Donini), silvia.giannini@poliba.it (S. Giannini), eugenio.disciascio@poliba.it (E. Di Sciascio)

plementation, we show that also (non-least) Common Subsumers can be useful in several applications. In order to manage the peculiarities of LCSs in **RDF**, we make the original contributions below:

1. we adapt definitions about paths and connectedness from Graph Theory to **RDF**-graphs, taking into account the fact that labels of nodes and arcs can be mixed in (what we call) **RDF**-paths

2. we define *rooted* **RDF**-graphs (r-graphs) $\langle r, T_r \rangle$, that focus on a particular resource r inside an **RDF**-graph, and on a set of chosen triples T_r describing r in the Web of Data

3. we define entailment between r-graphs, that is entailment in which roots must be mapped to roots

4. we change the LCS definitions (originally set up for Description Logics) to r-graphs.

The redefinition of LCSs described above led us to a remarkable result in terms of complexity, which constitutes a main contribution of the article. In particular, we found a polynomial-time characterization of LCS in **RDF**, based on the computational properties we investigate. Such a characterization uses a form of graph composition and distinguishes the computation of an LCS from Entailment in **RDF**, which is known to be an NP-complete graph matching problem. According to this novel polynomial-time characterization, we provide algorithms to compute an LCS.

In order to support our claims and motivate our work, we identified three application scenarios in which LCSs may significantly improve information extraction in the Web of Data: *(i)* entity disambiguation/linking in the Semantic Web, *(ii)* automated clustering of collections of **RDF** resources, and *(iii)* automated extraction of features shared among drugs. In all the three scenarios, we show how the computation of a CS not only provides a semantic approach to similarity between two **RDF** resources, but also a description of their shared features—in other words, the CS does not only show *if* or *how much* two resources are similar, but also *why* they are. In this respect, our work makes applicable to **RDF** several similarity measures based on LCSs often called *semantic similarity* [7, 8, 9].

The article is organized as follows: in the next section, we give some background to make the paper self-contained. Least Common Subsumers are defined in Section 3 and their theoretical properties are investigated in Section 4. Their computation is addressed in Section 5 and exemplified with reference to the three application scenarios in Section 6, which also evaluates performance. The literature related to our work is analyzed in Section 7, before concluding the paper.

2. Background and Notation

We denote by U the set of all URIs, by B the set of all blank nodes, and by L the set of all possible literals, which can be either plain or typed literals. An ***RDF**-graph* is a subset of the set of all possible triples in $(U \cup B) \times (U) \times (U \cup B \cup L)$ while a *generalized **RDF**-graph*—according to ter Horst [10]—adds B to the middle term of the above cartesian product—*i.e.*, generalized **RDF**-graphs allow a blank node to appear also in the predicate position of a triple[1]. When a graph G contains no blank nodes, we say that G is *ground*.

To correctly embed **RDF** triples in English text we use the notation $\ll a\ p\ b \gg$ to mean what in **RDF** documents is written as "*a p b* ." (with the full stop at the end of the triple).

Definition 1. *Given a generalized **RDF**-graph G, we denote by terms(G) the* terms *of G,* i.e.*, the subset of $U \cup B \cup L$ occurring in any position in any triple of G. Moreover, we denote by bl(G) $\subset B$ the set of blank nodes occurring in (some triple of) G.*

Recall that each blank node corresponds to an existentially quantified variable, where the scope of the quantification is the document the blank node occurs in. A blank node without a name is denoted by `[]`, while named blank nodes are prefixed by "`_:`", *e.g.*, `_:xxx`.

Definition 2 ([10, 2.4]). *Given a generalized **RDF**-graph G and a partial function $h : B \to B \cup U \cup L$, we denote by G_h the* instance *of G, obtained by replacing in every triple of G, every blank node b in the domain of h with $h(b)$.*

The semantics of **RDF** is given in terms of set-theoretic interpretations I. Given two partially overlapping sets of *resources* R_I and *properties* P_I, terms in U are mapped by an interpretation function I into $R_I \cup P_I$. Intuitively, terms that occur in triples only as subjects or objects can be mapped into R_I, terms that occur only as properties can be mapped into P_I, while terms that appear *both* as properties and as subjects/objects must be mapped into $R_I \cap P_I$—which explains why R_I and P_I must overlap. Then, another interpretation function—call it I_{ext}—maps elements in P_I to

[1] For our setting, we do not need to consider the even larger definition of generalized **RDF**-graphs in which, *e.g.*, literals in the subject or predicate position are allowed.

subsets of $R_I \times R_I$, that is, pairs of resources. Plain literals are interpreted as themselves, while typed literals are interpreted as (a special kind of) resources. An interpretation I *satisfies* a ground triple $\ll s\ p\ o \gg$ when the pair $(I(s), I(o)) \in I_{ext}(I(p))$, and I satisfies a ground graph G when it satisfies all triples in G. Blank nodes are interpreted by another function $A : B \to U \cup L$, and we can extend it to non-ground graphs saying that $A(G)$ is G where every blank node b has been replaced by $A(b)$ in every triple. Finally, an interpretation I satisfies a non-ground graph G if there exist a function A such that I satisfies the ground graph $A(G)$.

We remark that elements interpreted in $R_I \cap P_I$ make **RDF** an essentially untyped logic. To make **RDF** well-typed, several researchers adopted *punning* [11], in which the different occurrences of a resource (as an individual, a class, or a predicate) are treated as different symbols—that is, $R_I \cap P_I = \emptyset$. However, we do not want to assume punning, for reasons that are clarified in Section 3.1, and embrace the original **RDF** semantics.

The weakest form of entailment between two **RDF**-graphs G and H is Simple Entailment, denoted by $G \models_S H$. Every statement which holds for Simple Entailment holds also for stronger entailment relations, such as **RDF**-Entailment, and **RDF-S**-Entailment. In this paper, we concentrate only on Simple Entailment, which can be characterized in three equivalent ways.

The first definition of Simple Entailment states that $G \models_S H$ if every interpretation satisfying G satisfies also H. Simple Entailment is a transitive relation, *i.e.*, $G_1 \models_S G_2$ and $G_2 \models_S G_3$ implies $G_1 \models_S G_3$.

The second characterization of Simple Entailment says that $G \models_S H$ if and only if there exists a subgraph $G' \subseteq G$ such that G' is an *instance* of H—*i.e.*, there exists a partial mapping $h : B \to B \cup U \cup L$ such that $G' = H_h$. The Interpolation Lemma [12] proves that this characterization of Simple Entailment is indeed equivalent to the one based on interpretations of **RDF**-graphs. The Interpolation Lemma has been extended to generalized **RDF**-graphs too [10, Prop.2.12]. In the proofs of Section 4 and Section 5, we use this characterization of Simple Entailment.

For sake of completeness, we mention that $G \models_S H$ if and only if there is a way of applying Rules **se1** and **se2** shown in Figure 1 to triples in G such that the resulting graph G' contains H as a subgraph. Hence, this statement can be taken as a third, equivalent, characterization of Simple Entailment.

We stress the fact that Simple Entailment is the most general entailment relation in **RDF**, which means that $G \models_S H$ always implies $G \models_{\mathbf{RDF}} H$, which in turn implies $G \models_{\mathbf{RDF\text{-}S}} H$—where the subscript refers to the entailment regime. Hence in the sections about properties of Least Common Subsumers, we discuss also how to extend proofs based on Simple Entailment to stronger entailment relations.

As usual, we define logical equivalence as bi-directional entailment.

Definition 3 (Equivalence). *Two **RDF**-graphs G, H are* equivalent under Simple Entailment *(or simply, equivalent), denoted by $G \equiv_S H$, iff both $G \models_S H$ and $H \models_S G$ hold.*

We now recall two properties of entailment that will be used later on in the proofs of this paper. Since the former property does not appear in the original paper on **RDF** Semantics [12], for sake of completeness we give here its (rather simple) proof.

Proposition 1. *For every pair of **RDF**-graphs G, H and every **RDF**-graph $H' \subseteq H$, if $G \models_S H$ then $G \models_S H'$.*

Proof. From the Subgraph Lemma [12], $H \models_S H'$. If $G \models_S H$, then by transitivity $G \models_S H'$ too. □

Proposition 2 (Monotonicity Lemma [12]). *For every pair of **RDF**-graphs G, H and every **RDF**-graph $G' \supseteq G$, if $G \models_S H$ then $G' \models_S H$.*

The above propositions are intuitive if Simple Entailment is considered as a graph-matching relation between H and a subset of G—namely, enlarging G or restricting H preserves the match.

3. Least Common Subsumers in RDF

3.1. Choosing the triples

No implementation analyzing Web resources extends to analyzing all available triples in the Semantic Web; yet depending on the chosen triples, the analysis may yield different results. Hence, before presenting our formalization about Least Common Subsumers, we also want to discuss how to choose the triples these definitions will be applied on, in our subsequent implementation.

Just to make a concrete example[2], the resource `dbpedia:Chrysler_Building` is the subject of the triple:

```
dbpedia:Chrysler_Building
```

[2] Throughout all the paper, we use Turtle notation [13]. Prefixes are resolved in Table 1. Examples were taken from *DBPedia* on June 2015

$$\textbf{se1}: \frac{u\ p\ v\ .}{u\ p\ b_1\ .} \qquad\qquad \textbf{se2}: \frac{u\ p\ v\ .}{b_2\ p\ v\ .}$$

where b_1 (and b_2) are blank nodes that do not occur in G, and they are used to replace the same resources u, v in every rule application.

Figure 1: Rules for Simple Entailment

Table 1: Table of prefixes

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix dbpprop: <http://dbpedia.org/property/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix www: <http://www.w3.org/ns/prov#> .
@prefix category: <http://dbpedia.org/resource/Category> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix drugbank: <http://wifo5-04.informatik.uni-mannheim.de/drugbank/resource/>
@prefix drugs: <http://wifo5-04.informatik.uni-mannheim.de/drugbank/resource/drugs/>
@prefix drugcategory: <http://wifo5-04.informatik.uni-mannheim.de/drugbank/resource/drugbank/drugCategory/>
@prefix dosageform: <http://wifo5-04.informatik.uni-mannheim.de/drugbank/resource/dosageforms/>
@prefix disease: <http://wifo5-04.informatik.uni-mannheim.de/diseasome/resource/diseases/>
@prefix wikidata:<http://www.wikidata.org/ontology#>
@prefix powder: <www.w3.org/2007/05/powder-s#>
```

```
dbpedia-owl:architecturalStyle
  dbpedia:Art_Deco.
```

but observe that every logical property of `dbpedia:Chrysler_Building`—in our case, the computation of an LCS with some other resource—is influenced by the fact that `dbpedia:Art_Deco` is, in turn, involved as subject in the triple:

```
dbpedia:Art_Deco
 dcterms:subject
   category:20th-century_architectural_styles.
```

and so on. Continuing this line of reasoning, in principle, all triples of the Semantic Web should be considered for the computation of an LCS of two resources. In other terms, one cannot be sure to compute the *Least* Common Subsumer of two resources in the Web of Data unless all present triples are considered. Besides being clearly unfeasible, several studies suggest that such an accumulation of triples from different sources may lead to conclusions that were quite unintended—*e.g.*, the study of Halpin *et al.* [14] for the predicate `owl:sameAs`. Also, our analysis of papers comparing resources (*e.g.*, for similarity) in the Semantic Web [15, 16, 17, 18, 19] reveals that all papers carry out the computation over a selected set of triples. For example, Hulpus *et al.* [16] describe a concept of interest C in *DBPedia* by a so-called *sense graph* rooted in C itself.

Without an explicit statement about how triples are chosen, the results reported by researchers are not replicable. Hence, we believe it is important to make explicit the subgraph of the Semantic Web to be used as a *context* for the computation of an LCS of a resource r.

Our first, general selection criterion is that the triples selected for a resource a should be connected to a in some way. This would seem a trivial requirement; but observe that connected **RDF**-graphs are not "connected" in the usual meaning of Graph Theory: in fact, a resource r is "connected" also to a resource p that appears as *predicate* in a triple $\ll r\ p\ o \gg$ and this can happen recursively. Hence, a path connecting two resources in **RDF**-graphs can mix usual edges with jumps from the label of an arc to the same label, used this time as the source of another arc, as illustrated in Figure 2. Remark that in Figure 2 the connection between r and t is possible only because p is interpreted both as a predicate and a resource—*i.e.*, $I(p) \in R_I \cap P_I$—and that if punning were assumed, the two occurrences of p would be considered as different symbols, making no connection between r and t possible.

These peculiarities of **RDF**-graphs lead us to redefine

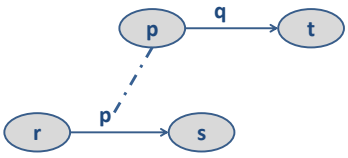

Figure 2: An example of **RDF**-graph in which a resource r is **RDF**-connected to a resource t and not just connected in the usual sense of Graph Theory. This is a quite common pattern in the Web of Data, *e.g.*, when q = `rdfs:subPropertyOf`.

for our purposes the notion of a *path* in **RDF**-graphs.

Definition 4 (RDF-path, -connection, -distance). *Let T be a set of triples.*

- *A resource r is always **RDF**-connected to itself by an **RDF**-path of length 0 (independently of T).*
- *A resource r is **RDF**-connected to another resource p by a path of length $n+1$ if r is connected to a resource s with a path of length n, and either there is a triple $\ll s\ p\ t \gg$, or there is a triple $\ll s\ q\ p \gg$ in T.*

A resource r is **RDF**-connected *to a resource p in T if there is an **RDF**-path from r to p.*

The **RDF**-distance *between two **RDF**-connected resources r and p is the length of the shortest **RDF**-path from r to p.*

Observe that the above definition could be easily extended to *undirected* paths, in which a resource r would be also connected "backwards" to the subjects of the triples r is involved in, *e.g.*, $\ll a\ p\ r \gg$. However, such an extension would only make more complex our definitions and proofs, without changing the properties we are going to prove—*e.g.*, uniqueness or associativity, in the next section. Hence, we defer such an extension after its usefulness is ascertained.

For the rest of this section, we postpone how the triples are selected, and assume that the set of triples T_r has been chosen in some way for each resource r, to concentrate on the main definitions of LCS and its properties. However, in Section 5 and subsequent ones, when we actually compute an LCS, we make explicit our choices about triples by defining (and computing) a *characteristic function* of T_r denoted by σ_{T_r}. This function is built based on the **RDF**-distance, some chosen datasets, and a boolean condition that lets us exclude triples with non-relevant predicates. From now on, we assume that the set of triples T_r associated with a resource r is always computed according to σ_{T_r}. We notice that, although the selection criterion may change with the resource to be modeled, only a single characteristic function can choose the triples T_r for a given resource r. In other words, in order to preserve coherence, our approach does not allow different sets of triples to be cut out for the same resource.

3.2. Rooted graphs and entailment

Our proposal is to attach to a resource r the set of triples T_r used in the computation, and define such a pair as *rooted-graph* of r. To the best of our knowledge, ours is the first proposal in which the triples a computation is based upon, are explicitly represented paired with the resource they are related to.

Definition 5 (Rooted RDF-Graph (r-graph)). *A* Rooted **RDF**-Graph (r-graph *for short) is a pair $\langle r, T_r\rangle$, where:*

1. *r is either the URI of a resource, or a blank node*
2. *T_r is a set of **RDF** triples, such that r is **RDF**-connected to every resource in T_r.*

We immediately distinguish the above definition from the 2005 proposal of Concise Bounded Description[3] (CBD). Starting from a resource r, its CBD collects triples whose subject is r, and then recursively adds triples whose subject is the object of the previous triples, until a triple whose object is not a blank node is reached. Besides the fact that our rooted **RDF**-graphs do not refer to blank nodes, the important difference is that the "properties of properties" (as the resource t in Figure 2) are not included in the CBD of r.

We denote by $G[x \mapsto y]$ the substitution in a graph G of all occurrences of x with y. We now come to entailment between rooted graphs.

Definition 6. *[Rooted Entailment] Let $\langle r, T_r\rangle, \langle s, T_s\rangle$ be two r-graphs. We say that $\langle r, T_r\rangle$* entails *$\langle s, T_s\rangle$—denoted by $\langle r, T_r\rangle \models_S \langle s, T_s\rangle$—in exactly these cases:*

1. *if s is a blank node, then*
   (a) *if r is not a blank node, $T_r \models_S T_s[s \mapsto r]$ must hold;*
   (b) *if also r is a blank node, then $T_r[r \mapsto u] \models_S T_s[s \mapsto u]$ must hold for a new URI u occurring neither in T_r nor in T_s;*
2. *otherwise (*i.e.*, s is not a blank node), if $s = r$, then $T_r \models_S T_s$ must hold.*

[3] `http://www.w3.org/Submission/2005/SUBM-CBD-20050603/`

*In all other cases (*i.e.*, s is not a blank node and $s \neq r$), $\langle r, T_r\rangle$ never entails $\langle s, T_s\rangle$.*

Equivalence $\langle r, T_r\rangle \equiv_S \langle s, T_s\rangle$ is defined accordingly, namely, both $\langle r, T_r\rangle \models_S \langle s, T_s\rangle$ and $\langle s, T_s\rangle \models_S \langle r, T_r\rangle$ must hold.

Intuitively, entailment between r-graphs puts an additional requirement to ordinary entailment: we consider only matches that map the root s of the right-hand side graph into the root r of the left-hand side graph. When s is not a blank node, this happens only if $s = r$ (Case 2 above). When s is a blank node (Case 1), we enforce the mapping between s and r by either substituting s with r, if r is not a blank node, or by substituting both s and r with a new URI, if r is a blank node too.

Making use of the transitivity of substitutions, and transitivity of the entailment relations, one could prove that the relation $\equiv_S$ is an equivalence relation between r-graphs. We do not delve in to the details of this proof, but in order to show how our machinery of rooted entailment works, we give all details of the following statement, which we use later in the paper. Intuitively, it says that rooted entailments can be "merged" preserving rooted-ness.

Proposition 3. *Let $\langle a, T_a\rangle$ be an r-graph, and let $\langle x, G\rangle$, $\langle x, H\rangle$ be two r-graphs whose only shared blank node is x. If both $\langle a, T_a\rangle \models_S \langle x, G\rangle$ and $\langle a, T_a\rangle \models_S \langle x, H\rangle$, then $\langle a, T_a\rangle \models_S \langle x, G \cup H\rangle$.*

Proof. If a is not a blank node, from Case 1a of Definition 6, both $T_a \models_S G[x \mapsto a]$ and $T_a \models_S H[x \mapsto a]$ must hold. We make use of the second characterization of S-entailment: if $T_a \models_S G[x \mapsto a]$, then there is a partial mapping $h1 : B \to B \cup U \cup L$ such that $(G[x \mapsto a])_{h1} \subseteq T_a$, and similarly there is a mapping $h2$ such that $(H[x \mapsto a])_{h2} \subseteq T_a$. Now if a is not a blank node, we can safely define $h := h1 \cup h2$ since the domains of $h1$ and $h2$ do not overlap, and we have $T_a \supseteq (H[x \mapsto a])_{h2} \cup (G[x \mapsto a])_{h1} = ((H \cup G)[x \mapsto a])_h$, which proves $T_a \models_S (G \cup H)[x \mapsto a]$. But this is Case 1a in Definition 6 defininig $\langle a, T_a\rangle \models_S \langle x, G \cup H\rangle$.

If a is a blank node, then Case 1b applies. In this case, both $T_a[a \mapsto u] \models_S G[x \mapsto u]$ and $T_a[a \mapsto u] \models_S H[x \mapsto u]$ must hold, for some URI u occurring in none of T_a, G, H. Again, this means that there are two mappings $h1, h2$ such that both $(G[x \mapsto u])_{h1} \subseteq T_a[a \mapsto u]$ and $(H[x \mapsto u])_{h2} \subseteq T_a[a \mapsto u]$. Defining $h := h1 \cup h2$, we have $(G[x \mapsto u])_{h1} \cup (H[x \mapsto u])_{h2} = ((G \cup H)[x \mapsto u])_h \subseteq T_a[a \mapsto u]$, which proves $\langle a, T_a\rangle \models_S \langle x, G \cup H\rangle$ also in this case. □

Note that we can directly characterize Simple Rooted-Entailment in terms of mappings: $\langle r, T_r\rangle \models_S \langle s, T_s\rangle$ if and only if there exists a mapping h such that $(T_s)_h \subseteq T_r$ (as in Simple Entailment) and moreover, $h(s) = r$, *i.e.*, the root of the r-graph $\langle s, T_s\rangle$ is mapped into the root of the r-graph $\langle r, T_r\rangle$. We use this characterization in the next section.

3.3. (Least) Common Subsumers

We are now ready to give our definition of Common Subsumer of two resources.

Definition 7 (Common Subsumer). *Let $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ be two r-graphs. An r-graph $\langle x, T_x\rangle$ is a Common Subsumer (CS) of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ iff both $\langle a, T_a\rangle \models_S \langle x, T_x\rangle$ and $\langle b, T_b\rangle \models_S \langle x, T_x\rangle$.*

We show later on (Sections 6) that finding a Common Subsumer can be already useful for some application scenarios, such as Clustering of **RDF** resources. Yet, in analogy with the definition in Description Logics [3], we define also Least Common Subsumers of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$, which are CS not entailed by any other (non-equivalent) CS. Thinking the relation $G \models H$ as a partial order $G < H$, Least Common Subsumers are "minimal" CSs w.r.t. such a partial order. However, minimality should be considered modulo equivalence, *i.e.*, when both $G \models H$ and $H \models G$ hold, both G and H represent the same logical properties, up to some renaming of blank nodes.

Definition 8 (Least Common Subsumer). *Let $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ be two r-graphs. An r-graph $\langle x, T_x\rangle$ is a Least Common Subsumer (LCS) of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ iff both conditions below hold:*

1. *$\langle x, T_x\rangle$ is a CS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$;*
2. *for every other CS $\langle y, T_y\rangle$ of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$: if $\langle y, T_y\rangle \models_S \langle x, T_x\rangle$ then $\langle x, T_x\rangle \models_S \langle y, T_y\rangle$, (*i.e.*, $\langle x, T_x\rangle$ and $\langle y, T_y\rangle$ are equivalent under Simple Entailment).*

RDF-graphs are (a special form of) graphs; so, a reader may wonder about the relationship between notions of CS, LCS, and the problems of Common Subgraph and Maximum Common Subgraph in graph problems [20, p.388], that are known to be NP-hard problems. We immediately discuss this issue (and not in the Related Work Section), in order to highlight the differences.

We first observe that nodes in subgraph problems can be freely mapped one into the other, while nodes labeled by an URI in **RDF**-graphs can be mapped only

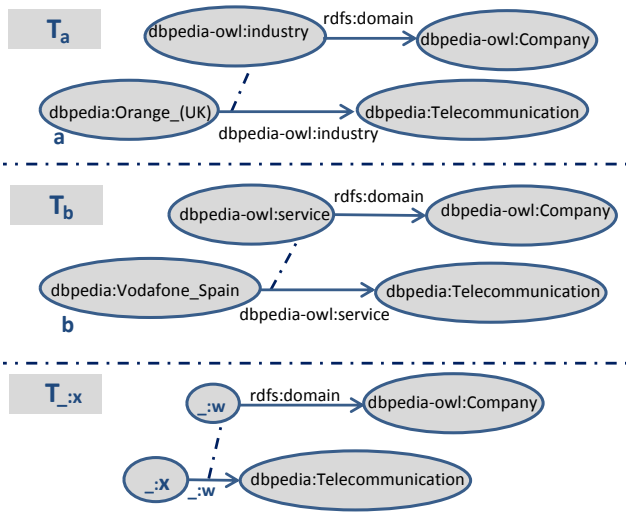

Figure 3: A pair of **RDF**-graphs in DBPedia (November 2015), both instances of the pattern in Figure 2 and one of their LCSs

into nodes with the same URI. So, the two definitions may overlap only when $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$ are all constituted by blank nodes. However, subgraph problems implicitly consider inequality predicates between nodes, that is, two nodes of a subgraph H can never be mapped into one node of a graph G, while **RDF** cannot prevent blank nodes of H to be mapped into a single resource of G. Moreover, an LCS can have many equivalent forms, with more or less blank nodes, while clearly this number is fixed (and maximum) in a Maximum Common Subgraph problem. For these reasons, this problem cannot be put in relation with an LCS, as we prove later on also using a "P *vs.* NP" argument.

We illustrate the peculiarities of LCSs with the aid of an example referred to two concrete r-graphs and one of their LCSs.

Example 1. *Let $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ be two r-graphs, whose sets T_a and T_b are shown in the upper part of Figure 3, where a = `dbpedia:Orange_(UK)` and b = `dbpedia:Vodafone_Spain`. Both r-graphs are instances of the pattern mentioned in Figure 2. The reader may verify that an LCS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ is the r-graph $\langle x, T_x\rangle$, where T_x is shown in the lower part of Figure 3 (for an algorithm computing such an LCS see Section 5). Observe that both a and b have a triple that connects them to* `dbpedia:Telecommunication` *through two different predicates, and this fact is represented in the LCS with the triple*

```
_:x  _:w  dbpedia:Telecommunication .
```

Then, the fact that both predicates share the same domain `dbpedia-owl:Company` *is represented by the triple*

```
_:w  rdfs:domain  dbpedia-owl:Company .
```

In general, a (Least) Common Subsumer of a, b is a blank node, except for the case $a = b$, in which the root of the Least Common Subsumer is the resource itself (see next section).

Proposition 4. *Let $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$ two r-graphs. If $a \neq b$, then for every CS $\langle x, T_x\rangle$ of them, x is a blank node.*

Proof. Suppose an r-graph $\langle c, T_c\rangle$ is a CS of $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$, with $a \neq b$, and that c is not a blank node. Then either $c \neq a$, or $c \neq b$ (or both). Suppose without loss of generality that $c \neq a$: since $\langle c, T_c\rangle$ is a CS of $\langle a, T_a\rangle$, it should be $\langle a, T_a\rangle \models_S \langle c, T_c\rangle$. However, from Definition 6, entailment between such two r-graphs never holds. Hence c must be a blank node. □

Observe that in the above proof, we do not distinguish whether either a or b, or both, are blank nodes. In fact, a triple with a blank node as subject can only entail triples whose subject is a blank node, hence in these cases c must be a blank node too.

As a general remark, we observe that all definitions (Definition 4 through Definition 8) of this section could be stated also for **RDF**-entailment and **RDF-S**-entailment regimes. Proposition 4 holds also for **RDF**- and **RDF-S**-entailment, thanks to the monotonicity of entailment regimes [12].

4. Properties of RDF Least Common Subsumers

In this section we prove some logical and computational properties of LCS in **RDF**. We start by proving uniqueness (up to equivalence), then we prove some algebraic properties, and finally we give a computational characterization, amenable to an implementation.

4.1. Uniqueness of LCS

Starting with uniqueness, we first prove that when one finds two Common Subsumers $\langle x, T_x\rangle$, $\langle y, T_y\rangle$ of two resources $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$, they can always be "merged", either rewriting y as x in T_y, or *vice versa*, producing yet another Common Subsumer, which is at least as specific as the merged ones.

Theorem 1 (Merge of two CS). *Let $\langle x, T_x\rangle$ and $\langle y, T_y\rangle$ be both Common Subsumers of $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$, with*

both x and y blank nodes. Suppose without loss of generality that T_x and T_y do not share any blank node[4]*. Then, both $\langle x, T_x \cup T_y[y \mapsto x]\rangle$ and $\langle y, T_y \cup T_x[x \mapsto y]\rangle$ are CSs of $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$.*

Moreover, $\langle x, T_x \cup T_y[y \mapsto x]\rangle$ and $\langle y, T_y \cup T_x[x \mapsto y]\rangle$ are equivalent r-graphs.

Proof. We prove the first claim only for $\langle x, T_x \cup T_y[y \mapsto x]\rangle$, since the proof for the second r-graph is similar.

From the definition of Common Subsumer, we know that the following four entailments hold:

1. $\langle a, T_a\rangle \models_S \langle x, T_x\rangle$;
2. $\langle b, T_b\rangle \models_S \langle x, T_x\rangle$;
3. $\langle a, T_a\rangle \models_S \langle y, T_y\rangle$;
4. $\langle b, T_b\rangle \models_S \langle y, T_y\rangle$.

Since y is a blank node, entailments no.3–4 hold also when y is replaced by any other blank node, and in particular, when y is replaced by x. In fact, making use of the second characterization of Simple Entailment, it is sufficient to shift the mapping between T_y and T_a [resp., T_b] from y to x.

Hence also the r-graph $\langle x, T_y[y \mapsto x]\rangle$ is entailed by both $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$. Observe that T_x and $T_y[y \mapsto x]$ do not share blank nodes except x. Therefore, from Proposition 3, also $\langle a, T_a\rangle \models_S \langle x, T_x \cup T_y[y \mapsto x]\rangle$ holds.

Regarding the second claim, since T_x and T_y do not share blank nodes, the two set of triples $T_x \cup T_y[y \mapsto x]$ and $T_y \cup T_x[x \mapsto y]$ are just the same set, with x in place of y. Since two **RDF**-graphs that differ just in the names of the blank nodes are equivalent, so are also the r-graphs $\langle x, T_x \cup T_y[y \mapsto x]\rangle$ and $\langle y, T_y \cup T_x[x \mapsto y]\rangle$. □

The above theorem is the basis for proving uniqueness—up to equivalence and renaming—of Least Common Subsumers, since once two CSs are found, one can always construct a new one by just combining them with some renaming. Moreover, the new CS is at least as specific as the merged ones, since $\langle x, T_x \cup T_y[y \mapsto x]\rangle$ entails both $\langle x, T_x\rangle$ and $\langle y, T_y\rangle$.

Corollary 1 (Uniqueness of LCS). *All LCSs of two r-graphs $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$ are equivalent.*

Proof. Let $\langle x, T_x\rangle$ and $\langle y, T_y\rangle$ be two such LCSs. If $a = b$, then the (only) LCS is $\langle a, T_a\rangle$ itself—*i.e.*, $x = y = a$, and the claim is proved (we recall that different sets of triples for the same resource are not allowed—see the end of Section 3.1—that is, if $a = b$, then also $T_a = T_b$). If $a \neq b$, then from Proposition 4 both x and y are blank nodes. Then from Theorem 1 also $\langle x, T_x \cup T_y[y \mapsto x]\rangle$ is a CS of $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$. Moreover, $\langle x, T_x \cup T_y[y \mapsto x]\rangle \models_S \langle x, T_x\rangle$ from Proposition 2. Hence, Condition 2 of Definition 8 applies, imposing that

$$\langle x, T_x \cup T_y[y \mapsto x]\rangle \equiv_S \langle x, T_x\rangle \tag{1}$$

Using the same line of reasoning on $\langle y, T_y\rangle$ and $\langle y, T_y \cup T_x[x \mapsto y]\rangle$, one can show also that

$$\langle y, T_y \cup T_x[x \mapsto y]\rangle \equiv_S \langle y, T_y\rangle \tag{2}$$

Then, from the last statement of Theorem 1, we have that $\langle x, T_x \cup T_y[y \mapsto x]\rangle \equiv_S \langle y, T_y \cup T_x[x \mapsto y]\rangle$. □

Intuitively, the above proofs rely on the possibility of merging **RDF**-graphs, which is a form of logical conjunction plus renaming. Hence, although the above results might not be surprising, we are not aware of any previous paper even mentioning them (to say nothing of the proofs).

Uniqueness of LCS could be proved also for stronger entailment regimes, thanks to the fact that equivalence of two **RDF**-graphs under Simple Entailment implies *e.g.*, **RDF**- and **RDF-S**-equivalence.

4.2. Algebraic properties

The above corollary allows us to talk about *the* Least Common Subsumer—up to equivalence. Hence, we can use the notation LCS($\langle a, T_a\rangle$,$\langle b, T_b\rangle$) to represent any of the equivalent forms of LCS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ in the following algebraic properties.

Theorem 2 (Properties of LCS). *The following equivalences hold:*

Idempotency. *LCS($\langle a, T_a\rangle$, $\langle a, T_a\rangle$) $\equiv_S \langle a, T_a\rangle$*

Commutativity. *LCS($\langle a, T_a\rangle$, $\langle b, T_b\rangle$) $\equiv_S$ LCS($\langle b, T_b\rangle$, $\langle a, T_a\rangle$)*

Associativity. *LCS(LCS($\langle a, T_a\rangle$, $\langle b, T_b\rangle$), $\langle c, T_c\rangle$) $\equiv_S$ LCS($\langle a, T_a\rangle$, LCS($\langle b, T_b\rangle$, $\langle c, T_c\rangle$))*

Proof. Idempotency and Commutativity are direct consequences of Definition 8, hence we concentrate on Associativity. Let $\langle x, T_x\rangle$ be (an equivalent form of) the LCS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$, and let $\langle y, T_y\rangle$ the LCS of $\langle b, T_b\rangle$, $\langle c, T_c\rangle$. Rewriting the above statement, we want to prove that

$$\begin{aligned} &\langle w, T_w\rangle \doteq LCS(\langle x, T_x\rangle, \langle c, T_c\rangle) \equiv_S \\ &\equiv_S LCS(\langle a, T_a\rangle, \langle y, T_y\rangle) \doteq \langle z, T_z\rangle \end{aligned} \tag{3}$$

[4] Shared blank nodes can always be replaced by new ones.

where for convenience we named $\langle w, T_w\rangle$, $\langle z, T_z\rangle$ the two LCSs.

Because entailment is transitive we have that both $\langle w, T_w\rangle$ and $\langle z, T_z\rangle$ in (3) are (non-least) CSs of any pair of r-graphs taken from $\langle a, T_a\rangle$, $\langle b, T_b\rangle$, $\langle c, T_c\rangle$, and in particular, both $\langle w, T_w\rangle$ and $\langle z, T_z\rangle$ are CSs of the pair of r-graphs $\langle a, T_a\rangle$, $\langle c, T_c\rangle$. Applying Theorem 1, also $\langle w, T_w \cup T_z[z \mapsto w]\rangle$ is a CS of $\langle a, T_a\rangle$, $\langle c, T_c\rangle$, and from Proposition 2, the two entailments below both hold:

$$\langle w, T_w \cup T_z[z \mapsto w]\rangle \quad \models_S \quad \langle w, T_w\rangle \tag{4}$$

$$\langle w, T_w \cup T_z[z \mapsto w]\rangle \quad \models_S \quad \langle z, T_z\rangle \tag{5}$$

Since $\langle w, T_w\rangle$ is an LCS of $\langle x, T_x\rangle$, $\langle c, T_c\rangle$, and $\langle w, T_w \cup T_z[z \mapsto w]\rangle$ is a CS of $\langle x, T_x\rangle$, $\langle c, T_c\rangle$ that entails $\langle w, T_w\rangle$, we apply Condition 2 of Definition 8 to (4), and obtain that also $\langle w, T_w\rangle \models_S \langle w, T_w \cup T_z[z \mapsto w]\rangle$, that is, they are equivalent. Analogously, from (5) we can conclude that also $\langle z, T_z\rangle$ is equivalent to $\langle w, T_w \cup T_z[z \mapsto w]\rangle$. By transitivity, we obtain that $\langle w, T_w\rangle \equiv_S \langle z, T_z\rangle$. □

Associativity is important when one wants to compute an LCS of *several* resources—for example, the resources grouped in some cluster obtained by numerical methods. In this case, one could start by computing the LCS $L_2 = LCS(r_1, r_2)$ of an initial pair of resources, then adding one resource r_i at a time one computes a new, broader LCS $L_i = LCS(L_{i-1}, r_i)$, till all resources in the cluster were considered. Associativity grants that no matter which pair one starts with, this computation ends with the same LCS (up to equivalence). This was not ensured by uniqueness of the LCS of a *pair* of resources. Again, we are not aware of any similar, previously published result on **RDF**-graphs.

Also these properties—as for uniqueness—can be easily extended to hold for stronger regimes like **RDF**- and **RDF-S**-entailment, because every extension to Simple Entailment is monotonic—that is, every simple entailment is preserved in stronger entailment regimes.

4.3. Computational Properties

We now characterize the computational complexity of computing Least Common Subsumers. First, we compare pairwise triples from T_a and T_b, and construct a set of triples $T_{a\times b}$ which is quite redundant, but also quite easy to compute. Then we prove that the subset $T_{x_{00}}$ of $T_{a\times b}$ in which triples are **RDF**-connected to the root node x_{00}, is the LCS of T_a and T_b. Since both the computation of $T_{a\times b}$ and the extraction of $T_{x_{00}}$ can be accomplished in polynomial time, we prove that the problem of computing the LCS can be solved in polynomial time. Lastly, we show a property of the **RDF**-connected subset of $T_{a\times b}$ that we will use in the implementation in Section 5.

Definition 9. *Let $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ two r-graphs, with T_a and T_b non-empty sets of triples, and denote the triples of T_a, T_b with the following enumeration*

$$\begin{aligned} T_a &= \{t_i^a = a_i\ p_i\ c_i\ .\ |\ i = 0, 1, \ldots, n\} \\ T_b &= \{t_j^b = b_j\ q_j\ d_j\ .\ |\ j = 0, 1, \ldots, m\} \end{aligned}$$

where $n = |T_a|$ and $m = |T_b|$. Given the assumptions $T_a \neq \emptyset$ and $T_b \neq \emptyset$, it always exists at least one triple in T_a (resp. T_b) whose subject is a (resp. b). From now on, we assume $a_0 = a$ and $b_0 = b$, i.e.*, the subject of the first triple in each enumeration is always the root resource. Notice that all of a_i, p_i, c_i, b_j, q_j, d_j are here* variables*, denoting the actual resources, so the values of many of them in fact coincide. Referring to the operators of Relational Algebra, a_i denotes $\pi_1(t_i^a)$—the projection on the first resource of the i-th triple of T_a—and so on for other variables.*

Let also $X = \{x_{ij}\}$, $Y = \{y_{ij}\}$, $Z = \{z_{ij}\}$ (for $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, m$) be three sets of variables representing values in $U \cup B \cup L$. Define now a function $\tau(\cdot, \cdot)$ from $terms(T_a) \times terms(T_b)$ to $U \cup B \cup L$ as follows (the three lines below are distinguished for the middle variable they yield):

$$\tau(a_i, b_j) = x_{ij} = \begin{cases} a_i \in U & \text{if } a_i = b_j \\ \in B & \text{otherwise} \end{cases} \tag{6}$$

$$\tau(p_i, q_j) = y_{ij} = \begin{cases} p_i \in U & \text{if } p_i = q_j \\ \in B & \text{otherwise} \end{cases} \tag{7}$$

$$\tau(c_i, d_j) = z_{ij} = \begin{cases} c_i \in U \cup L & \text{if } c_i = d_j \\ \in B & \text{otherwise} \end{cases} \tag{8}$$

where we assume that the same blank node is allocated for the same pair of different resources, e.g.*, if $a_2 = a_5$ and $b_3 = b_7$, then also $\tau(a_2, b_3) = x_{23} = \tau(a_5, b_7) = x_{57}$.*

Then, define the set of triples $T_{a\times b}$ as follows:

$$T_{a\times b} = \{x_{ij}\ \ y_{ij}\ \ z_{ij}\ .\ |\quad i = 0, 1, \ldots, n \quad j = 0, 1, \ldots, m\}$$

Clearly, $T_{a\times b} \in O(n \cdot m)$, *i.e.*, it is quadratic w.r.t. the input size. We show through an example what $T_{a\times b}$ looks like, and why a direct use of it seems not to fulfill our intuition about characterizing the LCS.

Example 2. *Consider the following two r-graphs extracted from DBPedia on November 2015 (that intuitively, should have nothing in common)*

$$\langle \texttt{dbpedia:Beam\_me\_up,\_Scotty}, T_a\rangle$$
$$\langle \texttt{dbpedia:Calvary\_(CRT\_station)}, T_b\rangle$$

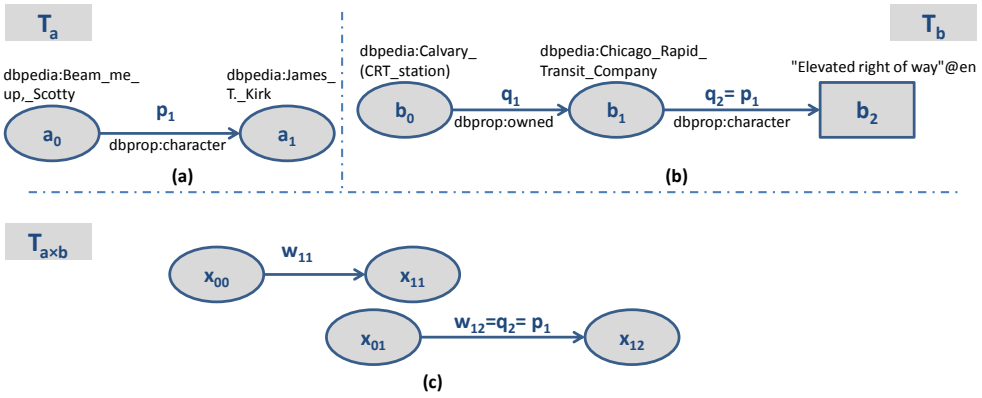

Figure 4: Two r-graphs and the set of triples computed according to Definition 9

depicted in Figure 4 (a-b), respectively. To ease reading, we abbreviate `dbpedia:Beam_me_up,_Scotty` *as "a", and* `dbpedia:Calvary_(CRT_station)` *as "b", so that the composition of the two r-graphs is abbreviated as* $T_{a\times b}$*. The resulting set of triples is shown in Figure 4 (c). The reader can verify that both* $T_a \models_S T_{a\times b}[x_{00} \mapsto a]$ *and* $T_b \models_S T_{a\times b}[x_{00} \mapsto b]$ *hold: in the former case, both triples of* $T_{a\times b}$ *are mapped to the only triple of* T_a*, while in the latter case, the blank nodes* x_{11} *and* x_{01} *are both mapped to* `dbpedia:Chicago_Rapid_Transit_Company`*, forming a single path. If* $T_{a\times b}$ *was taken as a Least Common Subsumer under Simple Entailment, it would mean that the two r-graphs both contain some triple whose predicate is* `dbprop:character` *(triple* $\ll x_{01}\ w_{12}\ x_{12} \gg$ *in* $T_{a\times b}$*). On the other hand, a different intuition says that the two resources have nothing in common, so an LCS of them should include only the first triple* $\ll x_{00}\ w_{11}\ x_{11} \gg$*, which it is composed by blank nodes only—and just says that both resources are subjects of some triple.*

We think $T_{a\times b}$ is not satisfactory as an LCS, because of two different reasons:

1. the first reason is empirical: in the papers analyzing similarities between two resources in **RDF**-graphs, researchers (*e.g.*, Dojchinovski *et al.* [17], Lehmann&Bühmann [19], Zhu *et al.* [21], Zhang *et al.* [22], just to name a few) compare triples at the same **RDF**-distance from the two observed resources, while in this case the second triple of $T_{a\times b}$ in the example above comes from comparing a triple at 0-distance in T_a with a 1-distance triple in T_b;

2. the second reason is more technical: observe that $T_{a\times b}$ is not **RDF**-connected: there is no **RDF**-path from x_{00} to the other subject x_{01}, not even through the predicate. Hence to consider this an LCS, we should not impose connectedness in LCSs. While this would be possible—leading to a formalization different from ours—this would mean that the representation of an LCS is different from those of r-graphs it comes from, making properties like Associativity not expressible.

The above example—which shows that in general $T_{a\times b}$ might be not **RDF**-connected—justifies the next definition, in which only an **RDF**-connected subset of $T_{a\times b}$ is kept.

Definition 10. *Let* $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ *two r-graphs, and let* $T_{a\times b}$ *be the set of triples as in Definition 9. Let* $\langle x_{00}, T_{x_{00}}\rangle$ *be an r-graph whose set of triples is given by the triples in* $T_{a\times b}$ *whose subject is* ***RDF****-connected to* x_{00}*, in formulas:*

$$T_{x_{00}} = \{u\ p\ v\ .\ \in T_{a\times b} \mid \textit{there is an } \textbf{\textit{RDF}}\textit{-path from } x_{00} \textit{ to } u\}$$

We are now ready to prove that $\langle x_{00}, T_{x_{00}}\rangle$ is a representation of the LCS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$.

Theorem 3. $\langle x_{00}, T_{x_{00}}\rangle$ *is an LCS of* $\langle a, T_a\rangle$, $\langle b, T_b\rangle$.

Proof. We first prove that $\langle x_{00}, T_{x_{00}}\rangle$ is a CS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$. Let h_a, h_b be two mappings from $X \cup Y \cup Z$ such that $h_a(x_{ij}) = x_{ij}$ if x_{ij} is not a blank node, and otherwise $h_a(x_{ij}) = a_i$ and $h_b(x_{ij}) = b_j$, and similarly for arguments from $Y \cup Z$.[5] Since by construction

[5] Observe that h_a and h_b are the inverse of τ on the first and on the second argument, respectively, *i.e.*, $h_a(x_{ij}) = h_a(\tau(a_i, b_j)) = a_i$, and $h_b(x_{ij}) = h_b(\tau(a_i, b_j)) = b_j$. Such inverse functions exist because for every pair of different resources a_i, b_j, a different blank node is allocated.

h_a, h_b map triples in $T_{x_{00}}$ back to the triples in T_a, T_b they were built from, $h_a(T_{x_{00}}) \subseteq T_a$, and $h_b(T_{x_{00}}) \subseteq T_b$. Hence, from the second characterization of Simple Entailment, both $T_a \models_S T_{x_{00}}$ and $T_b \models_S T_{x_{00}}$, and since $h_a(x_{00}) = a_0 = a$ and $h_b(x_{00}) = b_0 = b$, also entailment between the r-graphs hold.

We now show that every other set of triples which is a Common Subsumer of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$ is entailed by $\langle x_{00}, T_{x_{00}}\rangle$. Let $\langle w, T_w\rangle$ be a CS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$, where from Proposition 4 if $a \neq b$, then w is a blank node. Since both $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$ entail $\langle w, T_w\rangle$, there are two mappings h_{wa}, h_{wb} such that $h_{wa}(w) = a$ and $h_{wa}(T_w) \subseteq T_a$, and similarly for b. Define a mapping $h_w(\cdot)$ from the resources in T_w to resources in $T_{x_{00}}$ as follows: $h_w(u) = \tau(h_{wa}(u), h_{wb}(u))$, where τ is the function of Definition 9. We now analyze the properties of the mapping h_w. By definition, $h_w(w) = \tau(h_{wa}(w), h_{wb}(w)) = \tau(a, b) = x_{00}$. Moreover, for every triple $\ll e\ f\ g \gg$ in T_w, there must be some triple $\ll a_i\ p_i\ c_i \gg$ in T_a such that $h_{wa}(\ll e\ f\ g \gg) = \ll h_{wa}(e)\ h_{wa}(f)\ h_{wa}(g) \gg = \ll a_i\ p_i\ c_i \gg$, and similarly for some triple $\ll b_j\ q_j\ d_j \gg$ in T_b. Hence, for the same triple, $h_w(\ll e\ f\ g \gg) = \ll h_w(e)\ h_w(f)\ h_w(g) \gg = \ll \tau(a_i, b_j)\ \tau(p_i, q_j)\ \tau(c_i, d_j) \gg = \ll x_{ij}\ y_{ij}\ z_{ij} \gg$, which is a triple in $T_{a\times b}$. Moreover, since $\langle w, T_w\rangle$ is an r-graph, there is an **RDF**-path from w to e, and applying to each resource in the path h_w, there is a path from $h_w(w) = x_{00}$ to $h_w(e) = x_{ij}$ in $T_{a\times b}$, which proves that $\ll x_{ij}\ y_{ij}\ z_{ij} \gg$ belongs also to $T_{x_{00}}$.

Hence, globally $h_w(T_w) \subseteq T_{x_{00}}$, which proves $\langle x_{00}, T_{x_{00}}\rangle \models_S \langle w, T_w\rangle$. From the generality of $\langle w, T_w\rangle$, we can conclude that every r-graph which is a CS of $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$ is entailed by $\langle x_{00}, T_{x_{00}}\rangle$, which proves that $\langle x_{00}, T_{x_{00}}\rangle$ is a Least Common Subsumer of $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$. □

Clearly, computing $T_{a\times b}$ is a task that can be accomplished in time proportional to $|T_a| \cdot |T_b|$, and extracting the subset of triples which are **RDF**-connected to x_{00} can be done by a slight modification of a graph search algorithm[6] in time linear in $|T_{a\times b}|$. This yields a first, rough, computational-complexity upper bound in the computation of the LCS.

Theorem 4. *Given two r-graphs $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$, a representation of their Least Common Subsumer under Simple Entailment can be computed in time $O(|T_a| \cdot |T_b|)$.*

[6] Starting from a triple $\ll x_{00}\ p\ u \gg$, it is sufficient to proceed with a depth-first exploration both on triples whose subject is p, and on triples whose subject is u (see Figure 2).

We stress the fact that the above result clearly distinguishes the computation of an LCS from the computation of Simple Entailment, and hence, from **RDF**- and **RDF-S**-Entailment: while computing these entailments is an NP-complete problem, whose hardness strongly depends on the presence of blank nodes in the entailed **RDF**-graph, computing an LCS is a polynomial-time problem, regardless the presence of blank nodes. An intuition for the reader could be that while computing Simple Entailment involves solving graph homomorphism problems (where a blank node opens all matching choices), computing an LCS is more similar to graphs composition (where the presence of blank nodes is irrelevant). This comment applies also to **RDF**- and **RDF-S**-Entailment, since both are NP-complete problems only because they include Simple Entailment: in fact, when passing from the latter to the formers, the complexity of entailment does not increase over NP, and moreover, if the entailed graph does not contain blank nodes, both **RDF**- and **RDF-S**-Entailment can be solved in polynomial time.

So far, we discussed properties of the LCS that are rather declarative (in contrast with procedural properties): in fact, it would be highly inefficient to first compute all triples in $T_{a\times b}$ and then discard the triples which x_{00} is not **RDF**-connected to. We now highlight a more procedural characterization that leads us, in Section 5, to an efficient implementation of the computation of the LCS.

Theorem 5. *There is an **RDF**-path of length $n \geq 0$ in $T_{a\times b}$ from x_{00} to a resource $z_{ij} = \tau(e_i, f_j)$ if and only if there are both an **RDF**-path of length n in T_a from a_0 to e_i, and an **RDF**-path of length n in T_b from b_0 to f_j.*

Proof. We prove the claim by induction on the length of the paths.

For the base case, we just observe that $x_{00} = \tau(a_0, b_0)$ is **RDF**-connected to itself with an **RDF**-path of length 0, and so are a_0 in T_a and b_0 in T_b, so the claim is true for paths of length 0.

Suppose now that the claim is true for some $m \geq 0$, that is, there is an **RDF**-path of length m from x_{00} to (say) $x_{ij} = \tau(a_i, b_j)$ if and only if there are both an **RDF**-path of length m from a_0 to a_i and an **RDF**-path of length m from b_0 to b_j.

Then, in $T_{a\times b}$ there is an **RDF**-path of length $m + 1$ from x_{00} to w_{ij} iff there exist both (i) an **RDF**-path of length m from x_{00} to some resource x_{ij}, and (ii) a triple $\ll x_{ij}\ y_{ij}\ z_{ij} \gg$ in which w_{ij} is either the predicate or the object of the triple, *i.e.*, either $w_{ij} = y_{ij}$, or $w_{ij} = z_{ij}$. Consider the case $w_{ij} = y_{ij}$ (the other case follows the same line of reasoning): since $\ll x_{ij}\ y_{ij}\ z_{ij} \gg =$

$\ll \tau(a_i, b_j)\ \tau(p_i, q_j)\ \tau(c_i, d_j) \gg$, the triple above exists if and only if there are both a triple $\ll a_i\ p_i\ c_i \gg$ in T_a and a triple $\ll b_j\ q_j\ d_j \gg$ in T_b. Let $p_i = e_i$ and $q_j = f_j$. Now from the inductive hypothesis, (i) holds iff there are both an **RDF**-path of length m from a_0 to a_i, and an **RDF**-path of length m from b_0 to b_j. Taken together, the consequences of (i) and (ii) hold iff there are both an **RDF**-path of length $m + 1$ from a_0 to $p_i = e_i$, and an **RDF**-path of length $m + 1$ from b_0 to $q_j = f_j$, that is, the inductive claim. □

In Section 5, we show how the above property can be used to set up an algorithm that performs a parallel depth-first exploration of T_a and T_b, building $\langle x_{00}, T_{x_{00}} \rangle$.

4.4. Common Subsumers in stronger entailments

Given that the above analysis has been made only for Simple Entailment, the reader may wonder if and how it applies also to stronger entailment regimes. We discuss here some implications for **RDF-S**. First of all, since every extension of Simple Entailment must be monotonic, the LCS presented here is always a (generally non-least) CS of the two r-graphs, under stronger entailments as **RDF-S**. One might think that such a CS is a very generic one, but we now show that some of the deductions that can be drawn in **RDF-S** for both r-graphs are preserved by our LCS. As an example observe again Figure 3: both T_a and T_b, **RDF-S**-entail a similar deduction, namely:

$T_a \models_{\textbf{RDF-S}}$ `dbpedia:Orange_(UK) a dbpedia-owl:Company .`

$T_b \models_{\textbf{RDF-S}}$ `dbpedia:Vodafone_Spain a dbpedia-owl:Company .`

(where the predicate "a" abbreviates `rdf:type`, as usual) because Rule **rdfs2** [6] can be applied—intuively, both resources appear in a triple whose predicate has domain `dbpedia-owl:Company`. Observe that this deduction is kept also in the LCS (least w.r.t. Simple Entailment) that we compute (Figure 3, below), namely

$T_x \models_{\textbf{RDF-S}}$ `_:x a dbpedia-owl:Company .`

by applying the very same Rule **rdfs2** to T_x. More generally, it can be proved that our LCS is also a CS in **RDF-S** that preserves all **RDF-S** deductions whose sequences of rule applications are "common" to T_a and T_b. However, a thorough analysis would make this paper exceed space limits, so we defer the study of LCS in **RDF-S** to future publications.

5. Finding Least Common Subsumers in RDF

In this section, we illustrate how the computation of an LCS of two resources a, b can be carried on. We divide the computation in three algorithms. Algorithm 1 is a wrapper for the other two algoritms. Algorithm 2 makes it explicit how the subsets T_a, T_b are extracted from a much larger set of Semantic Web triples TW. Algorithm 3 performs a recursive exploration of T_a, T_b, while building the set of triples T_{cs} of an LCS $\langle cs, T_{cs} \rangle$ of $\langle a, T_a \rangle$, $\langle b, T_b \rangle$. We detail each algorithm below.

Algorithm 1 takes as input two resources, a and b, together with two integers, n_a and n_b, a union of datasets D, and a boolean function $\phi(\cdot)$. Considering a resource a, the integer n_a is the maximum **RDF**-distance from a of triples to be put in T_a, and the same for b, n_b.

In order to manage cycles possibly occurring in the computation—*i.e.*, while computing a CS of a pair of resources, the same pair reappears in an inner recursive call—Algorithm 1 sets up a global data structure S. Such a data structure collects 3-tuples $[r, s, \langle x, T_x \rangle]$, where r, s are visited pairs of **RDF** resources and $\langle x, T_x \rangle$ is an r-graph.

We note that Algorithm 1 can be run as an *anytime* algorithm: at any time the recursive exploration of T_a,T_b in Row 7 is safely stopped[7] Algorithm 1 returns a (non-least) Common Subsumer $\langle cs, T_{cs} \rangle$, along with the set T_{cs} of commonly entailed triples inferred till the moment of its interrupt. If the exploration of T_a and T_b is completed, Algorithm 1 terminates returning an LCS of $\langle a, T_a \rangle$, $\langle b, T_b \rangle$.

Algorithm 1 calls (in Rows 1–2) the function *compute_σ*, which—following Algorithm 2—computes the characteristic function σ_{T_r}, whose role we discuss now. Given a resource r, to form the r-graph $\langle r, T_r \rangle$ we need to extract the set T_r from a much larger set TW, involving possibly several datasets. Various criteria are possible: for example, the **RDF**-distance of a triple from r (see Section 3), assuming that the nearest triples—the ones whose subject is r itself—are probably the most important, while importance fades with the **RDF**-distance. But also other criteria, like which dataset a triple comes from, or which predicate and/or object is involved in the triple, can be considered and combined. Hence, in order to determine T_r in a modular way, we isolate in Algorithm 2 the choice of its characteristic function $\sigma_{T_r} : TW \rightarrow \{false, true\}$ and let σ_{T_r} be a parameter in the computation of an LCS, that can be changed or tuned according to the application problem. Note that since Algorithm 2 returns a function, the computation it performs is a so-called *higher-order* computation[8]. In order to make a concrete case, in this paper we choose

[7] By "safely" we mean that no new recursive call is made, and the ones on the call stack are regularly closed.

[8] Besides, higher-order computations are quite common in the

Find$LCS(a, n_a, b, n_b, D, \phi)$;
Input : resources a, b, integers $n_a, n_b \geq 0$, union of datasets D, boolean function ϕ
Output: r-graph $\langle cs, T_{cs} \rangle$

1 **let** T_a be defined by $\sigma_{T_a} = compute\_\sigma(a, n_a, D, \phi)$ /* see Algorithm 2 */ ;
2 **let** T_b be defined by $\sigma_{T_b} = compute\_\sigma(b, n_b, D, \phi)$ /* see Algorithm 2 */ ;
3 **let** $S = \emptyset$ be a global set of records [A, B, R] where
4 – field A is a resource occurring in T_a,
5 – field B is a resource occurring in T_b, and
6 – field R is an r-graph;
7 **let** $cs = explore(a, b)$ /* see Algorithm 3 */ ;
8 /* now also S has been filled during the exploration */ ;
9 **let** $T_{cs} = \bigcup\{T_x \mid [r, s, \langle x, T_x \rangle] \in S\}$;
10 **return** $\langle cs, T_{cs} \rangle$;

Algorithm 1: Initialization, start, and final collect of an LCS construction

to select the triples based on (1) their **RDF**-distance n from the root of the r-graph, (2) the dataset they come from, and (3) a boolean function $\phi(t)$ excluding triples matching given patterns[9].

Observe that selecting triples in this way is quite common: *e.g.*, Hulpus *et al.* [16] extract triples from three specific datasets of interest, whose subject is connected to r with a distance less than 2, and which exclude resources belonging to a list of so-called *stop-URIs* provided by the authors[10] and extended by other research works [15]. Nevertheless, our approach has a couple of original distinguishing features: *(i)* the inputs D, n, ϕ of Algorithm 2 allow for stating the same constraints as in the approach by Hulpus *et al.* , but in a parametric fashion: D and n are not set in advance and ϕ may contain a much more general criterion for selecting triples; *(ii)* it deals with r-graphs which are **RDF**-connected and thus, notably, it follows **RDF**-paths stemming from predicates, too.

In order to build an LCS of $\langle a, T_a \rangle$ and $\langle b, T_b \rangle$, Algorithm 1 calls in Row 7 the recursive function *explore*, defined in Algorithm 3.

Web, *e.g.*, Javascript functions which may accept functions as arguments (as the function **foreach** of ECMAScript5) or—as in our case—return as a result a pointer to another function.

[9] Intuitively, the union of datasets D corresponds to the `FROM` part of a SPARQL query, while the function ϕ corresponds to a simple `FILTER` part of it.

[10] http://uimr.deri.ie/sites/StopUris

compute$\_\sigma(r, n, D, \phi)$;
Input : resource r, integer $n \geq 0$, union of datasets D, boolean function ϕ ;
Output: function σ_{T_r}

1 **let** σ_{T_r} = **function** $\sigma_{T_r}(t)$ {
2 **if** ($t \in D$) *and* (r *is* ***RDF****-connected to subject*(t) *with* ***RDF****-distance* $\leq n$) *and* $\phi(t) = true$ **then**
3 **return** true
4 **else**
5 **return** false
6 } ;
7 **return** σ_{T_r};

Algorithm 2: Computation of a simple characteristic function σ_{T_r} defining T_r

Algorithm 3 performs a slight modification of a joint depth-first exploration of T_a and T_b, where the modification takes into account the fact that both r-graphs might not be connected according to classical graph theory, yet they are **RDF**-connected (recall Figure 2). So, for each pair of triples $\ll a_i\ p\ c \gg \in T_a$ and $\ll b_j\ q\ d \gg \in T_b$, Algorithm 3 performs a recursive call over both the pair of resources p and q (Row 12), and the pair c and d (Row 13). All triples inferred during the exploration are added to the set T_x (Row 14) inside the record $[a_i, b_j, \langle x, T_x \rangle]$.

Once the recursive exploration of Algorithm 3 terminates (or it is safely stopped), Algorithm 1 in Rows 9–10 accumulates in T_{cs} all triples stored in records of S, and returns the r-graph $\langle cs, T_{cs} \rangle$.

Example 3. *In order to illustrate the algorithm workflow, we refer to an example pair of* ***RDF*** *resources, a_0 and b_0. For the example, let $n_a = n_b = 3$, let D be some union of datasets and let ϕ be some boolean function, such that the resulting r-graphs, rooted in a_0 and b_0, are the ones shown in the left part of Figure 5. When Find$LCS(a_0, 3, b_0, 3, D, \phi)$ is launched, the r-graphs $\langle a_0, T_{a_0} \rangle$ and $\langle b_0, T_{b_0} \rangle$ are computed according to Algorithm 2 and S is initialized as an empty set (Rows 1–6). Algorithm 3 is then called (Row 7) with parameters (a_0, b_0) and returns the root of an LCS of a_0 and b_0 as a blank node cs. In Table 2, the call stack of Algorithm 3 is detailed, with specific reference to the values of local and global variables in each call. We notice that S is progressively built throughout the execution and reaches its final value at the end of all recursive calls (see last row in Table 2).*

At this stage, Algorithm 1 is able to compute the set T_{cs} according to Row 9. By mapping cs to the

Table 2: Call Stack of Algorithm 3 for Example 3

Stack	**Local Variables Status**								**Status of S**	**Call**
	a_i	b_j	x	T_x	t_1	t_2	y	z		
1	a_0	b_0	x_{00}	$\emptyset$	$\ll a_0\ p_1\ c_1 \gg$	$\ll b_0\ q_1\ c_3 \gg$	y_{11}	z_{11}	$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle]\}$	$y_{11} = explore(p_1, q_1)$
1.1	p_1	q_1	y_{11}	$\emptyset$	$\ll p_1\ p_4\ c_4 \gg$	$\ll q_1\ q_4\ d_4 \gg$	y_{44}	z_{44}	$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \emptyset\rangle]\}$	$y_{44} = explore(p_4, q_4)$
1.1.1	p_4	q_4	y_{44}	$\emptyset$					$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \emptyset\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle]\}$	
1.1	p_1	q_1	y_{11}	$\emptyset$	$\ll p_1\ p_4\ c_4 \gg$	$\ll q_1\ q_4\ d_4 \gg$	y_{44}	z_{44}		$z_{44} = explore(c_4, d_4)$
1.1.2	c_4	d_4	z_{44}	$\emptyset$					$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \emptyset\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle]\}$	
1.1	p_1	q_1	y_{11}	$\{\ll y_{11}\ y_{44}\ z_{44} \gg\}$ (*Row*14)			y_{44}	z_{44}	$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle]\}$	
1	a_0	b_0	x_{00}	$\emptyset$			y_{11}	z_{11}		$z_{11} = explore(c_1, c_3)$
1.2	c_1	c_3	z_{11}	$\emptyset$	$\ll c_1\ p_2\ c_2 \gg$	$\ll c_3\ q_2\ d_2 \gg$	y_{22}	z_{22}	$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle], [c_1, c_3, \langle z_{11}, \emptyset\rangle]\}$	$y_{22} = explore(p_2, q_2)$
1.2.1	p_2	q_2	y_{22}	$\emptyset$					$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle], [c_1, c_3, \langle z_{11}, \emptyset\rangle], [p_2, q_2, \langle y_{22}, \emptyset\rangle]\}$	
1.2	c_1	c_3	z_{11}	$\emptyset$	$\ll c_1\ p_2\ c_2 \gg$	$\ll c_3\ q_2\ d_2 \gg$	y_{22}	z_{22}		$z_{22} = explore(c_2, d_2)$
1.2.2	c_2	d_2	z_{22}	$\emptyset$	$\ll c_2\ p_1\ c_3 \gg$	$\ll d_2\ q_1\ c_3 \gg$	y_{33}	z_{33}	$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle], [c_1, c_3, \langle z_{11}, \emptyset\rangle], [p_2, q_2, \langle y_{22}, \emptyset\rangle], [c_2, d_2, \langle z_{22}, \emptyset\rangle]\}$	$y_{33} = explore(p_1, q_1)$
1.2.2.1	p_1	q_1	$y_{33} = y_{11}$ (Row 2)							
1.2.2	c_2	d_2	z_{22}	$\emptyset$	$\ll c_2\ p_1\ c_3 \gg$	$\ll d_2\ q_1\ c_3 \gg$	y_{33}	z_{33}		$z_{33} = explore(c_3, c_3)$
1.2.2.2	c_3	c_3	c_3 (Row 5)	$\emptyset$					$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle], [c_1, c_3, \langle z_{11}, \emptyset\rangle], [p_2, q_2, \langle y_{22}, \emptyset\rangle], [c_2, d_2, \langle z_{22}, \emptyset\rangle], [c_3, d_3, \langle c_3, \emptyset\rangle]\}$	
1.2.2	c_2	d_2	z_{22}	$\{\ll z_{22}\ y_{11}\ c_3 \gg\}$ (Row 14)			y_{11}	c_3	$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle], [c_1, c_3, \langle z_{11}, \emptyset\rangle], [p_2, q_2, \langle y_{22}, \emptyset\rangle], [c_2, d_2, \langle z_{22}, \{\ll z_{22}\ y_{11}\ c_3 \gg\}\rangle], [c_3, d_3, \langle c_3, \emptyset\rangle]\}$	
1.2	c_1	c_3	z_{11}	$\{\ll z_{11}\ y_{22}\ z_{22} \gg\}$ (Row 14)			y_{22}	z_{22}	$\{[a_0, b_0, \langle x_{00}, \emptyset\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle], [c_1, c_3, \langle z_{11}, \{\ll z_{11}\ y_{22}\ z_{22} \gg\}\rangle], [p_2, q_2, \langle y_{22}, \emptyset\rangle], [c_2, d_2, \langle z_{22}, \{\ll z_{22}\ y_{11}\ c_3 \gg\}\rangle], [c_3, d_3, \langle c_3, \emptyset\rangle]\}$	
1	a_0	b_0	x_{00}	$\{\ll x_{00}\ y_{11}\ z_{11} \gg\}$ (Row 14)			y_{11}	z_{11}	$\{[a_0, b_0, \langle x_{00}, \{\ll x_{00}\ y_{11}\ z_{11} \gg\}\rangle], [p_1, q_1, \langle y_{11}, \{\ll y_{11}\ y_{44}\ z_{44} \gg\}\rangle], [p_4, q_4, \langle y_{44}, \emptyset\rangle], [c_4, d_4, \langle z_{44}, \emptyset\rangle], [c_1, c_3, \langle z_{11}, \{\ll z_{11}\ y_{22}\ z_{22} \gg\}\rangle], [p_2, q_2, \langle y_{22}, \emptyset\rangle], [c_2, d_2, \langle z_{22}, \{\ll z_{22}\ y_{11}\ c_3 \gg\}\rangle], [c_3, d_3, \langle c_3, \emptyset\rangle]\}$	

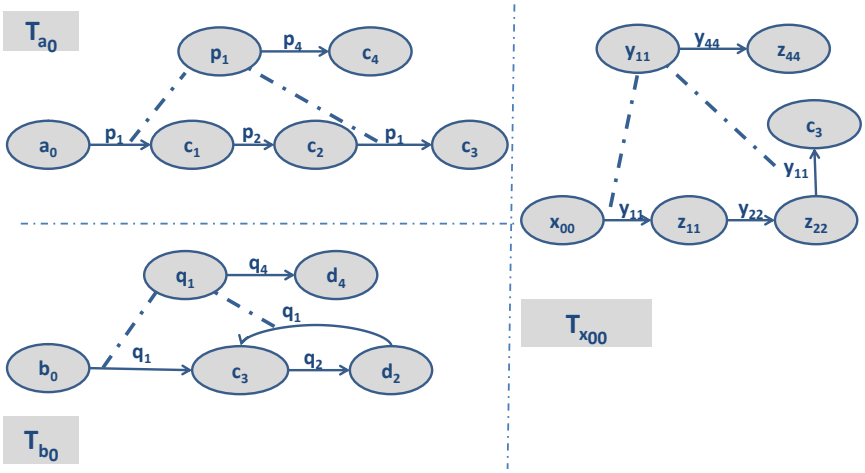

Figure 5: An example pair of **RDF**-graphs to be compared by Algorithm 1 and one of their LCSs $\langle x_{00}, T_{x_{00}} \rangle$.

blank node x_{00} returned by the call explore(a_0, b_0), the reader may verify that $T_{cs} = T_{x_{00}} = \{\ll x_{00}\ y_{11}\ z_{11} \gg, \ll y_{11}\ y_{44}\ z_{44} \gg, \ll z_{11}\ y_{22}\ z_{22} \gg, \ll z_{22}\ y_{11}\ c_3 \gg\}$. The resulting r-graph $\langle x_{00}, T_{x_{00}} \rangle$ (shown in the right part of Figure 5) is the LCS returned by Algorithm 1.

In order to prove that Algorithm 1 computes the LCS of $\langle a, T_a \rangle$, $\langle b, T_b \rangle$, we need the following auxiliary lemmata.

Proposition 5. *Algorithm 3 implements the function $\tau(\cdot, \cdot)$ as it is described in Definition 9, restricted to those pairs of resources a_i, b_j for which a_0 is **RDF**-connected to a_i and b_0 is **RDF**-connected to b_j with two **RDF**-paths of the same length.*

Proof. When the pair of resources a_i, b_j is encountered for the first time in some call of Algorithm 3, either a new blank node is allocated for the pair if $a_i \neq b_j$ (Rows 6–7), or a_i is returned (Rows 4–5). In all subsequent calls, whenever that pair is encountered again (Row 1) the same resource already allocated for that pair is returned. The restriction about **RDF**-paths in the domain of τ is implicit in the way T_a, T_b are jointly explored. □

Without loss of generality, we can assume that the names of the blank nodes assigned by *explore*$(\cdot, \cdot)$ are the same as in Definition 9—if not, some uniform renaming h would map resources in T_{cs} back to $T_{a \times b}$—and in particular, $cs = x_{00}$. Therefore, every triple $\ll x\ y\ z \gg$ added to S at Row 14 is in fact the triple $\ll \tau(a_i, b_j)\ \tau(p, q)\ \tau(c, d) \gg$.

Proposition 6. *Given two r-graphs $\langle a_0, T_{a_0} \rangle$, $\langle b_0, T_{b_0} \rangle$, a resource a_i in T_{a_0} and a resource b_j in T_{b_0}, the following two statements are equivalent:*

1. *there exists $n \geq 0$, such that the recursion stack of explore$(\cdot, \cdot)$ has n pending calls, and Algorithm 3 is called on resources a_i, b_j*
2. *there exists $m \geq 0$ such that there are both an **RDF**-path of length m from a_0 to a_i, and an **RDF**-path of the same length m from b_0 to b_j.*

We omit this proof for brevity, since it is a straightforward induction on the height n of the recursion stack. We just note that we need an integer n for the height of the call stack and a (possibly different) integer m for the length of the **RDF**-path since there might be multiple **RDF**-paths from the roots to a_i in T_{a_0} and b_j in T_{b_0}, and Algorithm 3 does not explore twice the same pair of resources. Hence, there can be two **RDF**-paths of length m in (2), yet in (1), a different pair of **RDF**-paths of length n led Algorithm 3 to issue a call to *explore*(a_i, b_j) on top of n stacked calls.

Theorem 6. *Algorithm 1 computes a representation of the Least Common Subsumer of $\langle a, T_a \rangle$, $\langle b, T_b \rangle$ under Simple Entailment.*

Proof. Since Algorithm 1 just initializes S, T_a, T_b, and collects the sets of triples built by Algorithm 3, we concentrate on the latter algorithm. Although $cs = x_{00}$ (as assumed after Proposition 5), we distinguish here between T_{cs} (the triples built by Algorithm 1) and $T_{x_{00}}$ (the set of triples representing the LCS in Theorem 3),

explore(a_i, b_j);
Input : resource a_i, resource b_j
Output: a resource x
if *for some resource* x_{ij}*, there is already the record* $[a_i, b_j, \langle x_{ij}, T_{x_{ij}}\rangle] \in S$ **then**
  let $x = x_{ij}$
else
  if $a_i = b_j$ **then**
    let $x = a_i$
  else
    let x **be** a new blank node not occurring in S;
  let $T_x = \emptyset$;
  add $[a_i, b_j, \langle x, T_x\rangle]$ to S;
  foreach $t_1 = \ll a_i\ p\ c \gg$ *such that* $\sigma_{T_a}(t_1) = true$ **do**
    foreach $t_2 = \ll b_j\ q\ d \gg$ *such that* $\sigma_{T_b}(t_2) = true$ **do**
      let $y = explore(p, q)$;
      let $z = explore(c, d)$;
      add $\ll x\ y\ z \gg$ to T_x;
return x;

Algorithm 3: Joint depth-first exploration of T_{a_i} and T_{b_j}

and we prove separately that (i) $T_{cs} \subseteq T_{x_{00}}$ and (ii) $T_{x_{00}} \subseteq T_{cs}$.

(i) Proposition 5 implies that Algorithm 3 adds triples $\ll \tau(a_i, b_j)\ \tau(p_i, q_j)\ \tau(c_i, d_j) \gg$ to the records in S, and Proposition 6 proves that there are two **RDF**-paths of equal length, one path from a_0 to a_i in T_a, and another from b_0 to b_j inT_b. The latter statement implies (from Theorem 5) that x_{00} is **RDF**-connected to $\ll \tau(a_i, b_j)\ \tau(p_i, q_j)\ \tau(c_i, d_j) \gg$, hence $T_{cs} \subseteq T_{x_{00}}$.

(ii) We can reverse the above line of reasoning: from Definition 10, every triple $\ll x_{ij}\ w_{ij}\ y_{ij} \gg$ in $T_{x_{00}}$ has a subject x_{ij} such that there is an **RDF**-path from x_{00} to x_{ij}. Then Theorem 5 proves that there are two **RDF**-paths of equal length in T_a and T_b, from a_0 to a_i and from b_0 to b_j, respectively, and finally Proposition 6 proves that the triple $\ll \tau(a_i, b_j)\ \tau(p_i, q_j)\ \tau(c_i, d_j) \gg$ is added to S. Therefore, $T_{x_{00}} \subseteq T_{cs}$. □

We recall that since $\langle x_{00}, T_{x_{00}}\rangle$ is the LCS of $\langle a, T_a\rangle$, $\langle b, T_b\rangle$, under Simple Entailment, which is the weakest entailment relation, $\langle x_{00}, T_{x_{00}}\rangle$ is also a (possibly non-least) Common Subsumer under stronger entailment regimes, like **RDF**- and **RDF-S**-entailment.

6. Proof-of-concept

In this section, we exemplify how CSs could be useful in three possible application scenarios. Although previous sections focused on LCSs, in this section we show that even non-least Common Subsumers (CSs) may be a useful support in the solution of several problems:

- in Section 6.1 we show how to use CSs to model an Named Entity Linking (NEL) and Disambiguation (NED) problem;
- in Section 6.2 we show how CSs may be useful for clustering **RDF** resources;
- in Section 6.3 we present the use case of drugs comparison to provide an evidence of the importance of CSs and show that knowing *what* features two resources have in common is much more important than counting *how many* such features are.

All scenarios share the guidelines below:

- a complete solving approach for the addressed tasks is out of the scope of this article; we just hint how the solution processes may benefit from CSs and we provide a formalization only for the r-graphs to be adopted as inputs and the CSs that could be used to solve the problem;
- the automated extraction of CSs in each task is realized by a Java application implementing the three algorithms of Section 5, adopting Apache Jena libraries[11] for **RDF** data querying and processing;
- for each scenario, we report online at `http://193.204.59.20/rdfcs/` the complete results of the calls to Algorithm 1 both for $n_a = n_b = 1$ and for $n_a = n_b = 2$; in order to ease readability, we here refer to simplified sets of triples describing the two resources to be compared and we explicitly show both such sets and the related results;
- although a different predicate ϕ, selecting relevant triples, may be adopted to specific problems, for a given dataset, ϕ can be set to exclude general *stop-patterns*[12], that match triples not relevant for our analysys. For example, $\ll$ `?s rdf:type rdf:Property` $\gg$ is a stop-pattern. A complete definition of the predicates ϕ we used in all examples can be found at `http://193.204.59.20/rdfcs/`.

[11] http://jena.apache.org/

[12] The name stop-patterns is given in analogy with the previously cited *stop-URIs* [16] and with well-known *stop-words* in Information Retrieval

The Least Common Subsumer of a pair of r-graphs might include a number of triples which are not informative in the application scenarios. We call such triples *uninformative-triples*.The set of uninformative-triples has been determined on the basis of a deep investigation we made on the results provided by Algorithm 1. For instance, in the previous Example 3, one such uninformative-triple is $\ll y_{11}\ y_{44}\ z_{44} \gg$, which adds no informative content to the description of the LCS. In general, all triples of the form $\ll x_{ij}\ y_{ij}\ z_{ij} \gg$ where x_{ij}, y_{ij}, z_{ij} are all blank nodes, and neither y_{ij} nor z_{ij} are subjects of some other triples in T_x may be considered uninformative-triples. We show that also a (non-least) CS, in which uninformative-triples are eliminated, is useful. Hence, we modify Algorithm 1 so that it computes a CS $\langle x, T_x \rangle$ in which T_x contains no uninformative-triple[13]. Triples are discarded recursively based on this criterion, *e.g.*, all triples in chain (meaning that the predicate or the object of a triple is the subject of the next one) are discarded starting from the last one, if they are uninformative-triples. Such a modification substitutes Row 14 in Algorithm 3 with the following row:

if $\ll x\ y\ z \gg$ is not an uninformative-triple
then add $\ll x\ y\ z \gg$ to T_x (9)

6.1. Named Entity Disambiguation

NED in the Semantic Web is the challenge of determining a correct resource out of various candidate ones [23]. In the literature, different kinds of knowledge models have been adopted to resolve disambiguation: ontologies [24], textual information sources like Wikipedia [25], multiple and heterogeneous sources [26, 27]. We refer to the problem of disambiguating **RDF** resources, which arises when trying to export in **RDF** the content of textual documents. This task deals with choosing the resource which best fits a given domain of knowledge, when more than one homonym resource is retrieved. In particular, we propose the use of CSs as a service supporting the selection among candidate entities and adopt the Web of Data as information source for the disambiguation. So far, techniques based on natural language processing [28] or graph similarity [29] have been adopted, but it is our belief that a Knowledge Representation service (although not completely solving the problem by itself) would significantly improve the results of the disambiguation task.

As an example, consider the phrase *"I prefer an orange to a mandarin"*. A typical task in information extraction from unstructured text is determining the correct entities to be associated to the words "orange" and "mandarin", out of all candidate ones (including, *e.g.*, the phone company *Orange Ltd.* and the Chinese dialect *Mandarin*). For the sake of simplicity, let us retrieve in *DBPedia* only two **RDF** resources for each ambiguous word in the sentence[14], reported hereafter:

1. `http://dbpedia.org/resource/Orange_(fruit)`
2. `http://dbpedia.org/resource/Orange_(UK)`
3. `http://dbpedia.org/resource/Mandarin_orange`
4. `http://dbpedia.org/resource/Mandarin_Chinese`

In order to correctly link the two entities, we propose a semantic-based approach relying on CSs of all the four candidate pairs. The following calls to Algorithm 1 are performed:

1–3. FindLCS(`dbpedia:Orange_(fruit)`, n_a, `dbpedia:Mandarin_orange`, n_b, *DBPedia*, ϕ);

1–4. FindLCS(`dbpedia:Orange_(fruit)`, n_a, `dbpedia:Mandarin_Chinese`, n_b, *DBPedia*, ϕ);

2–3. FindLCS(`dbpedia:Orange_(UK)`, n_a, `dbpedia:Mandarin_orange`, n_b, *DBPedia*, ϕ_1);

2–4. FindLCS(`dbpedia:Orange_(UK)`, n_a, `dbpedia:Mandarin_Chinese`, n_b, *DBPedia*, ϕ);

For all of them, we set **RDF**-distances both to $n_a = n_b = 1$ and to $n_a = n_b = 2$, D = *DBPedia* (*i.e.*, the dataset accessible at the SPARQL [30] endpoint `http://dbpedia.org/sparql`), and a predicate ϕ such that, for each triple t, $\phi(t) = true$ if and only if t does not match any pattern in a set of stop-patterns $\mathcal{SP}$, defined at `http://193.204.59.20/rdfcs/disambiguation.html`.

Once the CSs of candidate pairs are computed, some preference relation has to be adopted for choosing the pair resolving NEL. A ranking criterion for CSs, supporting entity disambiguation has been recently proposed [31].

[13] In our current implementation, the set of uninformative-triples includes 16 items.

[14] The reader may verify that many more related URIs are available as of November 2015, instead (see `http://dbpedia.org/page/Orange` and `http://dbpedia.org/page/Mandarin`).

In order to make the paper self-contained, we sketch in the following all the steps involved in the CSs computation, adopting a more compact representation of resources[15]. Moreover, we remark that we are not delving into all details of the disambiguation process, which are out of the scope of this article.

According to Section 3, the candidate resources have to be modeled as rooted graphs (Definition 5) in order to elicit one of their CSs. The possible r-graphs of the four resources are presented in Table 3.

By applying Definition 7 to each pair of r-graphs shown in Table 3, the reader may easily verify that possible CSs are the following ones[16]:

1–3. CS($\langle$`dbpedia:Orange_(fruit)`$, T_{OF}\rangle$, $\langle$`dbpedia:Mandarin_orange`$, T_{MF}\rangle$) = $\langle$ `_` : $cs1, T_{\_:cs1}\rangle$, with `_:cs1` a blank node, and

```
T_:cs1 = _:cs1 rdf:type dbpedia-owl:Species,
                        dbpedia-owl:Eukaryote,
                        dbpedia-owl:Plant;
         dbpedia-owl:family dbpedia:Rutaceae;
         dbpedia-owl:genus dbpedia:Citrus;
         dbpprop:binomialAuthority _:z1.
```

1–4. CS($\langle$`dbpedia:Orange_(fruit)`$, T_{OF}\rangle$, $\langle$`dbpedia:Mandarin_Chinese`$, T_{MC}\rangle$) = $\langle$ `_` : $cs2, \emptyset\rangle$, with `_:cs2` a blank node

2–3. CS($\langle$`dbpedia:Orange_(UK)`$, T_{OC}\rangle$, $\langle$`dbpedia:Mandarin_orange`$, T_{MF}\rangle$) = $\langle$ `_` : $cs3, \emptyset\rangle$, with `_:cs3` a blank node

2–4. CS($\langle$`dbpedia:Orange_(UK)`$, T_{OC}\rangle$, $\langle$`dbpedia:Mandarin_Chinese`$, T_{MC}\rangle$) = $\langle$ `_` : $cs4, \emptyset\rangle$, with `_:cs4` a blank node

Evidently, CS 1–3 makes explicit the common context of resources (`dbpedia:Orange_(fruit)`, `dbpedia:Mandarin_orange`), and it is the most specific one (the results on the complete list of triples confirm this for $n_a = n_b = 1$). We remark that the candidate pairs in the real case are much more than four. Anyway, we here just aim at showing that CSs are a logical basis for extracting common features, and therefore for selection criteria among candidate pairs. Although automated tools for entity recognition and disambiguation do exist in the literature [28], a solution based on CSs would differ from most of them in grounding the disambiguation on a semantic basis, which should reduce (if not eliminate) the number of false positives. In fact, a high-ranked CS of two resources denotes a high sharing of informative triples between them [31], which may never apply to resources not related to each other.

[15]The interested reader is referred to the complete list of results at `http://193.204.59.20/rdfcs/disambiguation.html`

[16]The pattern $\ll$`_:x rdf:type _:z`$\gg$ where `_` : z has no successors belongs to the set of uninformative-triples.

6.2. Clustering

Automated clustering of collections of **RDF** resources has been investigated since 2001 [32], also with specific focus on Semantic Web data [33]. Our target here is hinting how CSs may support automated clustering of collections of **RDF** resources. In particular, the approach we have in mind performs *unsupervised* learning from the resources in the collection. The whole approach relies on the computation of a CS of a randomly selected seed–pair of resources. The CS description is in fact adopted as a search pattern for other resources in the collection sharing the same features as the two in the seed pair. The set made up by the seed pair and the retrieved resources is one cluster of the collection, described in terms of **RDF** triples related to the computed CS. By removing this cluster from the initial collection and iterating the process over the rest of it, a final clustering of the entire collection is reached. We outline that, thanks to its fully deductive nature, the hinted approach retrieves at the same time not only subsets in the collection, but also an abstract representative, in terms of an r-graph $\langle x_{00}, T_{x_{00}}\rangle$, whose $T_{x_{00}}$ models explicitly what all resources of the cluster have in common. The inference of such an abstract representative is completely new in the literature, except for a brief and preliminary introduction in a previous work of ours [2]. Intuitively, the choice of the seed-pair strongly affects the quality of clustering results. A complete approach to clustering should therefore iterate the random selection of the seed-pair and implement a mechanism to compare results and select the clustering with the best quality performance.

We remark that a detailed description and evaluation of the approach to solve clustering is out of the scope of this article. Here, we just use clustering as an application scenario motivating the extraction of CSs in **RDF**.

Consider a collection made up by **RDF** resources describing places of interest. In order to cluster it, the pair made up by the well-known architectural works Eiffel Tower (`http://dbpedia.org/resource/Eiffel_Tower`) and Chrysler Building (`http://dbpedia.org/resource/Chrysler_Building`) is selected as seed. Coherently with Definition 5, we model both resources as r-graphs, $\langle$`dbpedia:Eiffel_Tower`$, T_{ET}\rangle$, and $\langle$`dbpedia:Chrysler_Building`$, T_{CB}\rangle$.

Formally, the following call to Algorithm 1 is performed:
FindLCS(`dbpedia:Eiffel_Tower`, n_a, `dbpedia:Chrysler_Building`, n_b, *DBPedia*, ϕ);
both for $n_a = n_b = 1$ and for $n_a = n_b = 2$. We keep

Table 3: Possible r-graphs for the example resources in Section 6.1.

r-graph	**Set of Relevant Triples**

$\langle$dbpedia:Orange_(fruit), $T_{OF}\rangle$	T_{OF} =
	dbpedia:Orange_(fruit)
	  rdf:type dbpedia-owl:Species,
	  dbpedia-owl:Eukaryote,
	  dbpedia-owl:Plant;
	  dbpedia-owl:family dbpedia:Rutaceae;
	  dbpedia-owl:genus dbpedia:Citrus;
	  dbpprop:binomialAuthority "Osbeck".

$\langle$dbpedia:Orange_(UK), $T_{OC}\rangle$	T_{OC} =
	dbpedia:Orange_(UK)
	  rdf:type dbpedia-owl:Agent,
	  dbpedia-owl:Company,
	  dbpedia-owl:Organisation;
	  dbpprop:industry "Telecommunications";
	  dbpprop:location "Hatfield, England, UK"

$\langle$dbpedia:Mandarin_orange, $T_{MF}\rangle$	T_{MF} =
	dbpedia:Mandarin_orange
	  rdf:type dbpedia-owl:Species,
	  dbpedia-owl:Eukaryote,
	  dbpedia-owl:Plant;
	  dbpedia-owl:family dbpedia:Rutaceae;
	  dbpedia-owl:genus dbpedia:Citrus;
	  dbpprop:binomialAuthority dbpedia:Francisco_Manuel_Blanco.

$\langle$dbpedia:Mandarin_Chinese, $T_{MC}\rangle$	T_{MC} =
	dbpedia:Mandarin_Chinese
	  rdf:type dbpedia-owl:Language;
	  dbpedia-owl:languageFamily dbpedia:Varieties_of_Chinese,
	  dbpedia:Sinitic_languages;
	  dbpprop:ancestor dbpedia:Old_Chinese,
	  dbpedia:Middle_Chinese,
	  dbpedia:Old_Mandarin.

using *DBPedia* in the parameter D and the parameter ϕ defined in Section 6.1.

The complete results of the previous calls are reported on-line[17]. According to the general guidelines, for readability we use the two reduced sets T_{ET} and T_{CB} below:

```
T_ET = dbpedia:Eiffel_Tower
        rdf:type
            dbpedia-owl:ArchitecturalStructure;
        dbpedia-owl:architect
            dbpedia:Stephen_Sauvestre;
        dbpedia-owl:location
            dbpedia:Paris;
        dbprop:title dbpedia:List_of_tallest_
                     buildings_and_structures
                     _in_the_world.

T_CB =dbpedia:Chrysler_Building
        rdf:type
            dbpedia-owl:ArchitecturalStructure;
        dbpedia-owl:architect
             dbpedia:William_Van_Alen;
        dbpedia-owl:location
            dbpedia:Manhattan;
        dbprop:title dbpedia:List_of_tallest_
                     buildings_and_structures
                     _in_the_world,
                     dbpedia:Skyscraper.
```

By applying Definition 7 to our example r-graphs, the reader may easily see that a Common Subsumer of $\langle$`dbpedia:Eiffel_Tower`$, T_{ET}\rangle$ and $\langle$`dbpedia:Chrysler_Building`$, T_{CB}\rangle$ is $\langle$`_:cs1`$, T_{\text{\_:cs1}}\rangle$ where

```
T_:cs1 = _:cs1
        rdf:type dbpedia-owl:ArchitecturalStructure;
        dbpedia-owl:architect _:z1;
        dbpedia-owl:location _:z2;
        dbprop:title dbpedia:List_of_tallest
                     _buildings_and_structures
                     _in_the_world.
```

The description of the extracted CSs may be adopted as a pattern for grouping together resources in the collection of places of interest by means of the following SPARQL query:

```
SELECT DISTINCT ?s WHERE
{?s rdf:type dbpedia-owl:ArchitecturalStructure;
    dbpedia-owl:architect ?z1;
    dbpedia-owl:location ?z2;
    dbprop:title dbpedia:List_of_tallest_
                 buildings_and_structures
                 _in_the_world}
```

The reader may verify that the query returns, as of November 2015, five places of interest which share the same features as the input pair.

The choice of a different seed for clustering would lead to different results. As an example, consider, as further resource, the architectural work Empire Building (`http://dbpedia.org/resource/Empire_Building_(Manhattan)`). We model it as an r-graph $\langle$`dbpedia:Empire_Building_(Manhattan)`$, T_{EB}\rangle$, choosing the following reduced set of triples:

```
T_EB = dbpedia:Empire_Building_(Manhattan)
        dbpedia-owl:architect
            dbpedia:Kimball_&_Thompson;
        dbpedia-owl:location
            dbpedia:Manhattan;
        dbpop:architecture
            dbpedia:Neoclassical_architecture.
```

By exploiting CSs associativity, proved in Theorem 2, we can extract the features shared by the three resources (`dbpedia:Eiffel_Tower`, `dbpedia:Chrysler_Building`, `dbpedia:Empire_Building_(Manhattan)`) by just computing a CS of the r-graphs $\langle$`_:cs1`$, T_{\text{\_:cs1}}\rangle$ and $\langle$`dbpedia:Empire_Building_(Manhattan)`$, T_{EB}\rangle$. The resulting CS[18] is the r-graph $\langle$`_:cs2`$, T_{\text{\_:cs2}}\rangle$, such that

```
T_:cs2 = _:cs2
        dbpedia-owl:architect _:z3;
        dbpedia-owl:location _:z4.
```

Such a CS may be adopted as pattern for clustering, by compiling A SPARQL query analogous to the previous one, but derived from $\_ : cs2$, which is less specific than $\_ : cs1$. As a consequence, the execution of such a query returns about 10.000 results, which include the ones returned by the query compiling $\_ : cs1$.

At this stage of work, we are still not able to provide a full evaluation of clustering results, or a thorough comparison with classical approaches and algorithms [34]. Nevertheless, we remark that even when any of such well known approaches is performed, it is not trivial to exhibit a representative for each cluster. Thanks to associativity, a CS of resources classified in a cluster (by any well-known algorithm) is a cluster representative by itself; therefore, CSs can provide an added-value service for clustering.

6.3. Drugs Comparison

As for the third application scenario, we identified a use case in which the significance of informative content

[17] `http://193.204.59.20/rdfcs/clustering.html`

[18] Results referred to the complete sets of triples are reported online

hidden in a CS is self-evident: the extraction of features shared among drugs. Such a task supports the identification of classes of drugs whose items are somehow substitutable with each other.

In this scenario we refer to the DrugBank Database[19] as of November 2015, a unique bioinformatics and cheminformatics resource that combines detailed drug data (*i.e.*, chemical, pharmacological and pharmaceutical) with comprehensive drug target (*i.e.*, sequence, structure, and pathway) information. The database has been converted in an **RDF** dataset, accessible at `http://wifo5-04.informatik.uni-mannheim.de/drugbank/`, through the D2R Server [35], developed by the Research Group Data and Web Science at the University of Mannheim.

Imagine now a health operator in the need for searching the best candidate drug to substitute the one a patient daily takes: Heparin.

In the addressed dataset, Heparin is identified by the resource `drugs:DB01109`. Now, consider two candidate drugs for substituting Heparin: Ardeparin (`drugs:DB00407`) and Lepidurin (`drugs:DB00001`). The choice of the best candidate drug may be supported by the computation of a CSs of the pairs (`drugs:DB01109`, `drugs:DB00407`) and (`drugs:DB01109`, `drugs:DB00001`). They are obtained performing the calls to Algorithm 1 as follows:

- FindLCS(`drugs:DB01109`, n_a, `drugs:DB00407`, n_b, *DrugBank*, ϕ_3);
- FindLCS(`drugs:DB01109`, n_a, `drugs:DB00001`, n_b, *DrugBank*, ϕ_3).

We performed both calls for $n_a = n_b = 1$ and for $n_a = n_b = 2$ and, as parameter ϕ, we use a predicate ϕ_3 such that, for each triple t, $\phi_3(t) = true$ if and only if t does not match any pattern in a set of stop-patters SP_3, defined at `http://193.204.59.20/rdfCS/drugs.html`. The reader interested in evaluating the complete results of the calls above may refer to the same address. Again, we here show results with reference to simplified sets: let, for the sake of example, *compute_*σ define the sets T_H (for Heparin), T_A (for Ardeparin) and T_L (for Lepidurin) below.

T_H = `drugs:DB01109`
```
drugbank:drugCategory
    drugcategory:anticoagulants,
    drugcategory:fibrinolyticAgents;
drugbank:dosageForm
    dosageform:liquidIrrigation,
    dosageform:solutionIntraperitoneal,
    dosageform:solutionIntravenous,
    dosageform:solutionSubcutaneous;
drugbank:possibleDiseaseTarget
    disease:2210,
    disease:2366,
    disease:2575,
    disease:2695,
    disease:2760,
    disease:345.
```
T_A = `drugs:DB00407`
```
drugbank:drugCategory
    drugcategory:anticoagulants,
    drugcategory:fibrinolyticAgents;
drugbank:dosageForm
    dosageform:liquidIrrigation,
    dosageform:solutionIntraperitoneal,
    dosageform:solutionIntravenous,
    dosageform:solutionSubcutaneous;
drugbank:possibleDiseaseTarget
    disease:1130,
    disease:4042.
```
T_L = `drugs:DB00001`
```
drugbank:drugCategory
    drugcategory:anticoagulants,
    drugcategory:antithromboticAgents,
    drugcategory:fibrinolyticAgents;
drugbank:dosageForm
    dosageform:powderForSolutionIntravenous;
drugbank:possibleDiseaseTarget
    disease:2210,
    disease:2695,
    disease:2760,
    disease:345,
    disease:560,
    disease:592.
```

In order to choose the best candidate drug, we show the CSs of the pairs (`drugs:DB01109`, `drugs:DB00407`) and (`drugs:DB01109`, `drugs:DB00001`) computed by Algorithm 1, if the version of Algorithm 3 modified according to 9 is called at Row 7:

- CS(`drugs:DB01109`, `drugs:DB00407`) =$\langle$`_:cs1`, $T_{\text{\_:cs1}}\rangle$, where `_:cs1` is a blank node and $T_{\text{\_:cs1}}$ = `_:cs1`
```
drugbank:drugCategory
    drugcategory:anticoagulants,
    drugcategory:fibrinolyticAgents;
drugbank:dosageForm
    dosageform:liquidIrrigation,
    dosageform:solutionIntraperitoneal,
    dosageform:solutionIntravenous,
    dosageform:solutionSubcutaneous;
drugbank:possibleDiseaseTarget
    _:z1.
```
- CS(`drugs:DB01109`, `drugs:DB00001`) = $\langle$`_:cs2`, $T_{\text{\_:cs2}}\rangle$, where `_:cs2` is a blank node and $T_{\text{\_:cs2}}$ = `_:cs2`
```
drugbank:drugCategory
```

[19] `http://www.drugbank.ca/`

```
    drugcategory:anticoagulants,
    drugcategory:fibrinolyticAgents;
drugbank:dosageForm
    _:z2;
drugbank:possibleDiseaseTarget
    disease:2210,
    disease:2695,
    disease:2760,
    disease:345.
```

The reader may notice that the two CSs share the same number of triples, so a similarity criterion counting triples would consider Heparin equally similar to both Ardeparin and Lepidurin.

Thus, this example clarifies the importance of inferring a description of the common content of two **RDF** resources, rather than just evaluating their degree of similarity. In particular, let us figure out that the patient mentioned before is affected by Dysprothrombinemia (`disease:345`) and specific conditions force him to take the dosage of Heparin in the form of subcutaneous solution. Even though Lepidurin shares with Heparin Dysprothrombinemia as target disease, it does not include subcutaneous solution among dosage forms.

The computed CSs may be used in a Decision Support System for aiding health operators in deciding which is the best candidate—Ardeparin (`drugs:DB00407`) or Lepidurin (`drugs:DB00001`)—to substitute Heparin (`drugs:DB01109`), on the basis of what features do these two drugs share with Heparin.

Of course, we are not proposing the automated selection of drugs. Nevertheless, also in this case CSs formalize logically knowing "in what" two **RDF** resources are similar, rather than only "how much" they are.

6.4. Implementation: analysis and properties

We give just a hint of our implementation to demonstrate its feasibility in terms of execution times. In order to support the claim of feasibility, we report in Table 4 some information about the execution[20] of some calls, including all the ones described in Subsections 6.1–6.3. In particular, we show:

- cs: the total time for solving each call (whose five inputs a, b, n_a, n_b, D, are reported in the same row, for the sake of completeness);
- σ: the total time for computing T_a and T_b;
- $|T_a|$, $|T_b|$, $|T_{cs}|$: the number of triples[21] in each of the three sets.

When the execution time is high, as in the fourth row, this is because the number of triples of T_a and T_b is quite high from the beginning. Moreover, such times are the worst-case ones, which include the response times of the *DBPedia* SPARQL endpoint. When we used a dump of the triples on our computers, the execution times were two to six times smaller, at the price of using a rapidly obsolete version of *DBPedia*. Hence, there is a tradeoff between up-to-date data and communication overhead. Anyway, since our algorithm is anytime, an execution cutoff can always be set, sacrificing completeness.

Execution times and sizes were significantly reduced by adding as many items as possible to the list of stop-patterns (set $\mathcal{SP}$). In particular, by adopting a list of 43 stop-patterns, we achieved the execution time cs in the tenth row of Table 4, while a previous implementation with only 13 stop-patterns (all stop-URIs) generated cs = 15618156 ms (due to the bigger sizes $|T_a|$ = 1629 and $|T_b|$ = 1720). As a matter of fact, a larger list of stop-patterns reduces also the size of T_{cs}.

The size of the result set T_{cs} is further reduced by discarding uninformative-triples, as previously introduced: we do not add to T_{cs} 16 kinds of triples we identified as useless w.r.t. to the objective of describing features shared by two resources. As a consequence, the size of T_{cs} is significantly reduced. As an example, we refer to call described in the tenth row of Table 4: in a previous implementation including uninformative-triples, the achieved values for $|T_{cs}|$ and cs were 326763 and 11131266 ms, respectively. We believe that this experience can be shared: when passing from restricted and highly controlled datasets (*e.g.*, DrugBank) to DBPedia, general filters on many useless triples are crucial for feasibility. In fact, the reader may verify that the execution times in the examples involving DrugBank are much more feasible for a prototype implementation.

7. Related Work

Recall that we already distinguished LCSs from the problem Maximum Common Subgraph (MCS), by showing in Section 3 that blank nodes can be collapsed or duplicated in an LCS, while they cannot in MCS, and by proving in Section 4 that the two problems belong to different complexity classes (LCS to polynomial-time vs. MCS to NP-complete). Moreover, we already compared our proposal with the ones about how subsets of triples can be selected, in Sections 3 and 5.

To the best of our knowledge, (least or non-least) Common Subsumers of **RDF** resources have never been proposed in the literature so far. There are only some

[20] All tests have been executed on an AMD eight-core server, equipped with a 3.10 GHz processor and 8 GB RAM.

[21] Datasets as of November 2015.

Table 4: Run times and sizes for the CSs computation processed in this section (for the characterization of function ϕ, we refer to Section 6). Run times include response times of the online SPARQL endpoints..

a	b	$n_a = n_b$	D	cs(ms)	$\|T_{cs}\|$	σ(ms)	$\|T_a\|$	$\|T_b\|$
dbpedia:Orange_(fruit)	dbpedia:Mandarin_orange	1	*DBPedia*	1803	26	1471	81	31
dbpedia:Orange_(fruit)	dbpedia:Mandarin_orange	2	*DBPedia*	2753087	2813	38392	920	689
dbpedia:Orange_(fruit)	dbpedia:Mandarin_Chinese	1	*DBPedia*	3308	2	1632	81	79
dbpedia:Orange_(fruit)	dbpedia:Mandarin_Chinese	2	*DBPedia*	41828066	8576	62317	920	1213
dbpedia:Orange_(UK)	dbpedia:Mandarin_orange	1	*DBPedia*	1870	0	1711	47	31
dbpedia:Orange_(UK)	dbpedia:Mandarin_orange	2	*DBPedia*	797924	822	28243	416	689
dbpedia:Orange_(UK)	dbpedia:Mandarin_Chinese	1	*DBPedia*	2221	2	1551	47	79
dbpedia:Orange_(UK)	dbpedia:Mandarin_Chinese	2	*DBPedia*	5185691	2614	45045	416	1213
dbpedia:Empire_Building_(Manhattan)	dbpedia:Chrysler_Building	1	*DBPedia*	2172	15	1568	48	70
dbpedia:Empire_Building_(Manhattan)	dbpedia:Chrysler_Building	2	*DBPedia*	781050	1009	42012	629	769
dbpedia:Eiffel_Tower	dbpedia:Chrysler_Building	1	*DBPedia*	4652	40	2385	95	70
dbpedia:Eiffel_Tower	dbpedia:Chrysler_Building	2	*DBPedia*	19515661	2081	66835	1438	769
drugs:DB01109	drugs:DB00407	1	*DrugBank*	6001	59	2316	139	61
drugs:DB01109	drugs:DB00407	2	*DrugBank*	260788	287	172784	751	123
drugs:DB01109	drugs:DB00001	1	*DrugBank*	6483	56	1805	139	65
drugs:DB01109	drugs:DB00001	2	*DrugBank*	822662	426	184035	751	224

works somehow related to ours, which can be divided into three groups.

*7.1. Similarity among **RDF** graphs*

Already in 2002, Zhu *et al.* [21] proposed to evaluate the similarity between two **RDF** graphs by a measure which combines the similarity between arcs and the similarity between nodes appearing at peer position in the two graphs. Zhu *et al.* decided not to choose MCS algorithms as a tool for evaluating similarity for complexity reasons, as we do—but observe that in addition we proved that MCS is not the right formalization, either. The position of arcs and nodes in **RDF** graphs is referred to the so-called *entries* of the the input graphs, which are the nodes the matching process starts from. The authors propose an application in the clothes market which maps clothes descriptions in **RDF** graphs whose entry is set to the clothes category; on the demand side, the users searching for clothes are asked to give the entries for the **RDF** graphs corresponding to their queries. Both graphs are then indexed in a WordNet category according to their entries. Differently from our proposal, the work by Zhu *et al.* lacks of generality in some aspects: i) entries of graphs to match seem to be not referred to any URI; ii) the criterion for choosing resources representation is subjective, not flexible and not made explicit to the reader.

Grimnes *et al.* [18] spend instead much effort in finding the most suitable representation of a resource as an **RDF** graph, proposing and evaluating three different approaches to the selection of a relevant portion of an **RDF** model including the resource to analyze. This work therefore shares with us the need to find a criterion for cutting out from the Web of Data a subgraph which is representative of the resource. Each of the three approaches adopts its (fixed) criterion: the first one selects only the triples whose subject is the resource of interest; the second one describes resources through their CBD introduced before (see Section 3.2 for a comparison); the third approach selects triples by limiting the sub-graph by depth and traversing the graph two edges forward and one edge backwards from the resource of interest. We notice that our approach to triples selection is more flexible: it manages **RDF**-connected graphs and allows for choosing the depth limit for selection, the datasets of interest and any filtering criterion (through the predicate ϕ). In other words, except for the edge backwards of the third criterion, the subgraph selected by each of the three approaches from Grimnes *et al.* can be easily retrieved by properly setting parameters to the function σ_{T_r} we compute in Algorithm 2.

Very recent proposals address the problem of evaluating similarity between **RDF** graphs through the adoption of metrics which are typical of information theory. In particular, Dojchinovski *et al.* [17] ground their approach to recommendation on the similarity between users, which are described by specifically introduced RDF models, called *resource context graphs* by the authors. Such models are defined as sub-graphs of the graph corresponding to a given dataset, including only nodes within a given distance with respect to resource of analysis. Resources similarity is computed starting from the so-called *shared context* of an input pair, which is a set of resources shared by context graphs of the two input items. The authors make the assumption that i) the more is the information the two resources share, the

more they are similar; ii) better connected shared resources carry more similarity information; iii) less probable shared resources carry more similarity information than the more common ones. As a consequence, their algorithm for measuring similarity adopts also measures to evaluate connectedness and frequency of shared resources. We notice that also Dojchinovski *et al.* share our need to cut out a portion of the Web of Data to operationally compute similarity and propose to adopt an input dataset and graph distance as selecting criterion. Our approach is instead more general in allowing users also to specify a function ϕ that we used for excluding stop-patterns, and in taking into account triples **RDF**-connected to the input resource (see Figure 2). Moreover, Dojchinovski *et al.* consider only nodes in the shared context and therefore do not take the graph structure into account when computing similarity.

Zhang *et al.* [22] point out the need for **RDF**-specific graph matching solutions, which on the one hand overcome the complexity issues of traditional graph theory algorithms and, on the other hand, are able to take into account not only arcs and nodes but also the whole structure of **RDF** graphs when computing graph similarity. The authors therefore propose a multilevel similarity measure, which embeds a component for structure similarity, but reduces the comparison of the so-called query and target graphs to the computation of a numerical value.

D'Amato *et al.* [36] propose a semantic similarity measure for concepts in DLs which takes into account also an underlying ontology. However, the measure specifically proposed applies only to concepts in DLs, which are not comparable to RDF (see next section). This makes the measure by D'Amato *et al.* [36] not suitable for RDF.

As a general comment, since all of the above cited works compute *measures—i.e.*, numbers—none of them is able to extract an **RDF** graph representing the common informative content of the two **RDF** graphs that are compared, as we do with the LCS. Moreover, while all similarity measures limit the comparison to a neighborhood of triples around each resource, such neighborhoods do not consider **RDF**-connection through predicates, and the criteria for the selection—*e.g.*, some kind of distance—are fixed. Our proposal of LCSs may give a logical basis for new, (semantic) similarity measures.

*7.2. Generalizations of **RDF** graphs and Description Logics*

Kernel methods have been proposed [37, 38] to generalize hypotheses from **RDF** graphs. However, when looking to common properties of two graphs, only triples with the *same predicate* are extracted [38, Def.3]. In the highly heterogeneous Web of Data, this constraint makes kernel methods too dependent on how syntactically different, but similar, predicates are used to express knowledge, even in the same dataset. We note that experiments with kernels focus on very controlled datasets—usually translated from databases. It is unclear how kernel methods would behave on loosely structured datasets. For instance, looking back at our example in Figure 3, kernel methods by Lösch *et al.* would extract nothing in common between `dbpedia:Orange_(UK)` and `dbpedia:Vodafone_Spain` because the triples stemming from the two resources use different predicates. Moreover, the paths considered by kernel methods are the usual Graph Theory paths [38, Def.5], which disregard commonalities that stem from other commonalities between predicates: going back to Figure 3, kernel methods do not find the common path between both `dbpedia:Orange_(UK)` and `dbpedia:Vodafone_Spain` on one hand, and `dbpedia-owl:Company` on the other hand.

A completely different approach by Ławrynowicz&Potoniec [39] tries to generalize RDF triples as SPARQL queries, aiming at *supervised* clustering of resources. Such queries have a special variable denoting a resource, and the pair (variable, pattern) might be compared to our r-graphs—although there is no canonical translation between r-graphs and SPARQL queries. We note that while r-graphs can be ordered using entailment, allowing for a definition of minimal element (which we proved is unique, so it is a *least* element), for their queries Ławrynowicz&Potoniec define a generality relation which is not query containment, and for which they do not prove existence of minimal elements—not to say about uniqueness or associativity. When analyzing the algorithm refining queries, *quality* criteria for stopping are established by referring to thresholds. Apart from clustering, a comparison with LCSs is not possible, since SPARQL queries were never used (nor even proposed) as instruments for solving Entity Disambiguation, or drug comparison.

The above comment applies also to the work by Lehmann&Bühmann [19], who model simple SPARQL queries as *query trees* rooted in a resource. Given two query trees T_a and T_b, rooted in a and b respectively, they find another tree that generalizes both queries. If we translate their roots and query trees into r-graphs $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$, it can be proved that the generalization computed by Lehmann&Bühmann is only a CS of $\langle a, T_a\rangle$ and $\langle b, T_b\rangle$; however, it is not an LCS in gen-

eral. The proposed generalization misses three aspects of **RDF** graphs: *(i)* it compares trees and not graphs (*i.e.*, cycles are excluded), *(ii)* it does not consider generalized triples (predicates can never be blank nodes), and *(iii)* the trees are composed only by usual paths and not **RDF**-paths (*i.e.*, a triple whose subject is the predicate of another triple in the tree as in Figure 2 is not considered). On the contrary, the approach by Lehmann&Bühmann bears strong similarities with the definition of LCS in the Description Logic $\mathcal{EL}$ [40]. Hence, the same three differences *(i)–(iii)* can be drawn when comparing our LCS in **RDF** with LCS in $\mathcal{EL}$.

A general comparison between our LCSs and the ones computed in DLs, reveals that LCSs for DLs with the meta-modeling capabilities specific of **RDF** have not been studied yet. On one side, the proposal of extending DLs with higher-order features either focused on query answering only [41] or included only fixpoint constructs [42] that are orthogonal with LCSs in **RDF**. On the other side, existing results on LCSs for (First-Order) DLs—notably, LCSs for the $\mathcal{EL}$ family of DLs [43]—do not include (so far) higher-order features. In general, even when DLs are extended with higher-order features, they assume a strongly typed logic in the sense of Russell&Whitehead, while **RDF** does not (basically, triples forming a cycle in which one of the predicates is `rdf:type` are allowed), and this fact makes any comparison possible only when **RDF** is restricted in some way [44]. Our results on LCS for **RDF**, on the contrary, are valid without any restriction. In fact, we did not revert to the DL-fragment of **RDF/RDF-S**—which would be not significant both for DLs and for **RDF**—but we propose a new **RDF**-specific definition and computation algorithm for Common Subsumers.

7.3. Alternative approaches: discussion

It is also worth discussing whether results about LCSs in DLs could be applied to the First-Order reification of a dataset. Namely, **RDF** can also express a triple $t = \ll a\ p\ c \gg$ by *reifying* the triple as a new individual $\hat{t}$, and then expressing t with the set

$$D_t = \left\{ \begin{array}{l} \hat{t}\ \texttt{rdf:subject}\ a\ . \\ \hat{t}\ \texttt{rdf:predicate}\ p\ . \\ \hat{t}\ \texttt{rdf:object}\ c\ . \end{array} \right\}$$

(to make D_t equivalent to the original triple t, however, one has to use **RDF**-entailment, and not just Simple Entailment). It might be argued that the reified triples could be expressed in some DL knowledge base, and an LCS could be computed with known algorithms. However, a DL expressing reified triples would need at least both inverse roles and individuals—using conventions on DL names, such a DL should be named $\mathcal{ELIO}$. For this DL, only a subsumption algorithm has been devised [45], but LCSs have never been proposed, to the best of our knowledge.

One more chance could be translating an r-graph in logical formulas and adopt approaches using *punning*. As an example, consider the simple r-graph in Figure 2; its translation in logic formulas is $p(r, s) \wedge q(p, t)$. When *punning* can be assumed—*i.e.*, the different occurrences of the resource (as an individual, a class, or a predicate) do not interfere—*contextual* First-Order Logic (FOL) is sufficient to correctly interpret the triples p occurs in [11]. However, observe that considering r connected to t via p in Figure 2 is actually making the different occurrences of p interfere: if we index the first occurrence of p (a predicate) as p_1 and the second occurrence of p (an individual) as p_2, the path exists only because the formula $p_1 = p_2$ is true. Hence, our approach cannot be captured by logically simpler approaches using punning.

8. Conclusion

In this work, we proposed the definition and the computation of Least Common Subsumers in **RDF**. In order to manage the peculiarities of such a reasoning service in **RDF**, we made some original and fundamental reformulations:

- we extended the definitions of path and connectedness from Graph Theory to **RDF**-graphs, taking into account the fact that labels of nodes and arcs can be mixed, producing paths that jump from an arc to a node with same label, as shown in Figure 2;
- we defined *rooted* **RDF**-graphs (r-graphs), in order to focus on a particular resource inside an **RDF**-graph, and adapted **RDF**-entailments to entailments between r-graphs, in which roots must be preserved in the mappings;
- we devised a new definition of Least Common Subsumer of two r-graphs, changing the definition originally set up for Description Logics.

Thanks to such reformulations, we studied the computational properties of computing an LCS, and proved that the problem is not related to Graph Matching, but to a particular form of composition of the two r-graphs, which is computable in polynomial time. This result clearly distinguishes the computation of an LCS from

graph matching problems related to NP-complete Simple Entailment—in spite of the fact that the definition of LCS involves entailment too. We divided the computation of an LCS into three coordinated algorithms, and presented a prototype implementation of our approach, which demonstrates its feasibility in terms of execution times, with reference to the three application scenarios we analyzed: entity disambiguation, unsupervised clustering, and drugs comparison.

So far, we concentrated on Simple Entailment, and just sketched how our results apply, or could be extended, to **RDF**- and **RDF-S**-entailment. The future work will be mainly devoted to the extension of the proposed approach to stronger entailment regimes, and to the analysis of possible heuristics for the selection of relevant triples. Furthermore, each of the suggested motivating scenarios will be investigated in detail.

Acknowledgments

We thank Alex Borgida for helpful discussions and hints on ideas inspiring this article and Tommaso Di Noia for suggesting Named Entity Disambiguation as application scenario.

We acknowledge partial support of Apulia cluster project PERSON, and of Apulia Region initiative "Future in Research".

References

[1] S. Colucci, F. M. Donini, E. Di Sciascio, Common Subsumers in RDF, in: Proceedings of the Thirteenth Conference of the Italian Association for Artificial Intelligence, vol. 8249 of *Lecture Notes in Artificial Intelligence*, Springer, 2013.

[2] S. Colucci, S. Giannini, F. M. Donini, E. Di Sciascio, A deductive approach to the identification and description of clusters in Linked Open Data, in: Proceedings of the 21th European Conference on Artificial Intelligence (ECAI 14), IOS Press, 2014.

[3] W. Cohen, A. Borgida, H. Hirsh, Computing Least Common Subsumers in Description Logics, in: P. Rosenbloom, P. Szolovits (Eds.), Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92), AAAI Press, 754–761, 1992.

[4] F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logic Handbook – 2nd edition, Cambridge University Press, 2007.

[5] N. Shadbolt, W. Hall, T. Berners-Lee, The Semantic Web Revisited, Intelligent Systems, IEEE 21 (3) (2006) 96–101.

[6] P. Hayes, P. F. Patel-Schneider, RDF Semantics, W3C Recommendation, URL `http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`, 2014.

[7] P. Resnik, Using Information Content to Evaluate Semantic Similarity in a Taxonomy, in: Proceedings of the 1995 International Joint Conference on Artificial Intelligence(IJCAI'95), Morgan Kaufmann Publishers Inc., 448–453, 1995.

[8] R. Möller, V. Haarslev, B. Neumann, Semantics-Based Information Retrieval, in: Proceedings of the International Conference on Information Technology and Knowledge Systems (IT&KNOWS '98), Vienna, Budapest, 49–56, 1998.

[9] K. Janowicz, M. Wilkes, M. Lutz, Similarity-Based Information Retrieval and Its Role within Spatial Data Infrastructures, in: T. J. Cova, H. J. Miller, K. Beard, A. U. Frank, M. F. Goodchild (Eds.), Proceedings of the 5th International Conference of Geographic Information Science (GIScience '08), vol. 5266 of *Lecture Notes in Computer Science*, Springer, 151–167, 2008.

[10] H. J. ter Horst, Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary, Journal of Web Semantics 3 (2–3) (2005) 79–115.

[11] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: The Next Step for OWL, Journal of Web Semantics 6 (4) (2008) 309–322.

[12] P. Hayes, RDF Semantics, W3C Recommendation, URL `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`, 2004.

[13] D. Beckett, T. Berners-Lee, Turtle - Terse RDF Triple Language, W3C Team Submission, URL `http://www.w3.org/TeamSubmission/turtle/`, 2011.

[14] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, H. S. Thompson, When owl:sameAs Isn't the Same: An Analysis of Identity in Linked Data, in: P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, B. Glimm (Eds.), Proceedings of the 9th International Semantic Web Conference (ISWC 2010), vol. 6496 of *Lecture Notes in Computer Science*, Springer, 305–320, 2010.

[15] M. Schuhmacher, S. P. Ponzetto, Knowledge-based graph document modeling, in: B. Carterette, F. Diaz, C. Castillo, D. Metzler (Eds.), Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM 2014), ACM, 543–552, 2014.

[16] I. Hulpus, C. Hayes, M. Karnstedt, D. Greene, Unsupervised graph-based topic labelling using dbpedia, in: S. Leonardi, A. Panconesi, P. Ferragina, A. Gionis (Eds.), Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM 2013), ACM, 465–474, 2013.

[17] M. Dojchinovski, T. Vitvar, Personalised Access to Linked Data, in: K. Janowicz, S. Schlobach, P. Lambrix, E. Hyvnen (Eds.), Knowledge Engineering and Knowledge Management, vol. 8876 of *Lecture Notes in Computer Science*, Springer International Publishing, 121–136, 2014.

[18] G. A. Grimnes, P. Edwards, A. D. Preece, Instance Based Clustering of Semantic Web Resources, in: S. Bechhofer, M. Hauswirth, J. Hoffmann, M. Koubarakis (Eds.), Proceedings of the Fifth European Semantic Web Conference (ESWC 2008), vol. 5021 of *Lecture Notes in Computer Science*, Springer, 303–317, 2008.

[19] J. Lehmann, L. Bühmann, AutoSPARQL: Let Users Query Your Knowledge Base, in: Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), vol. 6643 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 63–79, 2011.

[20] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and approximation, Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[21] H. Zhu, J. Zhong, J. Li, Y. Yu, An Approach for Semantic Search by Matching RDF Graphs., in: S. M. Haller, G. Simmons (Eds.), Proceedings of the 15th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2002), AAAI Press, 450–454, 2002.

[22] D. Zhang, T. Song, J. He, X. Shi, Y. Dong, A Similarity-Oriented RDF Graph Matching Algorithm for Ranking Linked Data, in: Proceedings of the 2012 IEEE 12th International Con-

ference on Computer and Information Technology (CIT'12), IEEE Computer Society, 427–434, 2012.

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

[24] J. Hassell, B. Aleman-Meza, I. B. Arpinar, Ontology-driven Automatic Entity Disambiguation in Unstructured Text, in: Proceedings of the 5th International Conference on The Semantic Web, ISWC'06, Springer-Verlag, Berlin, Heidelberg, 44–57, 2006.

[25] S. Cucerzan, Large-Scale Named Entity Disambiguation Based on Wikipedia Data, in: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), Association for Computational Linguistics, 708–716, 2007.

[26] X. Han, J. Zhao, Structural Semantic Relatedness: A Knowledge-based Method to Named Entity Disambiguation, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, ACL '10, Association for Computational Linguistics, 50–59, 2010.

[27] H. Nguyen, T. Cao, Named entity disambiguation on an ontology enriched by Wikipedia, in: Proceedings of IEEE International Conference on Research, Innovation and Vision for the Future, 2008 (RIVF 2008), 247–254, 2008.

[28] G. Rizzo, R. Troncy, NERD: A Framework for Unifying Named Entity Recognition and Disambiguation Extraction Tools, in: Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics, EACL '12, Association for Computational Linguistics, 73–76, 2012.

[29] P. Cudré-Mauroux, P. Haghani, M. Jost, K. Aberer, H. De Meer, idMesh: Graph-based Disambiguation of Linked Data, in: Proceedings of the 18th International Conference on World Wide Web, WWW '09, ACM, 591–600, 2009.

[30] M. Arenas, J. Pérez, Querying semantic web data with SPARQL, in: Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '11, ACM, 305–316, 2011.

[31] S. Giannini, S. Colucci, F. M. Donini, E. Di Sciascio, A Logic-based approach to Named-Entity Disambiguation in the Web of Data, in: Proceedings of the Fourteenth Conference of the Italian Association for Artificial Intelligence, Lecture Notes in Computer Science, Springer, 2015.

[32] A. Delteil, C. Faron-Zucker, R. Dieng, Learning Ontologies from RDF annotations, in: A. Maedche, S. Staab, C. Nedellec, E. H. Hovy (Eds.), Proceedings of the Second Workshop on Ontology Learning (in conjunction with the 17th International Conference on Artificial Intelligence (IJCAI 2001)), vol. 38 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2001.

[33] G. A. Grimnes, P. Edwards, A. Preece, Instance based clustering of semantic web resources, in: Proceedings of the Fifth European Semantic Web Conference (ESWC 2008), Springer, 303–317, 2008.

[34] A. K. Jain, R. C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN 0-13-022278-X, 1988.

[35] C. Bizer, R. Cyganiak, D2R Server Publishing Relational Databases on the Semantic Web, Poster at the 5th International Semantic Web Conference (ISWC2006), 2006.

[36] C. d'Amato, S. Staab, N. Fanizzi, On the Influence of Description Logics Ontologies on Conceptual Similarity, in: A. Gangemi, J. Euzenat (Eds.), Knowledge Engineering: Practice and Patterns, vol. 5268 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 48–63, 2008.

[37] V. Bicer, T. Tran, A. Gossen, Relational Kernel Machines for Learning from Graph-Structured RDF Data, in: G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer, J. Pan (Eds.), The Semantic Web: Research and Applications, vol. 6643 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 47–62, URL `http://dx.doi.org/10.1007/978-3-642-21034-1_4`, 2011.

[38] U. Lösch, S. Bloehdorn, A. Rettinger, Graph Kernels for RDF Data, in: E. Simperl, P. Cimiano, A. Polleres, O. Corcho, V. Presutti (Eds.), The Semantic Web: Research and Applications, vol. 7295 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 134–148, URL `http://dx.doi.org/10.1007/978-3-642-30284-8_16`, 2012.

[39] A. Lawrynowicz, J. Potoniec, Pattern Based Feature Construction in Semantic Data Mining, International Journal of Semantic Web Information Systems 10 (1) (2014) 27–65.

[40] R. Penaloza, A.-Y. Turhan, A Practical Approach for Computing Generalization Inferences in $\mathcal{EL}$, in: Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), vol. 6643 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 410–423, 2011.

[41] G. D. Giacomo, M. Lenzerini, R. Rosati, Higher-Order Description Logics for Domain Metamodeling, in: W. Burgard, D. Roth (Eds.), Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence(AAAI 2011), AAAI Press, San Francisco, California, USA, August 7-11, 2011, 2011.

[42] C. Lutz, R. Piro, F. Wolter, Enriching [Escr][Lscr]-Concepts with Greatest Fixpoints, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 10), vol. 215 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 41–46, 2010.

[43] A. Ecke, A. Turhan, Role-depth Bounded Least Common Subsumers for EL+ and ELI, in: Y. Kazakov, D. Lembo, F. Wolter (Eds.), Proceedings of the 2012 International Workshop on Description Logics, vol. 846 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2012.

[44] J. Z. Pan, I. Horrocks, RDFS(FA): Connecting RDF(S) and OWL DL, IEEE Transactions on Knowledge and Data Engineering 19 (2) (2007) 192–206.

[45] F. Baader, R. Molitor, S. Tobies, Tractable and Decidable Fragments of Conceptual Graphs, in: W. M. Tepfenhart, W. R. Cyre (Eds.), Conceptual Structures: Standards and Practices. Proceedings of the 7th International Conference on Conceptual Structures (ICCS '99), 480–493, 1999.