

# Entity Comparison in RDF Graphs

Alina Petrova, Evgeny Sherkhonov, Bernardo Cuenca Grau, and Ian Horrocks

University of Oxford

**Abstract.** In many applications, there is an increasing need for the new types of RDF data analysis that are not covered by standard reasoning tasks such as SPARQL query answering. One such important analysis task is entity comparison, i.e., determining what are similarities and differences between two given entities in an RDF graph. For instance, in an RDF graph about drugs, we may want to compare Metamizole and Ibuprofen and automatically find out that they are similar in that they are both analgesics but, in contrast to Metamizole, Ibuprofen also has a considerable anti-inflammatory effect. Entity comparison is a widely used functionality available in many information systems, such as universities or product comparison websites. However, comparison is typically domain-specific and depends on a fixed set of aspects to compare. In this paper, we propose a formal framework for domain-independent entity comparison over RDF graphs. We model similarities and differences between entities as SPARQL queries satisfying certain additional properties, and propose algorithms for computing them.

## 1 Introduction

The Resource Description Framework (RDF) is the standard format for representing and integrating information on the Web. The canonical reasoning task over RDF data exploited in applications is query answering, where SPARQL is the standard query language developed for that purpose [10]. There is, however, an increasing need in many applications for non-standard analysis tasks that do not directly correspond to SPARQL query answering. One such important task is *entity comparison*—that is, to determine what are the similarities and differences between the information about two given entities in an RDF graph.

Let us consider two example use cases. In the first one, a startup company is developing a toolkit for analysing widely-used biomedical RDF repositories, such as Bio2RDF [5]. The tool being developed should provide a drug comparison functionality; in particular, when given two drugs described in an RDF graph from the repository, such as Ibuprofen and Metamizole, the tool should be able to automatically report that “both drugs are analgesics and can reduce fever; however, Metamizole can also act as a spasm reliever, whereas Ibuprofen has an anti-inflammatory function”. The second use case concerns the development of an analysis tool on top of IMDB data; such tool should allow users to compare arbitrary aspects of movie-making, such as directors, producers, actors and so on. For example, when comparing Quentin Tarantino to Martin Scorsese, the tool

should report that they are similar in that they are both male directors who won both an Oscar and a Golden Globe and who have also acted in their own movies; in turn, they are different in that Tarantino won the Palme d'Or at the Cannes Film Festival, while Scorsese won an Emmy award, to which Tarantino was only nominated.

Entity comparison is conventionally seen in the Information Retrieval community as a type of exploratory search [15, 22]. It is an important task which is implemented in a wide range of tools and web portals, in domains as diverse as hotels,<sup>1</sup> cars,<sup>2</sup> universities,<sup>3</sup> or online shopping<sup>4</sup>. Existing entity comparison tools typically perform a side-by-side comparison of items based on a fixed (often hard-coded) template of features to compare (e.g., price, location, rating, and so on in the case of hotels). Relying on a fixed set of features is a reasonable solution for tabular, domain specific data whose structure is relatively rigid and stable. It is even appropriate in the context of graph data, provided that a limited set of relevant features can be specified beforehand; for instance, Facebook Friendship pages allow for the comparison of two Facebook users by displaying their shared information based on a limited set of features specific to social networks (e.g., “likes”, mutual friends, relationship status).

A more flexible approach to entity comparison is, however, needed in the context of Linked Data, where loosely structured RDF graphs (often describing overlapping domains) are frequently merged and updated. Up to now, such approaches have mainly been based on the structure of the graph, e.g., finding a path that connects the two entities (see Section 7 for a discussion of related work). In this paper we propose a novel approach based on the semantics of the graph.

In Section 3 we propose a logical framework for our approach, where similarities and differences between entities are formalised as conjunctive SPARQL queries. Specifically, a *similarity query* (resp. a *difference query*) for given entities in an RDF graph is a query having both entities as answers (resp. having one entity as answer but not the other). In the case of similarity queries, we are interested in the *most specific ones*, e.g., knowing that Tarantino and Scorsese are both American-born film directors is more informative than reporting only that they are both film directors. In turn, in the case of difference queries we are interested in the *most general ones*, e.g., knowing that Brad Pitt is an actor, whereas George Lucas is a producer is more informative than knowing that the former is an American actor while the latter is an American producer, since being American is irrelevant to differentiating them.

In Section 4 we focus on similarities, and show that most specific similarity queries for given entities in an RDF graph are unique modulo equivalence. We then propose a polynomial-time algorithm for computing one such most specific similarity query. The problem we consider in this section is strongly related to the

<sup>1</sup> <http://www.flightnetwork.com/pages/hotel-comparison-tool/>

<sup>2</sup> <http://www.cars.com/go/compare/modelCompare.jsp>

<sup>3</sup> <http://colleges.startclass.com/>

<sup>4</sup> <http://www.intel.co.uk/content/www/uk/en/products/compare-products.html>

*Query Reverse Engineering problem* in RDF [2], as well as to that of computing *Least Common Subsumers* in Description Logic ontologies [3].

In Section 5, we focus on difference queries. We first argue that this is a hard problem; specifically, we argue that simply checking existence of a difference query for two given entities in a graph is  $\text{coNP}$ -complete. We then propose an exponential-time algorithm for computing a most general difference query, should one exist.

Finally, we describe a prototype implementation of the algorithm for computing a most specific similarity query and present a proof of concept case study using the data from Wikipedia infoboxes.

## 2 Preliminaries

We follow [16] in the definition of RDF graphs and triple patterns. Let  $\mathbf{U}$ ,  $\mathbf{L}$  and  $\mathbf{B}$  be pairwise disjoint, countably infinite sets of URIs, literals and blank nodes, respectively. An *RDF triple* (or simply a *triple*) is a tuple  $(s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L} \cup \mathbf{B})$ . In such a triple,  $s$  is the *subject*,  $p$  the *predicate* and  $o$  the *object*. An *RDF graph*  $G$  is a finite set of triples. The *domain* of  $G$ , denoted  $\text{dom}(G)$ , is the set of all URIs, literals and blank nodes that occur in  $G$ . Any URI or literal from  $\text{dom}(G)$  is called an *entity*.

Let  $\mathbf{V}$  be a countably infinite set of variables disjoint from  $\mathbf{U}$  and  $\mathbf{L}$ . A *term* is an element from  $\mathbf{U} \cup \mathbf{L} \cup \mathbf{V}$ . The basic building block of our queries is a *triple pattern*, which is an element from  $(\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{L} \cup \mathbf{V})$ . A *basic graph pattern* is a non-empty finite set  $P$  of triple patterns. For any basic graph pattern  $P$ , we denote with  $\text{term}(P)$  and  $\text{var}(P)$  the sets of terms and variables occurring in  $P$ , respectively.

We define a *query*  $Q$  as a pair  $(\bar{X}, P)$ , where  $P$  is a basic graph pattern and  $\bar{X} \subseteq \text{var}(P)$  is the set  $\text{avar}(Q)$  of *answer variables* of  $Q$ . Such queries capture the fragment of SPARQL queries of the form `SELECT ? $\bar{X}$  WHERE  $P$` , with  $P$  a basic graph pattern. We define  $\text{term}(Q) = \text{term}(P)$  and  $\text{var}(Q) = \text{var}(P)$ . We say that  $Q$  is *monadic* if  $\text{avar}(Q)$  is a singleton. Basic graph pattern  $P$  is *connected* if for every  $t, t' \in \text{term}(P)$  there is a sequence of triple patterns  $tp_1, \dots, tp_n$  in  $P$  such that  $t \in \text{term}(tp_1)$ ,  $t' \in \text{term}(tp_n)$  and  $\text{term}(tp_i) \cap \text{term}(tp_{i+1}) \neq \emptyset$ , for  $1 \leq i < n$ . Query  $Q = (\bar{X}, P)$  is *connected* if so is  $P$ . For brevity, in examples we will write a query  $Q = (\bar{X}, P)$  simply as  $P$  and adopt the convention that  $X_{(i)}$  represent answer variables, whereas  $Y_{(j)}$  represent the remaining variables.

We next recapitulate the semantics of queries. A *valuation* over variables  $\bar{X}$  is a mapping  $\nu$  from  $\bar{X}$  to  $\mathbf{U} \cup \mathbf{L} \cup \mathbf{B}$ . For  $\nu$  a valuation over  $\bar{X}$  and  $\bar{Y} \subseteq \bar{X}$ , let  $\nu|_{\bar{Y}}$  be the restriction of  $\nu$  to  $\bar{Y}$ . Valuations are applied to triple patterns and basic graph patterns in the obvious way. Let  $Q = (\bar{X}, P)$  be a query, let  $G$  be an RDF graph, and  $\nu$  a valuation over  $\bar{X}$ . Then,  $G$  *satisfies*  $Q$  under  $\nu$ , denoted  $G, \nu \models Q$  if  $\nu = \mu|_{\bar{X}}$  for some valuation  $\mu$  over  $\text{var}(Q)$  satisfying  $\mu(P) \subseteq G$ . The semantics  $[Q]_G$  of a query  $Q = (\bar{X}, P)$  over  $G$  is

$$[Q]_G = \{\nu(\bar{X}) \mid G, \nu \models Q \text{ and } \nu \text{ is a valuation over } \bar{X}\}.$$

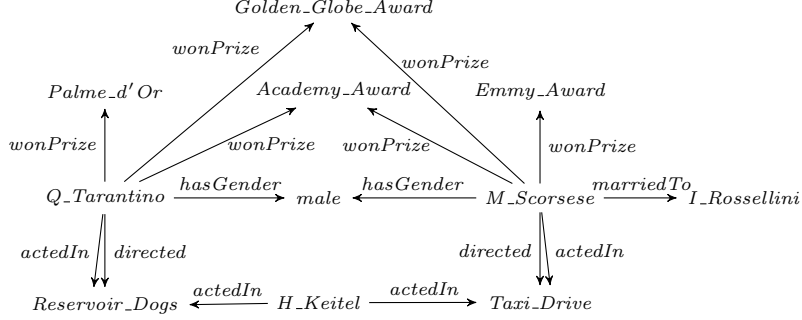


Fig. 1. An example RDF graph  $G_{mov}$ .

Let  $G$  be an RDF graph. The *canonical graph pattern* of  $G$  is the set  $\text{Can}(G)$  of triple patterns  $(X_s, X_p, X_o)$  for each triple  $(s, p, o)$  in  $G$ , where  $X_s$ ,  $X_p$  and  $X_o$  are variables uniquely assigned to  $s$ ,  $p$  and  $o$  in  $G$ . A *canonical query* of  $G$  is any query of the form  $(\bar{X}, \text{Can}(G))$ .

Let  $Q_1 = (\bar{X}_1, P_1)$  and  $Q_2 = (\bar{X}_2, P_2)$  be queries. We say that  $Q_1$  is *subsumed* by  $Q_2$ , denoted as  $Q_1 \subseteq Q_2$ , if  $[Q_1]_G \subseteq [Q_2]_G$  for every RDF graph  $G$ . The subsumption relation between two queries with equal number of answer variables can be characterised by existence of a homomorphism—a mapping  $h : \text{term}(Q_2) \rightarrow \text{term}(Q_1)$  that is the identity on URIs, literals and answer variables and satisfying  $(h(t_s), h(t_p), h(t_o)) \in P_1$  whenever  $(t_s, t_p, t_o) \in P_2$ . It is well-known that  $Q_1 \subseteq Q_2$  if and only if there exists a homomorphism from  $Q_2$  to  $Q_1$ . Subsumption allows us to compare queries relative to their specificity. We say that  $Q_1$  is *more specific* than  $Q_2$  if  $Q_1 \subseteq Q_2$ ; it is *strictly more specific*, denoted as  $Q_1 \subset Q_2$ , if  $Q_1 \subseteq Q_2$  and  $Q_2 \not\subseteq Q_1$ . Finally,  $Q_1$  and  $Q_2$  are *equivalent*, denoted  $Q_1 \equiv Q_2$ , if  $Q_1 \subseteq Q_2$  and  $Q_2 \subseteq Q_1$ .

### 3 A Framework for Entity Comparison

In this section, we present our formalisation of entity comparison. As a running example, consider a small subset  $G_{mov}$  of the YAGO graph [18] about the movie industry depicted in Figure 1. In our example, we would like to compare Quentin Tarantino and Martin Scorsese. By inspecting  $G_{mov}$  we can observe, for instance, that Tarantino and Scorsese are similar in that both of them are male, they both won an Academy Award and a Golden Globe Award, and they both acted in some of their own movies. In turn, they are different in that Tarantino directed *Reservoir Dogs*, whereas Scorsese directed *Taxi Driver*; furthermore, unlike Scorsese, Tarantino also won the Palme d’Or at the Cannes Film Festival, while Scorsese won an Emmy award, to which Tarantino was only nominated.

How can we formalise and automatically identify such similarities and differences? There has been significant recent work in the literature on discovering relationships between entities in an RDF graph [7, 11, 14]. Existing approaches

describe such relationships by means of explicit paths in the graph, which are then grouped and ranked. Using such an approach, we could view a similarity between entities as paths originating in those entities and converging into the same node; for instance, we could justify as a similarity the fact that both Tarantino and Scorsese are male by two paths leading to the node for male and starting from the nodes for Scorsese and Tarantino, respectively. In turn, we could justify a difference through the absence of such paths; for instance, the node for Emmy Award is reachable from the node for Scorsese but not from that for Tarantino. An important limitation of existing approaches, however, is that they cannot capture comparison at a *higher level of abstraction*; for instance, we cannot justify by means of explicit converging paths in a graph the fact that both Scorsese and Tarantino participated in a film as both actors and directors, where the specific names of those films are irrelevant.

In our framework we propose to capture similarities and differences using *queries* rather than explicit paths, where the presence of variables allows us to represent information at a higher level of abstraction. We start by formalising similarities. Given two entities in a graph, we view a similarity as a query having both entities as answers.

**Definition 1 (Similarity query).** *A similarity query for entities  $a$  and  $b$  in RDF graph  $G$  is a monadic connected query  $Q$  satisfying  $\{a, b\} \subseteq [Q]_G$ .*

For instance, the following queries  $Q_1$ – $Q_3$  are similarity queries for Tarantino and Scorsese in our example graph  $G_{mov}$ :

$$\begin{aligned} Q_1(X) &= \{(X, wonPrize, Academy\_Award)\}; \\ Q_2(X) &= \{(X, hasGender, male), (X, wonPrize, Academy\_Award)\}; \\ Q_3(X) &= \{(X, directed, Y), (X, actedIn, Y), (H\_Keitel, actedIn, Y)\}. \end{aligned}$$

These similarity queries can be interpreted as follows:  $Q_1$  says that both Scorsese and Tarantino received an Academy award, whereas  $Q_2$  additionally states that they are both male; in turn,  $Q_3$  states that they are both directors who acted in their own movies, in which Harvey Keitel was also part of the cast.

We next formalise the notion of a difference. Intuitively, given two entities in an RDF graph, a difference is a query having one of the entities as answer, but not the other. Furthermore, we are specially interested in differences that are *relevant to an identified similarity*, in the sense that they distinguish the entities based on an aspect that they have in common.

**Definition 2 (Difference query).** *Let  $a$  and  $b$  be entities in an RDF graph  $G$ . A difference query for  $a$  relative to  $b$  is a monadic connected query  $Q$  satisfying  $a \in [Q]_G$  and  $b \notin [Q]_G$ .*

*Additionally, let  $Q'$  be a similarity query for  $a$  and  $b$  in  $G$ . Then, we say that  $Q$  is a difference query modulo  $Q'$  if  $Q$  is a difference query for  $a$  relative to  $b$  and it holds that  $Q \subseteq Q'$ .*

For instance, the following query  $Q_4(X)$  is a difference query for Scorsese relative to Tarantino and modulo the similarity query  $Q_1(X)$  given before.

$$Q_4(X) = \{(X, \text{wonPrize}, \text{Academy\_Award}), (X, \text{wonPrize}, \text{Emmy\_Award})\}.$$

In turn, the following query is also a difference query for Scorsese relative to Tarantino, but it does not relate to any (non-trivial) similarity between them.

$$Q_5(X) = \{(X, \text{marriedTo}, Y)\}.$$

As we can see from the aforementioned examples, there may be multiple (even infinitely many) similarity and difference queries for a given pair of entities. Some of them are, however, more informative than others. In the case of similarity queries, it is natural to expect more specific queries to be more informative; for instance, it is natural to prefer our example query  $Q_2$  over  $Q_1$  since it better discriminates Tarantino and Scorsese from other directors, by ruling out those who won an Emmy but are female. In contrast, in the case of difference queries it is natural to favour more general queries over more specific ones; for instance,  $Q_5$  is more informative than the following query  $Q_6$  since it conveys the information that Scorsese is married, but Tarantino is not (or at least not known to be).

$$Q_6(X) = \{(X, \text{marriedTo}, I\_Rossellini)\}.$$

We now define these notions formally.

**Definition 3.** *Query  $Q$  is a most specific similarity query (MSSQ) for  $a$  and  $b$  in  $G$  if  $Q$  is a similarity query for  $a$  and  $b$  in  $G$ , and there is no similarity query  $Q'$  for  $a$  and  $b$  in  $G$  such that  $Q' \subset Q$ .*

*Query  $Q$  is a most general difference query (MGDQ) for  $a$  relative to  $b$  in  $G$  if  $Q$  is a difference query for  $a$  relative to  $b$  in  $G$ , and there is no difference query  $Q'$  for  $a$  relative to  $b$  in  $G$  such that  $Q \subset Q'$ . This definition extends to the notion of difference query modulo a similarity query in the obvious way.*

Intuitively, given two similarity queries  $Q$  and  $Q'$  for the same pair of entities, their conjunction is also a similarity query that is more specific than both of them. We will show in the following section that MSSQs for given entities and graph are unique modulo equivalence over the given input graph. As an example, we will show that the following query, which combines  $Q_2$  and  $Q_3$ , is a MSSQ for Scorsese and Tarantino in  $G_{mov}$ :

$$Q_7(X) = \{(X, \text{hasGender}, \text{male}), (X, \text{wonPrize}, \text{Academy\_Award}), \\ (X, \text{wonPrize}, \text{Golden\_Globe\_Award}), (X, \text{actedIn}, Y), \\ (X, \text{directed}, Y), (H\_Keitel, \text{actedIn}, Y)\}.$$

Indeed, query  $Q_8 = Q_7 \cup \{(X, \text{actedIn}, Z)\}$  is also a MSSQ but it is equivalent to  $Q_7$ .

In turn, both query  $Q_5$  and the following query  $Q_9$  are both MGDQs for Scorsese relative to Tarantino:

$$Q_9(X) = \{(X, Y, \text{Emmy\_Award})\}.$$

Furthermore, they are incomparable with respect to subsumption and hence, in contrast to MSSQs, we cannot formulate a uniqueness result for MGDQs.

## 4 Computing a Most Specific Similarity Query

In this section, we tackle the problem of computing a most specific similarity query. In particular, we present a polynomial time algorithm and then show, as a byproduct of the correctness proof, that MSSQs are unique up to equivalence.

Our algorithm relies on the notion of the *(tensor) product graph*, which is commonly exploited in Graph Theory and in Databases (under the name of *direct product* [19]). Given graphs  $G_1$  and  $G_2$ , the product  $G_1 \otimes G_2$  is a graph whose vertex set is the cartesian product of the vertices of  $G_1$  and  $G_2$ , and where two vertices in the product graph are connected by an edge if and only if their component elements are also related by an edge in the original graph. We next adapt the standard notion of product to RDF graphs. Intuitively, given entities  $a$ ,  $b$  and graph  $G$ , the connected subgraph of the product  $G \otimes G$  of  $G$  with itself represents the “largest common pattern” in the neighbourhoods of  $a$  and  $b$ .

**Definition 4 (Product graph).** *Let  $t_1 = (s_1, p_1, o_1)$  and  $t_2 = (s_2, p_2, o_2)$  be triples. The product of  $t_1$  and  $t_2$ , denoted as  $t_1 \otimes t_2$ , is the triple*

$$(\langle s_1, s_2 \rangle, \langle p_1, p_2 \rangle, \langle o_1, o_2 \rangle).$$

The product graph  $G_1 \otimes G_2$  of RDF graphs  $G_1$  and  $G_2$  is the set

$$\{t_1 \otimes t_2 \mid t_1 \in G_1 \text{ and } t_2 \in G_2\}.$$

For instance, the self-product  $G_{mov} \otimes G_{mov}$  of our example graph  $G_{mov}$  contains triples such as the following: <sup>5</sup>

$$\begin{aligned} &(\langle Q\_Tarantino, M\_Scorsese \rangle, \\ &\quad \langle wonPrize, wonPrize \rangle, \\ &\quad \langle Palme\_d'Or, Emmy\_Award \rangle) \end{aligned}$$

which is the product of triples  $(Q\_Tarantino, wonPrize, Palme\_d'Or)$  and  $(M\_Scorsese, wonPrize, Emmy\_Award)$ .

We are now ready to describe our algorithm (see Algorithm 1). Given  $a$ ,  $b$  and  $G$  as input, the first step is to compute the product graph  $G \otimes G$  and check whether  $\langle a, b \rangle$  occurs in a triple; if it doesn't then the algorithm fails and we can conclude that there is no query having both  $a$  and  $b$  as answers. If  $\langle a, b \rangle$  occurs in the product graph, then the algorithm computes the connected component  $G'$  in which it occurs. Given  $G'$ , we are interested in its canonical query having as answer variable the variable  $X_{\langle a, b \rangle}$  corresponding to  $\langle a, b \rangle$  in  $\text{Can}(G')$ . The result of this step is already a similarity query. In the last step, the algorithm

---

**Algorithm 1:** COMPUTE-MSSQ

---

**Input:** an RDF graph  $G$  and two entities  $a$  and  $b$  from  $G$ .

**Output:** a MSSQ for  $a$  and  $b$ .

- 1 Compute  $G \otimes G$ ;
  - 2 **if**  $\langle a, b \rangle$  does not occur in a triple in  $G \otimes G$  **then**
  - 3     **return fail**;
  - 4 Let  $G'$  be the connected component in  $G \otimes G$  that contains  $\langle a, b \rangle$ ;
  - 5 Construct the canonical query  $Q$  of  $G'$  with the answer variable  $X_{\langle a, b \rangle}$ ;
  - 6 Replace each variable  $X_{\langle c, c \rangle}$  in  $Q$  with  $c$ ;
  - 7 **return**  $Q$ .
- 

grounds all variables  $X_{\langle c, c \rangle}$  corresponding to nodes  $\langle c, c \rangle$  to  $c$  itself; this step is essential to ensure that the output similarity query is a most specific one.

Correctness of the algorithm follows from the following lemma.

**Lemma 1.** *Algorithm COMPUTE-MSSQ satisfies the following properties on input  $a$ ,  $b$  and  $G$ :*

1. *It fails if and only if there is no similarity query for  $a$  and  $b$  in  $G$ .*
2. *The output query  $Q'$  is a similarity query for  $a$  and  $b$  in  $G$  such that any similarity query  $Q''$  for  $a, b$  and  $G$  is homomorphically embeddable into  $Q'$ .*

*Proof.* 1. It is easy to see that a similarity query for  $a$  and  $b$  exists if and only if  $a$  and  $b$  appear as subjects, properties, or objects at the same time in  $G$ . This is equivalent to the fact that  $\langle a, b \rangle$  appears in a triple in  $G \otimes G$ . COMPUTE-MSSQ returns “fail” iff the latter is not the case.

2. We first show that  $\{a, b\} \subseteq [Q']_G$ . Define two valuations over  $\text{var}(Q')$ ,  $\nu_1$  and  $\nu_2$ , as follows: for every variable  $X_{\langle c, c' \rangle}$  in  $Q'$ ,  $\nu_1(X_{\langle c, c' \rangle}) = c$ , and  $\nu_2(X_{\langle c, c' \rangle}) = c'$ . We now show that  $G$  satisfies  $Q'$  under both  $\nu_1$  and  $\nu_2$ . Let  $(X_{\langle s_1, s_2 \rangle}, X_{\langle p_1, p_2 \rangle}, X_{\langle o_1, o_2 \rangle})$  be in  $Q'$ , then it follows by definition of  $Q'$  that  $(\langle s_1, s_2 \rangle, \langle p_1, p_2 \rangle, \langle o_1, o_2 \rangle) \in G'$ . Then by construction of  $G'$  we know that both  $(s_1, p_1, o_1)$  and  $(s_2, p_2, o_2) \in G$ . We then obtain that by definition of  $\nu_1$  and  $\nu_2$ :  $(\nu_i(X_{\langle s_1, s_2 \rangle}), \nu_i(X_{\langle p_1, p_2 \rangle}), \nu_i(X_{\langle o_1, o_2 \rangle})) \in G$ , for  $i = 1, 2$ . Hence,  $\nu_1$  and  $\nu_2$  are satisfying for  $Q'$  in  $G$ . We have  $\nu_1(X_{\langle a, b \rangle}) = a$  and  $\nu_2(X_{\langle a, b \rangle}) = b$ . Therefore,  $\{a, b\} \subseteq [Q']_G$ .

Let  $Q''(X)$  be an arbitrary similarity query for  $a$  and  $b$ . There are two satisfying valuations  $\nu_1$  and  $\nu_2$  over  $\text{var}(Q'')$  for  $Q''$  in  $G$  that map  $X$  to  $a$  and  $b$  respectively. We define  $\nu(Y) = \langle \nu_1(Y), \nu_2(Y) \rangle$  for  $Y$  a variable and  $\nu(e) = \langle e, e \rangle$  for  $e$  an entity. Since  $Q''$  is connected and  $\nu(X) = \langle a, b \rangle$ , the image of  $Q''$  under  $\nu$  is a connected subgraph in  $G \otimes G$  and thus is contained in  $G'$ . Since  $G'$  and  $Q'$  are isomorphic,  $\nu$  can be considered as a homomorphism from  $Q''$  to  $Q'$ .  $\square$

---

<sup>5</sup> Note that the product graph is strictly speaking not an RDF graph, but this is a technicality that is not important for our purposes.



Clearly, our algorithm works in polynomial time; in particular the size of the product graph  $G \otimes G$  is cubic in the size of  $G$ . Hence, using the previous Lemma we conclude the following.

**Theorem 1.** *COMPUTE-MSSQ is a polynomial time algorithm that returns a MSSQ for its input if one exists, and “fail” otherwise.*

Finally, note that the second statement in Lemma 1 ensures that the return query is, in fact, more specific than *any other* similarity query. Thus, it also follows from the lemma that MSSQs are unique up to equivalence.

**Corollary 1.** *If  $Q$  and  $Q'$  are MSSQs for  $a$  and  $b$  in RDF graph  $G$ , then  $Q \equiv Q'$ .*

We conclude by observing that the algorithm COMPUTE-MSSQ will compute, on our running example, a query that is significantly larger than (yet equivalent to)  $Q_7$  in the previous section. Indeed,  $Q_7$  is a *core* query in the sense that it cannot be further minimised while preserving equivalence.

## 5 Computing Most General Difference Queries

We now turn our attention to computing MGDQs. As already pointed out, MGDQs are not unique modulo equivalence and hence we focus on providing an algorithm that computes one of them.

In contrast to the case of computing MSSQs, we will not be able to provide a polynomial-time algorithm. In fact, we first show that the associated decision problem of checking whether a MGDQ exists is CONP-complete. This result stems from a characterisation of existence of MGDQs in terms of (non-)existence of homomorphisms.

In what follows we fix arbitrary entities  $a$  and  $b$  in an arbitrary graph  $G$ . We denote with  $Q_b$  to be the query  $(X_b, \text{Can}(G))$  and  $Q_a$  to be the query  $(X_a, P)$  with  $P$  the connected component of  $\text{Can}(G)$  including  $X_a$ .

**Lemma 2.** *A difference query for  $a$  relative to  $b$  in  $G$  exists if and only if there is no homomorphism from  $Q_a$  to  $Q_b$ .*

*Proof. Fix Me* ( $\Leftarrow$ ). The following properties hold for  $Q_a^C$ . It is (1) connected and (2)  $a \in \text{ans}(Q_a^C, G)$ . Moreover, since there is no homomorphism from  $Q_a^C$  to  $Q_b$ , it holds that (3)  $b \notin \text{ans}(Q_a^C, G)$ . Thus,  $Q_a^C$  is a difference query of  $a$  and  $b$ .

( $\Rightarrow$ ). Let  $Q(?X)$  be a difference query for  $a$  and  $b$ . It implies there is a satisfying variable assignment  $\nu$  for  $Q$  in  $G$  which can be regarded as a homomorphism from  $Q$  to  $Q_a^C$  (since  $Q$  is connected) with  $\nu(?X) = ?X_a$ . For the sake of contradiction, suppose there is a homomorphism  $h$  from  $Q_a^C$  to  $Q_b$ . This homomorphism can be regarded as a satisfying variable assignment for  $Q_a^C$  in  $G$ . Hence, the mapping  $h \circ \nu$  is a satisfying variable assignment for  $Q(?X)$  in  $G$  with  $h \circ \nu(x) = b$  which implies  $b \in \text{ans}(Q, G)$ , a contradiction with the fact that  $Q$  is a difference query for  $a$  and  $b$ .

Since homomorphism checking is a well-known NP-complete problem, the following result follows.

**Theorem 2.** *The problem of checking whether a difference query for a relative to  $b$  in  $G$  exists is CONP-complete.*

*Proof.* **FIX ME** It is known that checking existence of a homomorphism is in NP. Together with Lemma 2 it implies that existence of a difference query can be checked in CONP. We show the lower bound by reducing from the homomorphism problem for graphs to the complement of our problem. Let  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  be graphs which we can assume to be disjoint. We then construct an RDF graph  $G$  over the set of URIs  $V_1 \cup V_2 \cup \{a, b, e, e'\}$ , where  $\{a, b, e, e'\} \cap V_i = \emptyset, i = 1, 2$ , as the following set:

$$G = \{(u, e, v) \mid \langle u, v \rangle \in E_1 \cup E_2\} \cup \{(a, e', u) \mid u \in V_1\} \cup \{(b, e', v) \mid v \in V_2\}.$$

Let then  $Q_a^C$  be the connected component of the answer variable  $?X_a$  in the canonical query of  $G$ , and  $Q_b$  the canonical query of  $G$  with the answer variable  $?X_b$ . It is straightforward to show that there exists a homomorphism from  $G_1$  to  $G_2$  if and only there is a homomorphism from  $Q_a^C$  to  $Q_b$  (note that this homomorphism must map  $?X_a$  to  $?X_b$ ). Lemma 2 implies that this is equivalent to non-existence of a difference query for  $a$  and  $b$ .

In light of this result, there is no hope for a polynomial time algorithm for computing a MGDQ unless PTIME = NP. Therefore, we present a naive algorithm COMPUTE-MGDQ that issues a polynomial number of calls to an NP oracle and is therefore implementable in exponential time. In the first step, the algorithm computes  $Q_a$  (feasible in polynomial time). Then, it checks (using the oracle as per Lemma ??) whether  $Q_a$  is already a difference query. If it is not, then none can exist. If it is, then it may not be a most general one. Hence, the algorithm tries to make it more general by greedily removing triple patterns while checking (again using the oracle as per Lemma ??) whether the result is still a difference query.

Correctness is established in the following theorem.

**Theorem 3.** *Algorithm COMPUTE-MGDQ is feasible in exponential time and returns a MGDQ if one exists, and “fail” otherwise.*

*Proof.* **CHECK ME** By Lemma 2, if there is no most general difference query, then there is no homomorphism from  $Q_a^c$  to  $Q_b$ . According to Step 2, the algorithm returns “fail”. Let  $Q$  be the output of COMPUTE-MGDQ different from “fail”. This in particular means that no triple pattern can be removed from  $Q$  as a result of Step 4. The conditions on Steps 2 and 4 ensure that  $a \in \text{ans}(Q, G)$  and  $b \notin \text{ans}(Q, G)$ , i.e.,  $Q$  is a difference query for  $a$  and  $b$ . Most generality is implied from the fact that we greedily removed triple patterns from  $Q$  on Step 2. To show it, suppose there is a difference query  $Q''$  for  $a$  and  $b$  that is strictly more general than  $Q$ . Since  $a \in \text{ans}(Q'', G)$ , there is a satisfying assignment  $\nu$  for  $Q''$  in  $G$ . Let  $Q''_\nu$  be the canonical query of the image of  $Q''$  under  $\nu$ . For

---

**Algorithm 2:** Algorithm COMPUTE-MGDQ

---

**Input:** an RDF graph  $G$  and two entities  $a$  and  $b$  from  $G$ .

**Output:** a MGDQ for  $a$  and  $b$ .

```
1 Let  $Q := Q_a^c$ ;  
2 if there exists a homomorphism from  $Q$  to  $Q_b$  then  
3   return fail;  
4 foreach triple pattern  $tp$  in  $Q$  do  
5   Let  $Q' := Q \setminus \{tp\}$ ;  
6   if  $a \in \text{ans}(Q', G)$  and there is no homomorphism from  $Q'$  to  $Q_b$  then  
7     update  $Q := Q'$ ;  
8 return  $Q$ .
```

---

$Q''_\nu$  it holds that  $a \in \text{ans}(Q''_\nu, G)$  and there is no homomorphism from  $Q''_\nu$  to  $Q_b$ . Moreover,  $Q''_\nu$  is a strict subset of  $Q$ . Therefore there exists a triple pattern in  $Q$  that can be removed on Step 4, a contradiction. An exponential blow up is due to the fact that we need to check existence of a homomorphism on Steps 2 and 4.

## 6 Case Study

We have implemented a prototype system in Java that implements our algorithm COMPUTE-MSSQ for computing MSSQs. As a proof of concept, we have run the algorithm on a fragment of DBpedia [13] that captures the information corresponding to Wikipedia infoboxes—tables with a fixed structure used in Wikipedia to present the key information about an entity in a concise and structured way.<sup>6</sup> Infoboxes are located on the right-hand-side of Wikipedia pages that correspond to certain categories, such as people, organisations or geographical locations.

Entity comparison in Wikipedia could be implemented by comparing their infoboxes directly; such a tool would provide analogous functionality to that in existing comparison tools in Web portals, in the sense that the features to compare would be considered fixed. Figure 2 displays side by side the infoboxes corresponding to Brad Pitt and Tom Cruise, which are both fairly detailed. We can observe similarities such as their occupations and country of birth, or the fact that they have both been married and have children.

We tried our algorithm for Brad Pitt and Tom Cruise and the aforementioned fragment of DBpedia. We observed that the computed MSSQ provides much richer information than what can be obtained by direct inspection of the infoboxes. Since the resulting MSSQ is rather large, we concentrate on its subqueries of special interest. First, we notice that we generated all the aforementioned similarities that could be obtained by manual inspection of the infoboxes. In particular, we found that both Brad Pitt and Tom Cruise are:

---

<sup>6</sup> <https://en.wikipedia.org/wiki/Help:Infobox>

<b>Born</b>	William Bradley Pitt December 18, 1963 (age 53) <a href="#">Shawnee, Oklahoma, U.S.</a>	<b>Born</b>	Thomas Cruise Mapother IV July 3, 1962 (age 54) <a href="#">Syracuse, New York, U.S.</a>
<b>Occupation</b>	Actor • producer	<b>Occupation</b>	Actor, producer
<b>Years active</b>	1987–present	<b>Years active</b>	1981–present
<b>Works</b>	<a href="#">Filmography</a>	<b>Spouse(s)</b>	<a href="#">Mimi Rogers</a> ( <a href="#">m.</a> 1987; <a href="#">div.</a> 1990) <a href="#">Nicole Kidman</a> ( <a href="#">m.</a> 1990; <a href="#">div.</a> 2001) <a href="#">Katie Holmes</a> ( <a href="#">m.</a> 2006; <a href="#">div.</a> 2012)
<b>Home town</b>	<a href="#">Springfield, Missouri</a>	<b>Children</b>	3
<b>Spouse(s)</b>	<a href="#">Jennifer Aniston</a> ( <a href="#">m.</a> 2000; <a href="#">div.</a> 2005) <a href="#">Angelina Jolie</a> ( <a href="#">m.</a> 2014; separated 2016)	<b>Relatives</b>	<a href="#">William Mapother</a> (cousin)
<b>Children</b>	6	<b>Website</b>	<a href="#">tomcruise.com</a> 
<b>Relatives</b>	<a href="#">Douglas Pitt</a> (brother)		

**Fig. 2.** Wikipedia infoboxes for actors Brad Pitt (left) and Tom Cruise (right).

- both actors and producers, as witnessed by the subquery

$$\{(X, \textit{occupation}, \textit{Actor}), (X, \textit{occupation}, \textit{Producer})\};$$

- were born in the U.S., as witnessed by

$$\{(X, \textit{birth\_place}, Y_1), (Y_1, \textit{country}, \textit{United\_States})\};$$

- were married, have kids and relatives, as witnessed by

$$\{(X, \textit{children}, Y_2), (X, \textit{spouse}, Y_3), (X, \textit{relatives}, Y_4)\}.$$

However, the computed MSSQ also contains plenty of additional useful information. For instance, both Pitt and Cruise:

- were married to U.S. actresses, as witnessed by

$$\{(X, \textit{spouse}, Y_3), (Y_3, \textit{nationality}, \textit{United\_States}), (Y_3, \textit{occupation}, \textit{Actress})\};$$

- were born in cities that are both the administrative centers and largest cities of their respective counties:

$$\{(X, \textit{birth\_place}, Y_1), (Y_1, \textit{county}, Y_5), \\ (Y_5, \textit{largest\_city}, Y_1), (Y_5, \textit{seat}, Y_1), (Y_1, \textit{settlement\_type}, \textit{City})\};$$

- were married to actresses who were also married to musicians:

$$\{(X, \textit{spouse}, Y_3), (Y_3, \textit{occupation}, \textit{Actress}), \\ (Y_3, \textit{spouse}, Y_6), (Y_6, \textit{occupation}, \textit{Musician})\};$$

To sum up, even using only DBpedia data capturing Wikipedia infoboxes, we are able to significantly enhance the explicit contents of fairly comprehensive infoboxes and exploit the graph nature of the data to discover “deeper-level” similarities between the entities of interest. We envision that our approach could even be more useful if the whole of DBpedia had been considered, especially in the case where the infoboxes corresponding to the entities of interest are rather minimalistic and hence do not provide sufficiently many features to compare.

## 7 Related Work

There is a growing interest in techniques for discovering and explaining relationships between entities in an RDF graph [7, 11, 14]. These approaches are based on computing paths in the input graph connecting the input entities. Such paths are first computed via standard graph traversal algorithms, and then ranked according to certain structural and/or statistical measures [7]. We note that the problem of finding connections between entities is orthogonal to that of computing similarities and differences between them. Furthermore, as already argued, the natural adaptations of such techniques to our setting do not allow for entity comparison at a sufficiently high level of abstraction.

Computation of both similarity and difference queries can be seen as an instance of the more general problem of *Query Reverse Engineering (QRE)* in databases. An input to QRE is a database instance, a set of *positive examples* (i.e., elements that must be in the query result) and also in some cases a set of *negative examples* (i.e., elements that must not be included in the query result). The QRE problem for a query language  $\mathcal{L}$  is to decide whether an  $\mathcal{L}$ -query exists whose answers satisfy the given constraints imposed by positive and negative examples over the input database instance. This problem has been studied for regular languages over strings [1], queries over relational databases [20, 21, 23, 25], XML queries [9, 17], graph database queries [6] and SPARQL queries over RDF graphs [2]. QRE is known to be CONEXPTIME-complete for conjunctive queries over relational databases [4, 19]. When applied to our setting, this result implies CONEXPTIME-completeness of the following problem: given an RDF graph, and sets of entities  $A$  and  $B$  in  $G$ , does there exist a difference query for  $A$  relative to  $B$  in  $G$ , where the definition of a difference query is extended to sets of entities in the obvious way. QRE for RDF graphs was first studied in [2], where the complexity analysis of different variations of the problem is provided for SPARQL queries allowing for the AND, FILTER and OPT operators.

Computing MSSQs is also related to (a variant of) the problem of computing the *Least Common Subsumer* between concepts in Description Logics (DLs) [3]. Specifically, given entities  $a$  and  $b$ , we could cast our problem as that of finding the least (i.e., most specific modulo subsumption) DL concept that contains both  $a$  and  $b$  as instances. An important difference with our setting is that DL concepts in logics such as  $\mathcal{EL}$  and  $\mathcal{ALC}$  can only capture conjunctive queries that are both constant-free and tree-shaped. In this sense, our query language is more expressive, as it allows for arbitrarily-shaped connected CQs. The additional

expressivity turns out to be critical: while a least DL concept may not exist (e.g., if the input graph has cycles then the least concept could be infinite), our algorithm in Section 4 ensures that a MSSQ is always finite and can be computed in polynomial time.

Finally, it is worth mentioning that there has been a lot of work on *similarity measures* for computing a numeric score that estimates how similar two entities in a graph are [8,12,24]; this has applications, for instance, in discovering entities that are similar to a given one (i.e., those with the highest similarity score). Please note that we are considering a very different problem since our focus is on *describing* similarities and differences in a declarative way.

## 8 Conclusion and Future Work

- Extend the implementation to account also for difference computation. For this, need for practical algorithms.
- Develop your prototype into an entity comparison tool. This implies dealing with practical issues such as size of output, verbalising output...
- Consider more expressive query languages. In particular, inequalities, numeric comparisons,... Refer back to Wikipedia example.

## References

1. D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.
2. M. Arenas, G. I. Diaz, and E. V. Kostylev. Reverse engineering SPARQL queries. In *Proceedings of the 25th International Conference on World Wide Web*, pages 239–249. International World Wide Web Conferences Steering Committee, 2016.
3. F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In *Annual Conference on Artificial Intelligence*, pages 99–113. Springer, 2002.
4. P. Barcelo and M. Romero. The complexity of reverse engineering problems for conjunctive queries. In *Proceedings of the 20th International Conference on Database Theory*, page to appear. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
5. F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, and J. Morissette. Bio2rdf: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, 41(5):706–716, 2008.
6. A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *18th International Conference on Extending Database Technology (EDBT)*, 2015.
7. G. Cheng, Y. Zhang, and Y. Qu. Explass: exploring associations between entities via top-k ontological patterns and facets. In *International Semantic Web Conference*, pages 422–437. Springer, 2014.
8. S.-S. Choi, S.-H. Cha, and C. C. Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.
9. S. Cohen and Y. Y. Weiss. Learning tree patterns from example graphs. In *LIPICs-Leibniz International Proceedings in Informatics*, volume 31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

10. S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C proposed recommendation, 21 march 2013. *World Wide Web Consortium*. <https://www.w3.org/TR/sparql11-query/> (last visited October 1, 2016), 2013.
11. P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. Relfinder: Revealing relationships in RDF knowledge bases. In *International Conference on Semantic and Digital Media Technologies*, pages 182–187. Springer, 2009.
12. A. Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008)*, Christchurch, New Zealand, pages 49–56, 2008.
13. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.
14. J. Lehmann, J. Schüppel, and S. Auer. Discovering unknown connections-the dbpedia relationship finder. *CSSW*, 113:99–110, 2007.
15. G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.
16. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.
17. S. Staworko and P. Wiecek. Learning twig and path queries. In *Proceedings of the 15th International Conference on Database Theory*, pages 140–154. ACM, 2012.
18. F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A large ontology from Wikipedia and Wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.
19. B. ten Cate and V. Dalmau. The Product Homomorphism Problem and Applications. In *18th International Conference on Database Theory (ICDT 2015)*, pages 161–176, 2015.
20. Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 535–548. ACM, 2009.
21. Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23(5):721–746, 2014.
22. R. W. White and R. A. Roth. Exploratory search: beyond the query-response paradigm (synthesis lectures on information concepts, retrieval & services), 2009.
23. M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse engineering complex join queries. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 809–820. ACM, 2013.
24. P. Zhao, J. Han, and Y. Sun. P-rank: a comprehensive structural similarity measure over information networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 553–562. ACM, 2009.
25. M. M. Zloof. Query-by-example: A data base language. *IBM systems Journal*, 16(4):324–343, 1977.