A Human-readable Explanation for the Similarity of RDF Resources

Simona Colucci[1,*], Francesco M. Donini[2] and Eugenio Di Sciascio[3]

[1]Politecnico di Bari,Via Orabona, 4, Bari, 70125, Italy
[2]Università della Tuscia, Via Santa Maria in Gradi, 4, 01100, Viterbo,Italy
[3]Politecnico di Bari,Via Orabona, 4, Bari, 70125, Italy

Abstract

Evaluating the similarity of RDF resources is nowadays a thoroughly investigated research problem, with reference to a variety of contexts. In fact, several tools are available for the comparison of pairs and/or groups of resources in a knowledge graph, mostly based on machine learning techniques. Unfortunately such tools, though extensively tested and fully scalable, return non-explainable (often numerical) similarity results also when comparing RDF resources, treating them according to their vector embeddings. and making no use of the semantic information carried by RDF triples. In this work, we propose a tool able to compute the commonalities of compared resource and explain them through a text in English, produced by a Natural Language Generation approach. The proposed approach is logic-based and is grounded on the computation of the Least Common Subsumer (re)defined in RDF. The feasibility of the tool is demonstrated with reference to the similarity of Twitter accounts.

Keywords

Explainable Artificial Intelligence (XAI), Resource Description Framework (RDF), Least Common Subsumer (LCS), Natural Language Generation (NLG)

1. Introduction

Learning techniques based on Neural Networks or other forms of numerical vectors comparison have recently shown their effectiveness in producing tools for assessing similarity between complex objects and for clustering them according to (possibly unsupervised) criteria. These tools are nowadays also applied to data coming from **RDF** repositories [1], with the advantage of using widely tested, off-the-shelf tools, but with the disadvantage of obtaining a result whose explanation, if any, cannot be presented to end users without deep mathematical knowledge.

Yet **RDF**, as part of the Semantic Web effort, was equipped from its birth with a logical semantics [2] based on mappings from IRIs and literals into a set of resources. Such semantics allows us to describe the *symbolic* commonalities between resources as a logical formula in the form of an **RDF** graph, named *Least Common Subsumer* (LCS) [3] for its similarity to LCS in Description Logics [4].

XAI.it 2022 - Italian Workshop on Explainable Artificial Intelligence
*Corresponding author.
simona.colucci@poliba.it (S. Colucci); donini@unitus.it (F. M. Donini); eugenio.disciascio@poliba.it (E. Di Sciascio)
0000-0002-8774-4816 (S. Colucci); 0000-0003-0284-9625 (F. M. Donini); 0000-0002-5484-9945 (E. Di Sciascio)

We present a Natural Language Generation (NLG) tool, based on LCS for **RDF** data, that describes in English sentences the commonalities of **RDF** resources previously clustered, or declared similar, by any other non-transparent tool. The connectivity of **RDF**-graphs—objects/predicates of a triple being subjects of other triples—leads to non-trivial relative sentences in English, rendered with relative pronouns. Blank nodes are fundamental in our common subsumers to represent partial commonalities in **RDF** paths from clustered resources. We use relative pronouns in relative sentences to represent them. To our knowledge, there are no other tools that also accept blank nodes in the verbalization of **RDF** graphs. This tool has been already implemented for the comparison of drugs and of contracting processes [5]. We here present the NLG approach and the full set of algorithms at the basis of the tool, together with an application to the evaluation of similarity of Twitter accounts.

The paper is organized as follows: in the next section, we discuss related works and how different our approach is from them. In Section 3 we briefly summarize the necessary notions regarding LCS in **RDF**. Then in Section 4 we explain in detail how we construct (possibly relative) sentences from Common Subsumers. In Section 5 we show examples of the results of our tool when applied to similar Twitter accounts. The final section concludes the paper.

2. Related Work

The explanation of clustering results has been addressed since 1980 in the conceptual clustering research field [6]. Conceptual clustering is the problem of returning, together with clustering results, a concept explaining the criterion for resources aggregation.

So far, this problem has attracted several researchers, whose most influential proposals have been recently reviewed, for the interested reader, by Pérez-Suárez *et al.* [7]. Notably, this review does not include any approach dealing with **RDF** resources. To the best of our knowledge, the only approach dealing with the conceptual clustering of **RDF** resources is the one by Colucci *et al.* [8], based on LCS computation.

In this paper, similarly to conceptual clustering, we aim at explaining clustering results, but we do not cope with the problem of grouping resources. In fact, we propose a logic-based methodology to explain the commonalities of **RDF** resources grouped by any aggregation criterion. In other words, our methodology is agnostic w.r.t. the aggregation criterion.

A similar attitude is shared by Dedalo [9], an approach based on Inductive Logic Programming (ILP) that automatically produces explanations for given clusters using Linked Data as background knowledge. Dedalo evaluates possible explanation hypothesis according to specifically investigated measures and heuristics.

Differently from Dedalo, we take a deductive approach to the identification of cluster commonalities. In particular, given a group of resources returned in the same cluster, the approach computes their LCS,*i.e.*, a set ot triples that abstracts their commonalities. The **RDF** triples in the LCS are then parsed to produce a human-readable explanation of their content.

Thus, our approach falls in the widely populated field of Natural Language Generation (NLG) from the Semantic Web. Since 2014, Bouayad-Agha *et al.* [10] classified at least 11 NLG approaches working on **RDF** graphs. Bouayad-Agha *et al.* adopt several features for classification, including the part of the input graph to verbalize (verbalization request) and the

information to return (communicative goal).

None of the reviewed approaches is able to manage anonymous resources or to generate text from derived triples (not explicitly stated in **RDF**).

In recent years, NLG has mainly focused on the improvement of the readability of generated language, measured through the setting of common baselines. The WebNLG challenge [11], for example, provides a benchmark corpus of English (and also Russian in its second edition [12]) sentences verbalizing RDF triples. The challenge classifies competitors according to the performance in the generation of baseline sentences; no competition is set around the proposal of forms of verbalization lending to richer explanation.

The traditional trend of NLG approaches has been the usage of rules and templates, which represent solutions highly domain-dependent and demanding manual intervention.

A recent shift to this trend has been made possible by deep learning, thanks to neural network-based NLG models. As an example, Neural Wikipedian [13] employs the Sequence to Sequence (SEQ2SEQ) framework [14] (a neural network-based NLG model) to generate summaries of RDF triples. Similarly, Li *et al.*[15] propose the Neural Entity Summarization approach.

Both summarization approaches collect only triples that are already present in the **RDF** descriptions, without handling blank nodes.

Our template-based method for explaining **RDF** similarity has several distinguishing features w.r.t. the approaches above. To the best of our knowledge, it is the only one able to manage blank nodes, that are crucial for the purpose of abstracting several triples with common predicate/object. Even more importantly, our method uses blank nodes to chain triples reaching the same known object (see the example about Twitter accounts in Sect.5). Lastly, our method does not explain just trivial sets of triples, but **RDF** graphs logically computed to summarize the commonalities shared by groups of resources. This makes really significant the informative potential of the returned explanation, which is double-tied to the logic-based nature of computation.

3. LCS in RDF

We assume that the reader is familiar with **RDF** triples and graphs, but to make this paper self-contained, we briefly recall here the definition of [Least] Common Subsumers (LCS) in the context of **RDF** [16, 3], along with the necessary preliminaries.

First, given an **RDF**-graph T_r containing Resource r as a node, we denote a *rooted **RDF**-graph* (in brief *r-graph*) by the pair $\langle r, T_r \rangle$. This allows us to compare Resources r, s by referring to their respective r-graphs $\langle r, T_r \rangle, \langle s, T_s \rangle$.

Second, let $G[s \to t]$ be the graph obtained by substituting each occurrence of s with t in G; now the definition of Simple Entailment $T_r \models T_s$ [2] between two RDF-graphs T_r, T_s, can be restricted to r-graphs as below (rephrased from a previous publication [3, Def.6]):

Definition 1. *[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 $\langle r, T_r \rangle \models \langle s, T_s \rangle$—in exactly the cases summarized below:*

Conditions	s *is a blank node*	s *is not a blank node*
r *is a blank node*	$T_r[r \mapsto u] \models T_s[s \mapsto u]$ *for a new URI* u *occurring neither in* T_r *nor in* T_s	*No entailment*
r *is not a blank node*	$T_r \models T_s[s \mapsto r]$	$s = r$, *and* $T_r \models T_s$

Intuitively, Rooted Entailment restricts Simple Entailment by requiring that the root of one graph be mapped to the root of the other. When both resources r, s are URI, this is possible only when $s = r$ (2nd row, 2nd column), while when either r or s is a blank node (all other entries), a suitable substitution is necessary to obtain the mapping before checking Simple Entailment.

Rooted Entailment is needed to define the notion of Common Subsumer (CS) of two r-graphs $\langle a, T_a\rangle$, $\langle b, T_b\rangle$:

Definition 2 (Common Subsumer, [3, Def.7]). *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* $\langle a, T_a\rangle \models \langle x, T_x\rangle$ *and* $\langle b, T_b\rangle \models \langle x, T_x\rangle$.

Finally, a Least Common Subsumer (LCS) of two RDF resources can be defined as follows:

Definition 3 (Least Common Subsumer [3, Def.8]). *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 $\langle y, T_y\rangle \models \langle x, T_x\rangle$ *then* $\langle x, T_x\rangle \models \langle y, T_y\rangle$*, (*i.e.*,* $\langle x, T_x\rangle$ *and* $\langle y, T_y\rangle$ *are equivalent under Rooted Entailment).*

Colucci *et al.*[3] proved that an LCS of two r-graphs is unique—up to blank renaming—so we can talk about "the" LCS. Moreover, the LCS enjoys the properties of Idempotency, Commutativity, and Associativity. The latter written in formulas is
$LCS(\langle a, T_a\rangle, LCS(\langle b, T_b\rangle, \langle c, T_c\rangle)) =$
$LCS(LCS(\langle a, T_a\rangle, \langle b, T_b\rangle), \langle c, T_c\rangle)$.
Associativity relies on a fundamental property of LCSs as proposed by Colucci *et al.*, namely, the LCS of two r-graphs is itself an r-graph, so it can be used as an argument of another LCS operation with a third r-graph, etc. Associativity ensures that the order in which resources are taken—when computing the LCS of all of them—does not matter.

We also recall some definitions adapting the basic notions of Graph Theory to **RDF**-graphs [3], used in the rest of the paper. First, an ***RDF-path*** from r to s is a sequence of triples $t_1, \ldots, t_n$ in which the subject of t_1 is r, *either* the predicate *or* the object of t_n is s, and for $i = 1, ..., n-1$, either the predicate or the object of t_i is the subject of t_{i+1}. Basically, an **RDF**-path differs from usual graph paths in that it can pass through the resource p in the predicate (the arc in Graph Theory) to another path starting from p as a node (see Colucci *et al.*[3] for more details). A resource r is ***RDF-connected*** to a resource s if there exists an **RDF**-path from r to s. The *length* of this **RDF** path is n, and the ***RDF-distance*** between two resources is the length of the shortest **RDF**-path between them. Furthermore, the **RDF** distance between a resource r and a triple t is

the shortest **RDF**-distance between r and the subject of t—in particular, triples whose subject is r have zero-**RDF**-distance from r itself, as expected.

We propose to use the LCS of a cluster of **RDF** resources—obtained in any way—to explain their commonalities. The LCS could be useful both in a tuning phase of a clustering tool, and in an explanation of the result to an unacquainted final user. To this end, we attached to the construction of an LCS its explanation in English common language, as described in the next section.

4. Explanation of RDF r-graphs

The main contribution of this paper is the proposal of an approach grounded on LCS computation to the explanation of similarities among grouped **RDF** resources. The approach has been also implemented in a tool demonstrating its feasibility. The whole explanation process is performed in three steps: i) Problem settings; ii) (L)CS computation; iii) Natural Language Generation from the (L)CS computed at Step 2. We detail each step in the next subsections.

4.1. Problem Settings

The **first step** aims at specifying some preliminary settings, both for ensuring feasibility in the management of **RDF** resources and for tailoring the approach to the specific application scenario.

In particular, **RDF**-based applications need to select which triples qualify a resource r, among the ones available in the Web of Data. This filtering is necessary because the management of all triples linked to a resource would make unfeasible any application, given the huge and ever increasing dimensions of this information source.

Our approach adopts the explicit criteria proposed by Colucci *et al.* [3] for the selection of triples to include in the r-graph of a resource r. Such criteria ask for three settings: (1a) the datasets to analyse, (1b) the **RDF**-distance for exploration, and (1c) the list of so-called stop-patterns (triple patterns to be discarded).

The fourth additional setting (1d) is required to increase the significance of returned commonalities, by recursively eliminating from the LCS triples that provide little information, called *uninformative triples*. Thus, a—no more Least—Common Subsumer is obtained, containing only the most informative triples deducible from all r-graphs.

The set of stop-patterns and uninformative triples include both general patterns/triples (to be discarded in every application domain), and some domain-dependent patterns/triples, defined through the analysis of our results.

The last required settings (1e) are aimed at tailoring the explanation to the specific application domain to cope with. In particular, it is required a dictionary for the English language translation of URIs included in the reference dataset, to improve the readability of returned explanation. The full automation of this mapping (URI to common language term) is not always feasible, depending on the quality of used datasets. Notably, when a dataset allows for the retrieval of significant URI labels, the above mentioned dictionary may be automatically built. We summarize below all problem settings of Step 1:

1. a) *data sources*: which datasets (one or more) to explore for the comparison;
   b) ***RDF**-distance*: the maximum **RDF**-distance from r of triples to involve[1];
   c) *stop-patterns*: triples patterns to be excluded in the selection;
   d) LCS *uninformative triples*: triples which, although logically implied by the r-graphs of all analyzed resources, are recursively eliminated from the results;
   e) *dictionary of resources* involved in triples: English language form to be employed in the verbalization of URIs.

Such settings allow for flexibly tailoring the approach to different application scenarios, as demonstrated by its implementation for the comparison of drugs and contracting processes [5].

4.2. (L)CS Computation

The **second step** is implemented by Algorithm 1, that computes a CS of a $Cluster$ of **RDF** resources, given as first input. Since previous publications [3, 17] focused on the comparison of pairs, Algorithm 1 is an original contribution of this paper. Parameters n, D, ϕ are just passed as input to the call in Row 6 to the algorithm for computing the LCS of a pair of resources (one of which is the "running" LCS) [3]. The last input is the set of uninformative triples.

Algorithm 1: find a CS of a set of resources incrementally, by Associativity

1 $Find\_Cluster\_CS(Cluster, n, D, \phi, uninf\_triples)$;
Input :
- $Cluster$: an array containing the URIs of all resources to analyze;
- n: the **RDF-distance** for graphs exploration;
- D: the union of datasets to be explored;
- ϕ: a boolean predicate to be satisfied by selected triples;
- $uninf\_triples$: triples to be recursively eliminated from the results.

Output : r-graph $\langle cs, T_{cs}\rangle$ such that $\langle r_i, T_{r_i}\rangle \models \langle cs, T_{cs}\rangle$ for every $r_i \in Cluster$
Subroutine: $FindLCS(a, n_a, b, n_b, D, \phi)$ [3] returns the LCS (an r-graph) between a and b. Triples characterizing a and b are extracted by exploring D at **RDF**-distance n_a (resp. n_b) from a (resp. b). Only triples T satisfying $\phi(T) = true$ are extracted.

```
2  r_i ← Cluster[0];
3  seed ← r_i;
4  for j from 1 to |Cluster| do
5  |   r_j ← Cluster[j] ;
6  |   ⟨seed, T_seed⟩ ← FindLCS(seed, n, r_j, n, D, φ);
7  |   remove uninf_triples from T_seed;
8  |   if |T_seed| > 0 then
9  |   |   ⟨cs, T_cs⟩ ← ⟨seed, T_seed⟩;
10 |   else
11 |   |   break;
12 |   end
13 end
14 return ⟨cs, T_cs⟩;
```

Before entering the main loop (Rows 4–13), Algorithm 1 extracts the first resource from the list (Row 2). Then, the resource representing the root of the running CS, $seed$, is initialized to the first resource (Row 3). The CS $\langle cs, T_{cs}\rangle$ is incrementally built by looping over the remaining

[1]We recall that r-graph definition ensures connectedness: there must be an **RDF**-path from r to the subject of each chosen triple

resources (Rows 4–13). At each iteration stage, another resource r_j is extracted from the cluster (Row 5). In Row 6, we compute the LCS between this r_j and the running CS, $seed$, resulting from the previous iteration. At Row 7, uninformative triples are recursively removed from the CS computed so far. The loop continues until either $Cluster$ is empty or until the dimension of the last computed seed, T_{seed}, equals to zero (Row 11). In this second case, Algorithm 1 returns the last significant CS, whose T_{cs} is not empty and that is stored in $\langle cs, T_{cs} \rangle$ at each iteration (Row 9).

We notice that, in place of the exit criterion in Row 8, thresholds greater than 0 could be adopted to improve the significance of the returned set of commonalities, by giving up to the largest sharing of such commonalities among resources in the cluster and retrieving more significant CSs of specific cluster subsets. At implementation level, such more stringent requirements (that will be part of our future work) are possible because the running CSs, together with the subsets of resources it represents, is saved and available for further usage. In particular, at each iteration stage, the running CS is serialized[2] and saved in both *Turtle* and *N-triples* formats.

4.3. Natural Language Generation from triples in the (L)CS

The output of Algorithm 1 is passed to the **third step**, which implements the NLG approach for the explanation of (L)CS that we propose as our main contribution.

We recall that r-graphs modeling (L)CSs include blank nodes by construction (see Definition 1) and that blank nodes may occur in positions other than the root. For instance, while describing a cluster of Twitter accounts (see Sect. 5), we may find in the CS the following path (simplified here with prefix `ex:` for readability):

```
_:x   ex:mention  _:y .
_:y   ex:involved-in  ex:videogames .
```

which says that the accounts in the cluster (all represented by one blank node `_:x`) mention some resource (`_:y`) which is involved in the topic of videogames. We stress that the blank node `_:y` representing a generic resource does not occur by itself in the input r-graphs; it represents the fact that different accounts mention different IRIs, yet all such different IRIs are involved in videogames.

To the best of our knowledge, no available NLG tool[3] is able to verbalize **RDF** triples involving blank nodes in any position, and this makes our proposal (and tool) original.

In what follows, we describe our NLG approach, w.r.t. the six main building tasks synthesized by Gatt and Krahmer [18]:

1. **Content Determination** We determine the content to explain through the (L)CS construction detailed in Section 4.2, that discards from the (L)CS all information we consider uninformative; nevertheless the (L)CS r-graph may include paths which differ only in the involved blank nodes (see Figure 1) and would generate identical sentences in the explanation. Thus, we include only once such a content in the text under construction.

[2]Serialization uses the $RDFlib\ serialize$ method (https://rdflib.readthedocs.io/en/stable/plugin_serializers.html).
[3]https://aclweb.org/aclwiki/Downloadable_NLG_systems

2. **Text Structuring** Our (L)CS is an r-graph whose root is a blank node and that includes triple paths always **RDF**-connected to the root, involving triples with variable **RDF**-distance from the root (see the example at the beginning of this section). Such paths are considered equally informative and then presented in the order they appear in the (L)CS.

3. **Sentence Aggregation** We present each **RDF** path to the root in a single sentence.

4. **Lexicalisation** The lexicalization of triples depends on the **RDF**-distance form the root. We always generate a pronoun (depending on the **RDF**-distance) for the subject, a verb in past tense for the predicate and a noun for the object. When the object (respectively the predicate) is a blank node, we generate a phrase ("some generic resource"), that can be further explained through a relative sentence when the object (respectively the predicate) has successors in the r-graph (*i.e.*, when the path to verbalize has length greater than 1).

5. **Referring Expression Generation** The entities to describe in our text correspond to the **RDF** resources which stand as triple subject at any level of the **RDF** path to consider. We collect all such information around the root, generating sentences whose main subject is the phrase corresponding to the root ("They all").

6. **Linguistic Realisation** The final text is generated by following a human-crafted grammar-based approach. The main rules of the grammar are given below, with the conventions that terminal symbols are quoted, and that vertical bar represents a choice between two forms of a rule.

$$\begin{array}{rcl} CS & \rightarrow & \text{"They all" } Predicate\ (Noun \mid Noun\ RC) \\ RC & \rightarrow & \text{"which" } Predicate\ (Noun \mid Noun\ RC) \end{array}$$

   where RP is a nonterminal representing a relative clause. The nonterminals $Predicate$ and $Noun$ describe the (finitely many) predicates and nouns that linguistically realize terms in predicate- and subject/object-positions of a triple, respectively. Blank nodes are linguistically realized as "some generic resource" (see next section). As an aside, note that since $Predicate$ and $Noun$ produce a finite number of terminals, substituting all such terminals in the above rules yields a (very lengthy) right-recursive Type-3 Grammar. Jumping over some details about linguistic realisation, the reader can verify that one of the phrases such grammar can generate is indeed "They all mention some resource which is involved in videogames", that explains the example of the **RDF** path shown above.

   We use such a grammar as a guidance in the implementation of the linguistic realisation, with a breadth-first strategy: that is, taking all (say, n) **RDF**-paths contained in the CS, we follow the first triple of each one of them, generating n phrases that are complete if the path has length 0 (just one triple), or they contain a nonterminal RC when the path has length ≥ 1. Then we follow each path one triple further, and add their verbalization substituting all non terminals RC once. We proceed, until no path extends further. This breadth-first generation of phrases make it easier to perform subsequent verbal improvements, *e.g.*, subject sharing, sentence coordination, etc.

In the rest of this section, we delve into some implementation details. The uninterested reader can jump to the next section to see the results we obtained.

In brief, Algorithm 2 takes an r-graph in input and describes its root by generating a verbose human-readable explanation of its triples set.

If the input r-graph is the CS of a group of different resources (like the one returned by Algorithm 1), the root is a blank node and the generated text summarizes common features among analyzed resources.

Algorithm 2: The algorithm returns a dictionary storing for each explored **RDF**-distance (dictionary key), the text generated for explanation (dictionary value)

1 $get\_description\ (\langle g, T_g \rangle, d, URIs\_Dictionary)$;
Input :
- $\langle g, T_g \rangle$: the r-graph to explain;
- d: the **RDF-distance** for r-graph exploration;
- $URIs\_Dictionary$: a dictionary with keys indexing all URIs defined in the name-space and values storing the English terms corresponding to them;

Output :
- $output$: a dictionary storing for each explored **RDF**-distance n (dictionary key), the corresponding generated text (dictionary value);

Subroutine: $get\_node\_translation(\langle node, T_{node} \rangle, URIs\_Dictionary)$ (Algorithm 3) returns, for each $node$: the text generated for linguistically realizing the triples rooted in it; the list of node neighbours yet to be explored; a dictionary storing indexes and descriptive triples of nodes to be explained through relative sentences.

2 $output \leftarrow$ empty dictionary;
3 $to\_explore \leftarrow [g]$;
4 $explored \leftarrow \emptyset$;
5 **for** $iteration$ *from* 0 *to* d **do**
6 $\quad iteration\_output \leftarrow$ empty dictionary ;
7 $\quad$ **for** $(node \in to\_explore)$ *and* $(node \notin explored)$ **do**
8 $\quad\quad T_{node} \leftarrow$ all triples in T_g with $node$ as subject;
9 $\quad\quad$ add $node$ to $explored$;
10 $\quad\quad$ **if** $T_{node} \neq \emptyset$ **then**
11 $\quad\quad\quad node\_output, node\_neighbours, relative\_st =$ $get\_node\_realization(\langle node, T_{node} \rangle, URIs\_Dictionary)$;
12 $\quad\quad$ **end**
13 $\quad\quad$ add $node\_output$ to $iteration\_output$;
14 $\quad\quad to\_explore \leftarrow$ resources in $node\_neighbours$ and not in $explored$;
15 $\quad$ **end**
16 $\quad$ add $iteration\_output$ to $output$;
17 **end**
18 **for** *each* $resource \in output[n]$ *indexed with* k *in* $relative\_st$ **do**
19 $\quad$ add to $output[n]$ one relative sentence for each sentence in $relative\_st[k]$;
20 **end**
21 return $output$;

Algorithm 2 starts the exploration of $\langle g, T_g \rangle$ from its root node g and returns a dictionary, $output$, storing, for each explored **RDF**-distance, the text generated for its explanation; $output$ is initialized to an empty dictionary at Row 2.

The main loop of Algorithm 2 (Rows 5–17) is responsible for iteration over different **RDF**-distance values (up to the input d): for each **RDF**-distance, it computes an $iteration\_output$ which is initialized as empty at every iteration (Row 6) and added to the final $output$ in Row 16.

The loop in Rows 7–15 analyzes all resources present in $to\_explore$ (but not in $explored$) and, for each $node$ of them, finds all the triples, T_{node}, in T_g rooted in $node$ (Row 8). Then, $node$ is put in the array of already explored resources (Row 9). For each $node$ such that T_{node} is not empty, the subroutine $get\_node\_realisation$ (see Algorithm 3) is called in Row 11 to perform the linguistic realization of triples rooted in $node$. In particular, Algorithm 3 returns, for each node:

Algorithm 3: The algorithm returns, for a single node, the output text to generate, the list of nodes yet to be explored and a data structure supporting the writing of relative sentences at every **RDF**-distance.

1 $get\_node\_realization(\langle node, T_{node}\rangle, URIs\_Dictionary)$;
Input :
- $\langle node, T_{node}\rangle$: the r-graph of the node to explain;
- $URIs\_Dictionary$: the dictionary of URIs defined in the dataset;

Output :
- $node\_output$: a dictionary storing, for each node identifier, the linguistic realization of the triples rooted in the node;
- $node\_neighbours$: a list of nodes to be explored in next stages of iteration by Algorithm 2 ;
- $relative\_st$: a dictionary storing indexes and descriptive triples of nodes to be explained through relative sentences.

2 $node\_neighbours \leftarrow \emptyset$;
3 $node\_output \leftarrow$ empty dictionary ;
4 **for** *each triple* $\ll node\ predicate\ object \gg \in T_{node}$ **do**
5 **if** $predicate \in URIs\_Dictionary$ **then**
6 $pred\_realization \leftarrow URIs\_Dictionary[predicate]$
7 **else**
8 **if** $predicate$ *is a blank node* **then**
9 $pred\_realization \leftarrow$ "some generic resource"
10 **else**
11 $pred\_realization \leftarrow$ "not identified dataset resource"
12 **end**
13 add an $index$ for accessing $predicate$ in $relative\_st$;
14 **end**
15 add $predicate$ to $node\_neighbours$;
16 **if** $object \in URIs\_Dictionary$ **then**
17 $object\_realization \leftarrow URIs\_Dictionary[object]$
18 **else**
19 **if** $object$ *is a blank node* **then**
20 $object\_realization \leftarrow$ "some generic resource"
21 **else**
22 $object\_realization \leftarrow$ "not identified dataset resource";
23 **end**
24 add an $index$ for accessing $object$ in $relative\_st$;
25 **end**
26 add $object$ to $node\_neighbours$;
27 $realization_{node} = generate\_text(pred\_realization, object\_realization)$;
28 **if** $node$ *is indexed in* $relative\_st$ **then**
29 add $realization_{node}$ to $relative\_st$
30 **else**
31 add $realization_{node}$ to $node\_output$;
32 **end**
33 **end**
34 return $node\_output, node\_neighbours, relative\_st$;

the generated text ($node\_output$), the list of nodes yet to be explored ($node\_neighbours$) and a data structure supporting the writing of relative sentences at every RDF-distance ($relative\_st$).

Algorithm 2 uses the variable $node\_output$ to update the running $iteration\_output$ (Row 13) and $node\_neighbours$ to set the array $to\_explore$ (Row 14).

After the execution of the main loop (Rows 5–17), $output$ stores, for each analyzed **RDF**-distance n, the related text generated from $\langle g, T_g\rangle$. The text to generate, at any distance n, may refer to nodes that are indexed in $relative\_st$ with key k, and whose linguistic realization is

stored in $relative\_st[k]$. In this case, one sentence relative to k is added for each text item in $relative\_st[k]$ (Rows 18–20).

Algorithm 3 starts by initializing $node\_neighbours$ to an empty set and $node\_output$ to an empty dictionary (Rows 2–3). For each triple in T_{node}, both the predicate and the object are explored to determine a verbal form for them ($pred\_realization$ and $obj\_realization$). In particular, Algorithm 3 discriminates among 3 different kinds of resources:

1. The resource is in the input $URIs\_Dictionary$, that includes its linguistic realization (see Row 5 and 17);
2. The resource is a blank node, and is explained by the phrase "some generic resource" (see Row 9 and 20);
3. The resource is neither in the dictionary of defined URIs (Case 1) nor a blank node (Case 2): it is a generic (but not anonymous) resource, which has been given a URI, but which is not further defined in the reference namespace. Thus, a generic phrase—"not identified dataset resource"—is used for its explanation (see Row 11 and 22).

In Cases 2 and 3, Algorithm 3 updates the $relative\_st$ dictionary (Rows 13 and 24) supporting the building of relative sentences in the final output of Algorithm 2. In all cases, the triple predicate and object are added to $node\_neighbours$ (Row 15 and 26), to be explored in next iterations by Algorithm 2. Also (Row 27), the grammar above (see Linguistic Realization description) is used to generate a text on the basis of predicate and object realizations, through the subroutine $generate\_text$. Such a generated text is assigned to the variable $realization_{node}$.

Algorithm 3 proceeds (conditional block in Rows 28–32) by adding the triple $realization_{node}$ either to the dictionary $relative\_st$ for nodes indexed in it (and thus to be explained through relative sentences) or directly to $node\_output$ for other nodes.

5. Use case: Explaining Twitter Accounts Similarity

The exponential growth in popularity of online social networking systems led to an increasing importance of social media analytic tools for a variety of applications, such as customer segmentation, market analysis and recommendation systems based on collaborative filtering.

By exploiting the information available in Linked Data, our method is able to highlight the semantic properties shared by two or more users on a given platform, producing an easy-to-read, understandable description. In particular, we use two combined datasets as information source: Influence Tracker[4] and DBPedia[5].

The Influence Tracker (IT) ontology [19] collects activities and interactions over the so-called "Twittersphere" of several users modelled as **RDF**-graphs and grouped together according to the content they publish. In our examples, we refer to the prefixes in Turtle notation shown below:

```
@prefix it: <http://www.influencetracker.com/resource> .
@prefix it0: <http://www.influencetracker.com/ontology#> .
@prefix dct: <http://purl.org/dc/terms/>
@prefix dbp: <http://dbpedia.org/property/>
@prefix Category: <http://dbpedia.org/resource/Category>
```

[4]https://old.datahub.io/dataset/influence-tracker-dataset
[5]https://www.dbpedia.org/

The class `it:User` models owners of Twitter accounts, either a physical person or a company, in terms of mentions, replies, hashtags, photos, URLs, and so on. The activity of Twitter user accounts is traced through several predicates, useful for catching semantic similarity among different users. In our examples, we refer to the predicate `it0:hasMentioned`, that connects a resource (belonging to class `it:User`) to users mentioned in its tweets. Notably, the IT service provides an RDF graph, called *twitterSimilarity*, which traces the connection (through the property `it0:hasSimilarity`) of Twitter accounts considered similar based on their published content. We use the knowledge modeled in *twitterSimilarity* to extract the groups of similar accounts to be analyzed in our use case.

In order to enrich the informative content of our explanation, we combined the information in IT with selected knowledge from DBPedia. Such an integration is possible thanks to the high quality of IT dataset, which includes links to URIs in DBPedia, and is then classifiable as 5-stars Linked Open Data [20]. The IT service provides, in fact, the *dbpediaGraph*, that maps—through the property `it0:dbpediaUri`—the correspondence between Twitter users in the IT namespace and the related resource in DBpedia, if any.

Following such correspondences, we extract from DBpedia further information about account owners. In our example, we only extract triples including the predicates `dct:subject` and `dbp:industry`[6].

In the following, we analyze the commonalities of three accounts grouped by similarity in IT: *it:instagram*, *it:dannysullivan* and *it:jimmykimmel*. We first compute a significant CS of the three resources, by flexibly providing an **RDF**-distance, a set of stop-patterns and a set of uninformative triples. Then, we generate a text linguistically realizing such a CS.

In Figure 1, we focus only on the portion of CS computed at **RDF**-distance 1 from the root and show the related graph representation. The reader may notice that two different paths exist from the LCS root to the two resources "social media" and "videogames streaming services", passing through different blank nodes. In the generation of the verbal explanation to return, we notify only one of these paths[7], that would be indistinguishable for users unacquainted in **RDF**.

Figure 2 reports the verbal explanation generated from the LCS at **RDF**-distance 1 of the three Twitter accounts cited above[8].

Again, the explanation coming from the analysis at **RDF**-distance 1 (Row 96 in Fig. 2) benefits from the verbalization of paths passing through blank nodes, and collecting important commonalities.

6. Conclusion

Based on a theory on (Least) Common Subsumers in **RDF**, we built a tool that verbalizes in the English common language the commonalities between two or more resources in **RDF**. We piped our tool to a standard clustering tool applied to the domain of Twitter accounts, in order

[6]The portion of the DBPedia graph to include in the analysis may be easily adapted to the explanation need.
[7]It corresponds to a "lean" version [2] of the LCS.
[8]Recall that triples whose subject is r have **RDF**-distance 0 from r itself; hence, the mentioned triples at **RDF**-distance 1 from r have a subject which is either an object or a predicate of a triple at **RDF**-distance 0.

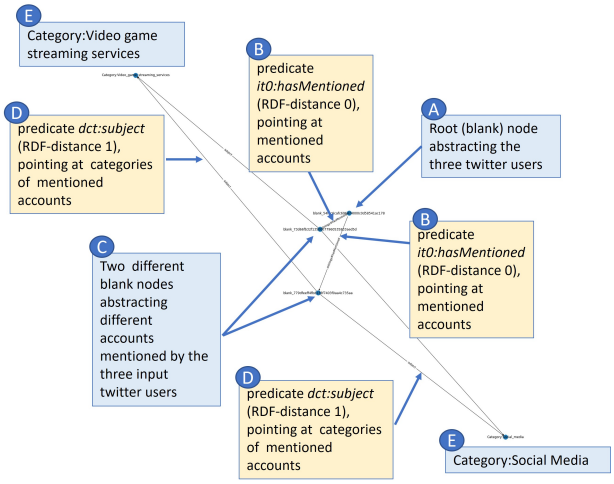

Figure 1: A CS graph obtained between the three twitter users: *it:instagram*, *it:dannysullivan* and *it:jimmykimmel* (whose r-graphs include only triples at **RDF**-distance 1 from the resources). The upper-right-most blank node (Callout A) represents the three resources; the two blank nodes to its left (Callout C) represent resources (different from each other, verbalized as "some generic resource" in Figure 2) which link (through predicates in Callout B and D) *it:instagram*, *it:dannysullivan* and *it:jimmykimmel* to "social media" on one side, and "videogames streaming services" on the other side (Callout E).

to explain the results to an end user. Blank nodes are fundamental in our Common Subsumers to represent partial commonalities in the **RDF**-paths starting from the clustered resources; we verbalized them using relative pronouns in relative sentences. To our knowledge, there is no tool that verbalizes also blank nodes in **RDF**-graphs.

According to the classification of Samek *et al.* [21], our tool fulfills at least three objectives of Explanations in AI: 1) verification of the system: it allows users to check for meaningful/clueless clusterizations; 2) improvement of the system: looking at the verbalization, a user can tune the clusterization tool in order to weigh some features more than others; 3) learning from the system: the verbalization makes explicit commonalities that might not be immediately visible by looking at the resources in the cluster.

Future work may include extension both to the LCS construction and to the natural language generation approach. In particular, LCS computation may be extended to consider also paths traversing triples backwards (from the object to the subject). The extension of **RDF**-LCS to backward **RDF**-paths is straightforward; yet their verbalization may need a chain of passive relative English sentences, which are less readable than the forward ones. Another future direction is verbalizing a subset of the cluster [22], since when the explanation for the commonalities of a set of resources is too shallow, focusing on the commonalities of a subset of the resources may yield more interesting explanations.

Regarding NLG, the analysis of results suggests some improvements to our approach, at least in the sentence aggregation and in the text structuring tasks. In particular, by looking at Figure 2, the first 95 sentences in the text generated from the CS appear like a list of mentioned accounts and included hashtags, that may be not interesting for all readers. In most cases, a text like "They all mentioned the same 59 Twitter users. They all included the same 36 hashtags." would

```
The resources in analysis present the following properties in common:

1) They all mentioned  the Twitter user: veephbo
2) They all mentioned  the Twitter user: gma
3) They all mentioned  the Twitter user: mindykaling
4) They all mentioned  the Twitter user: huffingtonpost
5) They all included the hashtag #vpdebate
6) They all mentioned  the Twitter user: katyperry
7) They all included the hashtag #lgbtq
8) They all mentioned  the Twitter user: taylorswift13
9) They all included the hashtag #thanksgiving
10) They all included the hashtag #theforceawakens
11) They all mentioned  the Twitter user: kingjames
12) They all included the hashtag #harrypotter
13) They all included the hashtag #sxsw
14) They all included the hashtag #sb51
15) They all included the hashtag #debate
16) They all mentioned  the Twitter user: hbo
17) They all mentioned  the Twitter user: time
18) They all included the hashtag #veteransday
19) They all mentioned  the Twitter user: stranger_things
20) They all mentioned  the Twitter user: glaad
21) They all mentioned  the Twitter user: cnnpolitics
22) They all mentioned  the Twitter user: ew
23) They all included the hashtag #emmys
24) They all mentioned  the Twitter user: ringer
25) They all mentioned  the Twitter user: kanyewest
26) They all mentioned  the Twitter user: chrissyteigen
27) They all mentioned  the Twitter user: kimkardashian
28) They all mentioned  the Twitter user: variety
29) They all mentioned  the Twitter user: starbucks
30) They all included the hashtag #rogueone
31) They all included the hashtag #veep
32) They all mentioned  the Twitter user: whitehouse
33) They all included the hashtag #olympics
34) They all mentioned  the Twitter user: oliviawilde
35) They all mentioned  the Twitter user: latimes
36) They all included the hashtag #goldenglobes
37) They all mentioned  the Twitter user: applemusic
38) They all included the hashtag #nationalselfieday
39) They all mentioned  the Twitter user: deadline
40) They all included the hashtag #gopdebate
41) They all included the hashtag #f8
42) They all included the hashtag #backtoschool
43) They all mentioned  the Twitter user: sesamestreet
44) They all mentioned  the Twitter user: bretbaier
45) They all mentioned  the Twitter user: facebook
46) They all mentioned  the Twitter user: sethmeyers
47) They all mentioned  the Twitter user: nerdist
48) They all included the hashtag #starwars
49) They all included the hashtag #gameofthrones
50) They all mentioned  the Twitter user: michelleobama
51) They all mentioned  the Twitter user: lesdoggg
52) They all mentioned  the Twitter user: netflix
53) They all mentioned  the Twitter user: tsa
54) They all mentioned  the Twitter user: thewrap
55) They all mentioned  the Twitter user: therock
56) They all included the hashtag #superbowl
57) They all included the hashtag #nbafinals
58) They all mentioned  the Twitter user: newyorker
59) They all included the hashtag #demdebate
60) They all mentioned  the Twitter user: abc7
61) They all included the hashtag #rncincle
62) They all mentioned  the Twitter user: jimmyfallon
63) They all mentioned  the Twitter user: nytimes
64) They all included the hashtag #westworld
65) They all mentioned  the Twitter user: beyonce
66) They all mentioned  the Twitter user: kobebryant
67) They all included the hashtag #aprilfools
68) They all included the hashtag #tbt
69) They all mentioned  the Twitter user: ladygaga
70) They all mentioned  the Twitter user: markruffalo
71) They all mentioned  the Twitter user: justintrudeau
72) They all mentioned  the Twitter user: traceeellisross
73) They all mentioned  the Twitter user: thr
74) They all mentioned  the Twitter user: foxnews
75) They all mentioned  the Twitter user: nasa
76) They all included the hashtag #sb50
77) They all mentioned  the Twitter user: pontifex
78) They all included the hashtag #vmas
79) They all included the hashtag #electionday
80) They all mentioned  the Twitter user: cnn
81) They all mentioned  the Twitter user: billboard
82) They all mentioned  the Twitter user: imkristenbell
83) They all included the hashtag #grammys
84) They all mentioned  the Twitter user: rickygervais
85) They all mentioned  the Twitter user: shondarhimes
86) They all mentioned  the Twitter user: johncena
87) They all mentioned  the Twitter user: usatoday
88) They all included the hashtag #strangerthings
89) They all included the hashtag #memorialday
90) They all included the hashtag #worldseries
91) They all included the hashtag #got
92) They all mentioned  the Twitter user: colbertlateshow
93) They all included the hashtag #worldcup
94) They all mentioned  the Twitter user: forbes
95) They all mentioned  the Twitter user: johnlegend


96) They all share the property "mentioned"
   each one of them referencing some generic resource
   which  is involved in the field of social media
   and  is involved in the field of videogames streaming services
```

Figure 2: Verbal explanation of a CS (at **RDF**-distance 1) of the three twitter users: *it:instagram*, *it:dannysullivan* and *it:jimmykimmel*. Phrases 1–95 explains short **RDF**-paths of length 0; Phrase 96 verbalizes an **RDF**-path of length 1.

be enough informative. Thus, as first improvement, we are investigating a different *sentence aggregation* strategy, aimed at returning a more compact explanation when the generated text is too verbose and highly repetitive. This strategy needs to be information-conservative and able to return on demand the full text generated from the CS.

Also, we are designing a different approach to *text structuring*; in this work we structured the text by presenting the sentences generated from the paths in the (L)CS in the order they appear in the r-graph. This causes a random alternation of predicates involved in sentences, as shown in the example in Figure 2 (w.r.t. to predicates "mentioned" and "included the hashtag"). Such a result suggests to present close to each other all sentences related to the same predicate. The

new design would significantly benefit also from the introduction of a weighting mechanism for the importance of predicates, to be used for setting the order of presentation of generated sentences.

Acknowledgments

Projects Regione Lazio-DTC/“SanLo” (CUP F85F21001090003) and “TEBAKA-SISTEMA PER ACQUISIRE CONOSCENZE DI BASE DEL TERRITORIO" (Code ARS01_00815, CUP B82C20000160005) partially supported this work.

References

[1] S. Eddamiri, A. Benghabrit, E. M. Zemmouri, RDF graph mining for cluster-based theme identification, Int. J. Web Inf. Syst. 16 (2020) 223–247. doi:10.1108/IJWIS-10-2019-0048.

[2] P. Hayes, P. F. Patel-Schneider, RDF 1.1 semantics, W3C recommendation, 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[3] S. Colucci, F. Donini, S. Giannini, E. Di Sciascio, Defining and computing least common subsumers in RDF, Web Semantics: Science, Services and Agents on the World Wide Web 39 (2016) 62 – 80. URL: http://dx.doi.org/10.1016/j.websem.2016.02.001.

[4] F. Baader, B. Sertkaya, A.-Y. Turhan, Computing the least common subsumer wrt a background terminology, Journal of Applied Logic 5 (2007) 392–420.

[5] S. Colucci, F. M. Donini, N. Iurilli, E. Di Sciascio, A business intelligence tool for explaining similarity, in: Proc. Of Model-driven Organizational and Business Agility (MOBA), Springer, 2022.

[6] R. S. Michalski, Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts, Int. Journal of Policy Analysis and Information Systems 4 (1980) 219—-244.

[7] A. Pérez-Suárez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, A review of conceptual clustering algorithms, Art. Intell. Review 52 (2019) 1267–1296.

[8] 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: Proc. of the 21st European Conf. on Artif. Intell. (ECAI 14), IOS Press, 2014.

[9] I. Tiddi, M. d’Aquin, E. Motta, Dedalo: Looking for clusters explanations in a labyrinth of linked data, in: V. Presutti, C. d’Amato, F. Gandon, M. d’Aquin, S. Staab, A. Tordai (Eds.), The Semantic Web: Trends and Challenges, Springer International Publishing, Cham, 2014, pp. 333–348.

[10] N. Bouayad-Agha, G. Casamayor, L. Wanner, Natural language generation in the context of the semantic web, Semantic Web 5 (2014) 493–513.

[11] E. Colin, C. Gardent, Y. M’rabet, S. Narayan, L. Perez-Beltrachini, The webNLG challenge: Generating text from DBpedia data, in: Proceedings of the 9th International Natural Language Generation conference, 2016, pp. 163–167.

[12] G. Zhou, G. Lampouras, WebNLG challenge 2020: Language agnostic delexicalisation for

multilingual RDF-to-text generation, in: Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), Association for Computational Linguistics, Dublin, Ireland (Virtual), 2020, pp. 186–191. URL: https://aclanthology.org/2020.webnlg-1.22.

[13] P. Vougiouklis, H. Elsahar, L.-A. Kaffee, C. Gravier, F. Laforest, J. Hare, E. Simperl, Neural wikipedian: Generating textual summaries from knowledge base triples, Journal of Web Semantics 52-53 (2018) 1–15. doi:10.1016/j.websem.2018.07.002.

[14] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Proc. of the 27th Int. Conf. on Neural Information Processing Systems - Volume 2, NIPS'14, MIT Press, Cambridge, MA, USA, 2014, p. 3104–3112.

[15] J. Li, G. Cheng, Q. Liu, W. Zhang, E. Kharlamov, K. Gunaratna, H. Chen, Neural entity summarization with joint encoding and weak supervision, in: C. Bessiere (Ed.), Proceedings of IJCAI-2020, ijcai.org, 2020, pp. 1644–1650. URL: https://doi.org/10.24963/ijcai.2020/228.

[16] S. Colucci, F. M. Donini, E. Di Sciascio, Common subsumers in RDF, in: Proc. of AI*IA-13, volume 8249 of *LNAI*, Springer, 2013.

[17] S. Colucci, F. M. Donini, E. Di Sciascio, Logical comparison over RDF resources in bio-informatics, Journal of Biomedical Informatics 76 (2017) 87–101. Https://authors.elsevier.com/a/1W4zU5SMDQUvEB.

[18] A. Gatt, E. Krahmer, Survey of the state of the art in natural language generation: Core tasks, applications and evaluation, J. Artif. Int. Res. 61 (2018) 65–170.

[19] G. Razis, I. Anagnostopoulos, Semantifying twitter: The influence tracker ontology, in: 2014 9th International Workshop on Semantic and Social Media Adaptation and Personalization, 2014, pp. 98–103. doi:10.1109/SMAP.2014.23.

[20] T. Berners-Lee, Linked Data - Design Issues, http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[21] W. Samek, T. Wiegand, K. Müller, Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models, CoRR abs/1708.08296 (2017). URL: http://arxiv.org/abs/1708.08296. arXiv:1708.08296.

[22] S. Colucci, E. Tinelli, E. Di Sciascio, F. Donini, Automating competence management through non-standard reasoning, Engineering Applications of Artificial Intelligence 24 (2011) 1368–1384. doi:10.1016/j.engappai.2011.05.015.