RDF graph anonymization robust to data linkage

Abstract. Privacy is a major concern when publishing new datasets in the context of Linked Open Data (LOD). A new dataset published in the LOD is indeed exposed to privacy breaches due to the linkage to objects already present in the other datasets of the LOD. In this paper, we focus on the problem of building *safe* anonymizations of an RDF graph to guarantee that linking the anonymized graph with any external RDF graph will not cause privacy breaches. Given a set of privacy queries as input, we study the data-independent safety problem and the sequence of anonymization operations necessary to enforce it. We provide sufficient conditions under which an anonymization instance is safe given a set of privacy queries. Additionally, we show that our algorithms for RDF data anonymization are robust in the presence of `sameAs` links that can be explicit or inferred by additional knowledge.

Keywords: Linked Open Data · Data privacy · RDF anonymization.

1 Introduction

Since its inception, the Linked Open Data (LOD) paradigm has allowed to publish data on the Web and interconnect uniquely identified objects by realizing widely open information exchange and data sharing. The LOD cloud is rapidly growing and contains 1,231 RDF graphs connected by 16,132 links (as of June 2018).[1] Since 2007, the number of RDF graphs published in the LOD has seen an increase of about two orders of magnitude.

Nevertheless, the participation of many organizations and institutions to the LOD movement is hindered by individual privacy concerns. Personal data are ubiquitous in many of these data sources and recent regulations about personal data, such as the EU GDPR[2] as well as regulations for healthcare and governance data, make these organizations reluctant to publish their data in the LOD.

While there has been some effort [13, 19] to bring data anonymization techniques from the relational database world to the LOD, such as variations of k-anonymity [20, 16, 14], most of the state of the art is mainly based on differential privacy techniques for relational data [6, 15].

However, differential privacy is not a perfect match for Linked Data, focusing more on statistical integrity rather than accurate, qualitative query results which represents the main usage of Linked Data through SPARQL endpoints [**?**,17]. Differential privacy is indeed useful whenever the aggregate results of data analysis (such as statistics about groups of individuals) can be seamlessly published. Whereas this is highly desirable in many applications, it becomes not sufficient in privacy-preserving data publishing (PPDP) [7] scenarios where the privacy of individuals need to be protected while at the same time ensuring that the

[1] See https://lod-cloud.net/
[2] See https://eugdpr.org/

published data can be utilized in practice. Whereas the underpinnings of PPDP under its most prominent form, such as anonymization, have been widely studied for relational data (see [7] for a comprehensive survey), the theoretical foundations for PPDP in the context of Linked Data have only been recently laid out in [10] by focusing on the computational complexity of checking whether requirements for Linked Data anonymization are fulfilled. In this paper, we build upon the theoretical framework of [10] by focusing on the linkage safety requirement and present practical algorithms to compute the anonymization operations needed to achieve such a requirement when a graph G is linked to external graphs in the LOD. By relying on the computational complexity of the linkage safety problem, which is AC_0 in data complexity under the *open-world* assumption, we address the problem of actually computing a safety-compliant sequence of anonymization operations setting up their guarantees against linkage attacks. In doing this, we also devote special care to `:sameAs` links (i.e. links expressed in RDF syntax) that can be either explicit in the original graph G linking to entities in external graphs or derived by inference mechanisms on G itself.

In particular, this approach exhibits two distinguishing features. First, it is *query-based* since the privacy policies as well as the anonymization operations are specified by means of conjunctive queries and updates in SPARQL, respectively. Second, our approach is *data-independent* since, given a privacy policy (specified as a set of privacy queries), our algorithms produce anonymization operations (under the form of delete and update queries) with the guarantee that their application to *any RDF graph* will satisfy the safety requirement.

Our main contributions can be summarized as follows: we first ground the linkage safety problem to the sequence of anonymization operations necessary to enforce it by providing a novel *data-independent* definition of safety; such a definition considers a set of privacy queries as input and does not look at the actual graph instances (Section 4); as such, it departs from the basic definition of linkage safety of [10]. We then provide sufficient conditions under which an anonymization instance is safe given a set of privacy queries and design an anonymization algorithm that solves the above query-based safety requirement and study its runtime complexity (Section 5). Next, we introduce `:sameAs` links and show that our algorithm is robust to them, modulo slight changes that do not affect its overall complexity (Section 6) and finally, we provide a quick discussion about the evaluation of our algorithms and the remaining utility of the anonymized graphs (Section 7) that confirms the good behavior of our framework in practice. Related work is discussed in Section 2, and we provide the necessary background in Section 3. We conclude the paper in in Section 8. All proofs and implementations are available online in a companion appendix.[3]

2 Related work

A query-based approach to privacy-preserving RDF data publishing has been presented in [4], in which the focus was to check the compatibility between a

[3] See https://www.dropbox.com/s/csg35k1qlx91epl/appendix.zip

privacy policy and an utility policy (both specified as queries) and to build anonymizations preserving the answers to a set of utility queries (when compatibility is ensured). However, the above approach suffers from the lack of resilience against privacy breaches caused by linking external datasets, which is clearly a recurrent situation in the LOD. In this paper, we address this problem by considering explicit `sameAs` links as well as implicit `sameAs` links that can be inferred by additional knowledge.

In line with existing works [5, 18, 9] on safety models defined in terms of secret or privacy queries for relational data, a query-based safety model for RDF data has been introduced in [10] where linking RDF graphs is reduced to their union. Several results are provided on the computational complexity of the decision problem consisting in checking whether an anonymization of an RDF graph is safe w.r.t a given privacy policy. In our paper, we slightly extend the considered safety model and we address the data-independent construction problem underpinning safety, i.e. how to produce a sequence of update operations that are safe *for any RDF graph*, given a privacy policy expressed as queries.

Compared to existing approaches based on k-anonymity [13, 19], we focus on generalizations that replace constants by blank nodes. We have also shown that in some cases, triple suppressions are required in addition to generalizations for guaranteeing *safe* anonymizations.

Privacy-preserving record linkage has been recently considered in [21] as the problem of identifying and linking records that correspond to the same real-world entity without revealing any sensitive information about these entities. For preserving privacy while allowing the linkage, masking functions are proposed to transform original data in such a way that there exists a specific functional relationship between the original data and the masked data. The problem of privacy-preserving record linkage is a difficult problem that is significantly different from the privacy-preserving data publishing problem considered in this paper, in which `sameAs` links are input of the anonymization process.

3 Formal background

We recall the usual concepts for RDF graphs and SPARQL queries as formalized in [11]. Let $\mathbf{I}$, $\mathbf{L}$ and $\mathbf{B}$ be countably infinite pairwise disjoint sets representing respectively *IRIs*, *literals* and *blank nodes*. IRIs (Internationalized Resource Identifiers) are standard identifiers used for denoting any Web resource described in RDF within the LOD. We denote by $\mathbf{T} = \mathbf{I} \cup \mathbf{L} \cup \mathbf{B}$ the set of *terms*, in which we distinguish *constants* (IRIs and literals) from *blank nodes*, which are used to model unknown IRIs or literals like in [8, 2] and correspond to *labeled nulls* [1].

We also assume an infinite set $\mathbf{V}$ of variables disjoint from the above sets. Throughout this paper, we adhere to the SPARQL conventions: variables in $\mathbf{V}$ are prefixed with a question mark (`?`), IRIs in $\mathbf{I}$ are prefixed with a colon (`:`), blank nodes in $\mathbf{B}$ are prefixed with an underscore and a colon (`_:`).

Definition 1 (RDF graph and graph pattern). *An* RDF graph *is a finite set of* RDF triples (s, p, o)*, where* $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{B})$*. A* triple

pattern *is a triple* $(s, p, o) \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{B} \cup \mathbf{V})$. *A* graph pattern *is a finite set of triple patterns.*

We can now define the three types of queries that we consider. Definition 2 corresponds to *conjunctive queries* (in fact, a slight restriction of the standard definition of graph pattern queries) and will be the basis for formalizing the sensitive information that must not be disclosed. Definition 6 corresponds to *counting queries* which will model a form of utility that it may be useful to preserve for analytical tasks. Finally, Definition 7 describes *update queries*, modeling the anonymization operations handled in our framework.

Definition 2 (Conjunctive query). *A* conjunctive query Q *is defined by an expression* `SELECT` $\bar{x}$ `WHERE` $GP(\bar{x}, \bar{y})$ *where* $GP(\bar{x}, \bar{y})$*, also denoted* $\mathsf{body}(Q)$*, is a graph pattern without blank nodes and* $\bar{x} \cup \bar{y}$ *is the set of its variables, among which* $\bar{x}$ *are the* result *variables, and the subset of variables in predicate position is disjoint from the subset of variables in subject or object position. A conjunctive query* Q *is alternatively written as a pair* $Q = \langle \bar{x}, GP \rangle$*. A* boolean query *is a query of the form* $Q = \langle \emptyset, GP \rangle$*.*

Note that considering graph patterns without blank nodes to define conjunctive queries is not a restriction since normative SPARQL query evaluation treats blank nodes in a query as variables. Also, conjunctive queries with variables in predicate position are allowed, if such variables do not appear in a subject or object position. This ensures that within a conjunctive query, all occurrences of a given variable are in the same connected component (see Definition 3).

Definition 3 (Connected components of a query). *Given a conjunctive query* $Q = \langle \bar{x}, GP \rangle$*, let* $G_Q = \langle N_Q, E_Q \rangle$ *be the undirected graph defined as follows: its nodes* N_Q *are the distinct variables and constants appearing in subject or object position in* GP*, and its edges* E_Q *are the pairs of nodes* (n_i, n_j) *such that there exists a triple* (n_i, p, n_j) *or* (n_j, p, n_i) *in* GP*.*

Each subgraph SG_Q *of* G_Q *corresponds to the subgraph of* $\mathsf{body}(Q)$ *made of the set of triples* (s, p, o) *such that either* (s, o) *or* (o, s) *is an edge of* SG_Q*. By slight abuse of notation, we will* call the connected components *of the query* Q *the (disjoint) subsets of* $GP = \mathsf{body}(Q)$ *corresponding to the connected components of* G_Q*. A connected component* GP_C *of the query* Q *is called* boolean *when it contains no result variable.*

Example 1. Let Q be the following query in the SPARQL syntax where `a` is a shorthand for `rdf:type` :

```
SELECT ?x ?y WHERE { ?x  :seenBy ?z.      ?z :specialistOf   ?y.
                     ?v  a       :VIP.    ?v :isHospitalized true. }
```

Q has two connected components, namely GP_1 and GP_2:

```
GP1 = { ?x :seenBy ?z.      ?z :specialistOf   ?y. }
GP2 = { ?v a       :VIP.    ?v :isHospitalized true. }
```

Critical terms are defined in Definition 4. They will play an important role in our main algorithm presented in Section 5.

Definition 4 (Critical terms). *A variable (resp. constant) in subject or object position having several occurrences within the body of a query is called a* join variable *(resp.* join constant*). We name join variables, join constants and result variables of a query as its* critical terms*.*

The evaluation of a query $Q = \langle \bar{x}, GP \rangle$ over an RDF graph G consists in finding mappings μ assigning the variables in GP to terms such that the set of triples, denoted $\mu(GP)$, obtained by replacing with $\mu(z)$ each variable z appearing in GP, is included in G. The corresponding answer is defined as the tuple of terms $\mu(\bar{x})$ assigned by μ to the result variables.

Definition 5 (Evaluation of a conjunctive query). *Let $Q = \langle \bar{x}, GP \rangle$ be a conjunctive query and let G be an RDF graph. The* answer set of Q over G *is defined by :* $\mathsf{Ans}(Q, G) = \{\mu(\bar{x}) \mid \mu(GP) \subseteq G\}$.

Definition 6 (Counting query). *Let Q be a conjunctive query. The query* $\mathsf{Count}(Q)$ *is a* counting query*, whose answer over a graph G is defined by:* $\mathsf{Ans}(\mathsf{Count}(Q), G) = |\,\mathsf{Ans}(Q, G)|$

We now define an additional ingredient: *update queries*. Intuitively, an update query `DELETE` $D(\bar{x})$ `INSERT` $I(\bar{y})$ `WHERE` $W(\bar{z})$ `isNotBlank`$(\bar{b})$ executed on a graph G searches for the instances of the graph pattern $W(\bar{z})$ in G, then deletes the instances of $D(\bar{x})$ and finally inserts the $I(\bar{y})$ part. Specifying `isNotBlank`$(\bar{b})$, which translates to `FILTER(!isBlank(b1) && ... && !isBlank(bn))` in SPARQL, rules out instances where a variable in $\bar{b}$ is mapped to a blank node. The `isNotBlank` operator will be used in Algorithm 1 to avoid replacing the images of critical terms that are already blank nodes.

Definition 7 (Update query). *An* update query *(or* update operation*)* Q_u *is defined by* `DELETE` $D(\bar{x})$ `INSERT` $I(\bar{y})$ `WHERE` $W(\bar{z})$ `isNotBlank`$(\bar{b})$ *where D (resp. W) is a graph pattern whose set of variables is $\bar{x}$ (resp. $\bar{z}$) such that $\bar{x} \subseteq \bar{z}$; and I is a graph pattern where blank nodes are allowed, whose set of variables is $\bar{y}$ such that $\bar{y} \subseteq \bar{z}$.* `isNotBlank`$(\bar{b})$ *is a parameter where $\bar{b}$ is a set of variables such that $\bar{b} \subseteq \bar{z}$. The evaluation of Q_u over an RDF graph G is defined by:*

$$\begin{aligned}\mathsf{Result}(Q_u, G) = G \setminus \{\mu(D(\bar{x})) | \mu(W(\bar{z})) \subseteq G \wedge \forall x \in \bar{b}, \mu(x) \notin \mathbf{B}\} \\ \cup \{\mu'(I(\bar{y})) | \mu(W(\bar{z})) \subseteq G \wedge \forall x \in \bar{b}, \mu(x) \notin \mathbf{B}\}\end{aligned}$$

where μ' is an extension of μ renaming blank nodes from $I(\bar{y})$ to fresh blank nodes, i.e. a mapping such that $\mu'(x) = \mu(x)$ when $x \in \bar{z}$ and $\mu'(x) = b_{new} \in \mathbf{B}$ otherwise. When $\bar{b}$ is empty, we avoid writing `isNotBlank`$(\emptyset)$*. Similarly, when* `INSERT` $I(\bar{y})$ *is empty, the query is written* `DELETE` $D(\bar{x})$ `WHERE` $W(\bar{z})$*. The application of an update query Q_u on a graph G is alternatively written $Q_u(G) = \mathsf{Result}(Q_u, G)$. This notation is naturally extended to a sequence of operations $O = \langle Q_u^1, \ldots Q_u^n \rangle$ by $O(G) = Q_u^n(\ldots(Q_u^1(G))\ldots)$.*

4 Safety model

We generalize the definition of a *safe anonymization* introduced in [10] as follows: an RDF graph is safely anonymized if it does not disclose any *new* answer to a set of privacy queries when it is joined with any external RDF graph. Additionally, compared to [10], we define a notion of *data-independent* safety for a sequence of anonymization operations independently of *any* RDF graph.

Given an RDF graph G, a sequence O of update queries called *anonymization operations* and a set $\mathcal{P}$ of conjunctive *privacy queries*, the safety of the *anonymization instance* $(G, O, \mathcal{P})$ is formally defined as follows.

Definition 8 (Safe anonymization instance). *An anonymization instance* $(G, O, \mathcal{P})$ *is* safe iff *for every RDF graph* G'*, for every* $P \in \mathcal{P}$ *and for every tuple of constants* $\bar{c}$*, if* $\bar{c} \in \mathsf{Ans}(P, O(G) \cup G')$ *then* $\bar{c} \in \mathsf{Ans}(P, G')$*.*

Notice that the *safety* property is stronger than the *privacy* property defined in [10, 4] which requires that for every privacy query P, $\mathsf{Ans}(P, O(G))$ does not contain any tuple made only of constants. Privacy can thus be seen as a degenerated case of safety where the external RDF graph G' is empty.

In contrast with [10], the safety problem that we consider is data-independent and is a construction problem. Given a set of privacy queries, the goal is to build a sequence of anonymization operations guaranteed to produce a safe anonymization when applied to any RDF graph, as defined below.

Definition 9 (Safe sequence of anonymization operations). *Let* O *be a sequence of anonymization operations, let* $\mathcal{P}$ *be a set of privacy queries,* O *is safe for* $\mathcal{P}$ *iff* $(G, O, \mathcal{P})$ *is safe* for every RDF graph G*.*

Problem 1. The data-independent SAFETY problem.
Input : $\mathcal{P}$ a set of privacy queries
Output: A sequence O of update operations such that O is safe for $\mathcal{P}$.

Our approach to solve Problem 1 is to favour *whenever possible* update operations that replace IRIs and literals by blank nodes over update operations that delete triples. We exploit the standard semantics of blank nodes that interprets them as existential variables in the scope of local graphs. As a consequence, two blank nodes appearing in two distinct RDF graphs cannot be equated.

The privacy-preserving approach described in [4] is also data-independent but is based on deleting operations that may lead to unsafe anonymizations, as shown in Example 2.

Example 2. Let consider the following privacy query P stating that IRIs of people seen by a specialist of a disease should not be disclosed.

```
SELECT ?x WHERE { ?x :seenBy ?y.   ?y :specialistOf ?z. }
```

Let the RDF graph to anonymize be G that is made of the following triples:

```
:bob :seenBy :mary.  :mary :specialistOf :cancer.   :mary :worksAt :hospital1.
:ann :seenBy :mary.  :jim  :worksAt :hospital1.
```

Let O_1 be the update query deleting all the `:seenBy` triples.

```
DELETE { ?x :seenBy ?y. } WHERE { ?x :seenBy ?y. }
```

The resulting anonymized RDF graph $O_1(G)$ is made of the following triples:

```
:mary :specialistOf :cancer.   :mary :worksAt :hospital1.
:jim  :worksAt :hospital1.
```

O_1 preserves privacy (since the evaluation of the privacy query P against $O_1(G)$ returns no answer). However, O_1 is not safe since the union of $O_1(G)$ with an external RDF graph G' containing the triple (`:bob`, `:seenBy`, `:mary`) will provide `:bob` as an answer (which is not an answer of P against G' alone).

Example 2 shows that the problem for safety comes from a possible join between an internal and an external constant (`:mary` here). This can be avoided by replacing critical constants by blank nodes. Different strategies exist, as detailed in various examples in the companion appendix. Example 3 illustrates the strategy considered in Algorithm 1 (see Section 5).

Example 3. Consider the following update query O_2:

```
DELETE {?x   :seenBy ?y.   ?y   :specialistOf ?z.}
INSERT {_:b1 :seenBy _:b2. _:b2 :specialistOf ?z.}
WHERE  {?x   :seenBy ?y.   ?y   :specialistOf ?z.}
```

The result RDF graph $O_2(G)$ is made of the following triples and is *safe*:

```
_:b1 :seenBy _:b2.  _:b2 :specialistOf :cancer. :mary :worksAt :hospital1.
_:b3 :seenBy _:b4.  _:b4 :specialistOf :cancer. :jim  :worksAt :hospital1.
```

It is worth noticing that the result of the counting query $\mathsf{Count}(P)$ is preserved i.e. it returns the same value as when evaluated on the original RDF graph G. Many other utility queries are preserved, such as for instance the one asking for who works at which hospital.

5 Safe anonymization of an RDF graph

In this section, we provide an algorithm that computes a solution to the SAFETY problem. To this end, we first prove a *sufficient condition* (Theorem 1) guaranteeing that an anonymization instance is safe, then we define an algorithm based on this condition. We have to extend the definition of a mapping, which is now allowed to map constants to blank nodes.

Definition 10. *An* anonymization mapping μ *is a function* $\mathbf{V} \cup \mathbf{I} \cup \mathbf{L} \to \mathbf{T}$. *For a triple* $\tau = (s, p, o)$ *we write* $\mu(\tau)$ *for* $(\mu(s), \mu(p), \mu(o))$.

Theorem 1 is progressively built on two conditions that must be satisfied by all the connected components of the privacy queries. The intuition of condition (i) is that if all the images of *critical terms* are blank nodes, it is impossible to graft external pieces of information to the anonymized graph as they cannot have common blank nodes. Condition (ii) deals with boolean connected components with no result variable.

Theorem 1. *An anonymization instance* $(G, O, \mathcal{P})$ *is* safe *if the following conditions hold for every connected component* GP_c *of all privacy queries* $P \in \mathcal{P}$*:*

(i) *for every critical term* x *of* GP_c*, for every triple* $\tau \in GP_c$ *where* x *appears, for each anonymization mapping* μ *s.t.* $\mu(\tau) \in O(G)$*,* $\mu(x) \in \mathbf{B}$ *holds;*
(ii) *if* GP_c *does not contain any result variable, then there exists a triple pattern of* GP_c *without any image in* $O(G)$ *by an anonymization mapping.*

We are now able to design an anonymization algorithm that solves the SAFETY problem. Algorithm 1 computes a sequence[4] of operations O for a privacy policy $\mathcal{P}$ such that O is safe for $\mathcal{P}$. Operations are computed for each connected component of each privacy query from $\mathcal{P}$. The crux is to turn conditions (i) and (ii) into update queries (Definition 7).

The starting point of Algorithm 1 is to compute joins variables and constants (Lines 5 to 8) then to compute critical terms by adding the result variables (Lines 9 to 10). Note that critical terms *do not* include variables in predicate position, except when these variables are also result variables of the considered query. The update queries that replace the images of critical terms by blank nodes are built from Line 14 to Line 17. The subtle point is that, in order to guarantee that condition (i) of Theorem 1 is satisfied on any updated RDF graph, as many update queries as the connected subsets of the component GP_c need to be constructed. Considering these subsets in decreasing order of cardinality (Line 11) and using the `isNotBlank`$(\bar{x}')$ construct introduced in Definition 7 (Line 17) guarantees that all the images of a critical term in a given RDF graph will be replaced only once. Non-connected subsets of GP_c are skipped because their own connected components are handled afterwards. Finally, if the connected component under scrutiny is boolean, one of its triple is deleted (Line 20), according to condition (ii).

When applied to the privacy query considered in Example 2, operation O_2 reported in Example 3 is the first one generated at Line 17. When applied to Example 1, Algorithm 1 will sequentially generate anonymization operations starting from those replacing the images of all the variables by blank nodes (since all its variables are critical), followed by those deleting all the triples (possibly modified by the preceding operations) corresponding to one of the triple patterns (`?v a :VIP` or `?v :isHospitalized true`) in the boolean connected component.

Note that anonymizations may create an RDF graph where some properties have been replaced by blank nodes (in the case where a result variable was appearing in as a predicate in the body of a policy query). In this case, the output is a *generalized* RDF graph[5]. Theorem 2 states the soundness and computational complexity of Algorithm 1.

Theorem 2. *Let* $O =$ `find-safe-ops`$(\mathcal{P})$ *be the sequence of anonymization operations returned by Algorithm 1 applied to the set* $\mathcal{P}$ *of privacy queries:* O *is safe*

[4] The + operator denotes the concatenation of sequences.
[5] As specified in Section 7 of the RDF 1.1 Specification: https://www.w3.org/TR/rdf11-concepts/#section-generalized-rdf.

for $\mathcal{P}$. The worst-case computational complexity of Algorithm 1 is exponential in in the size of $\mathcal{P}$.

Algorithm 1: Find update operations to ensure safety

Input : a privacy policy $\mathcal{P}$ of queries $P_i = \langle \bar{x}_i, GP_i \rangle$
Output : a sequence of operations O which is safe for $\mathcal{P}$

1 **function** `find-safe-ops(`$\mathcal{P}$`):`
2 Let $O = \langle\,\rangle$;
3 **for** $P_i \in \mathcal{P}$ **do**
4 **forall** *connected components* $GP_c \subseteq GP_i$ **do**
5 Let $I := [\,]$;
6 **forall** $(s, p, o) \in GP_c$ **do**
7 **if** $s \in \mathbf{V} \vee s \in \mathbf{I}$ **then** $I[s] = I[s] + 1$;
8 **if** $o \in \mathbf{V} \vee o \in \mathbf{I} \vee o \in \mathbf{L}$ **then** $I[o] = I[o] + 1$;
9 Let $\bar{x}_c := \{v \mid v \in \bar{x}_i \wedge \exists \tau \in GP_c \text{ s.t. } v \in \tau\}$;
10 Let $T_{crit} := \{t \mid I[t] > 1\} \cup \bar{x}_c$;
11 Let $SGP_c = \{X \mid X \subseteq GP_c \wedge X \neq \emptyset \wedge X \text{ is connected}\}$ ordered by decreasing size;
12 **forall** $X \in SGP_c$ **do**
13 Let $X' := X$ and $\bar{x}' = \{t \mid t \in T_{crit} \wedge \exists \tau \in X \text{ s.t. } t \in \tau\}$;
14 **forall** $x \in \bar{x}'$ **do**
15 Let $b \in \mathbf{B}$ be a fresh blank node;
16 $X' := X'[x \leftarrow b]$;
17 $O := O + \langle$`DELETE` X `INSERT` X' `WHERE` X `isNotBlank`$(\bar{x}')\rangle$
18 **if** $\bar{x}_c = \emptyset$ **then**
19 Let $\tau \in GP_c$ `// non-deterministic choice`
20 $O := O + \langle$`DELETE` τ `WHERE` $GP_c\rangle$
21 **return** O;

Since Algorithm 1 is data-independent, its exponential worst-case complexity (due to the powerset SGP_c computed on Line 11) is not necessarily an important limitation in practice, as it will be demonstrated in Section 7.

Algorithm 2 is a polynomial approximation of Algorithm 1 obtained as follows: instead of considering all possible subsets of triple patterns of SG_c (Line 12), we simply construct update queries that replace, in each triple pattern $\tau \in GP_c$, every critical term with a fresh blank node. As a result, there does not exist anymore any equality between images of join variables, literals or IRIs (while in Algorithm 1 all occurrences of each critical term were replaced by the same blank node). For instance, Algorithm 2 generates a sequence of three update queries, one for each triple, more general than O_2 from Example 3 of Section 4. For space reasons, the pseudocode of Algorithm 2 is reported in the companion appendix.

Theorem 3 states that Algorithm 2 is sound but leads to anonymizations that are more general than those produced by Algorithm 1.

Theorem 3. *The worst-case computational complexity of Algorithm 2 is polynomial in the size of $\mathcal{P}$. Let O and O' be the result of applying respectively Algorithm 1 and Algorithm 2 (with the same non deterministic choices) to a set $\mathcal{P}$ of privacy queries: for any RDF graph G, $(G, O, \mathcal{P})$* is safe *and $G \models O(G)$ and $O(G) \models O'(G)$.*

Theorem 4 establishes that the anonymization operations computed by Algorithm 1 preserve some information on $\mathsf{Count}(P)$ for privacy queries P with no boolean connected component. It is not necessarily the case for Algorithm 2.

Theorem 4. *Let $O =$ `find-safe-ops`$(\{P\})$ be the output of Algorithm 1 applied to a privacy query P with no boolean connected component. For every RDF graph G, $O(G)$ satisfies the the condition $\mathsf{Ans}(\mathsf{Count}(P), O(G)) \geq \mathsf{Ans}(\mathsf{Count}(P), G)$.*

6 Safe anonymization robust to `:sameAs` links

One of the fundamental assets of the LOD is the possibility to assert that two resources are the same by stating `owl:sameAs` triples (shortened to `:sameAs` later), also known as *entity linking*. We do not consider `:sameAs` between properties and we interpret `:sameAs` triples (called `:sameAs` links) as equality between constants (including blank nodes) that are in subject or object position. With this interpretation, `:sameAs` links can also be inferred by a logical reasoning on additional knowledge known on some properties (e.g. that a property is functional). In this section, we study the impact of both explicit and inferred `:sameAs` links on safety.

We extend Definition 5 to the semantics of query answering in presence of a set sameAs of `:sameAs` links. Let closure(sameAs) be the transitive, reflexive and symmetric closure of sameAs. This set can be computed in polynomial time. We write $G[b_0 \leftarrow b'_0, \ldots, b_k \leftarrow b'_k]$ for denoting the graph obtained from G by replacing each occurrence of b_i by b'_i for every $i \in [1..k]$.

Definition 11 (Answer of a query modulo sameAs). *Let Q be a conjunctive query, G an RDF graph and* sameAs *a set of `:sameAs` links. A tuple $\bar{a}$ is an answer to Q over G modulo* sameAs *iff there exists $(b_0, $ `:sameAs`$, b'_0)$, …, $(b_k, $ `:sameAs`$, b'_k)$ in* closure(sameAs) *s.t. $\bar{a} \in \mathsf{Ans}(Q, G[b_0 \leftarrow b'_0, \ldots, b_k \leftarrow b'_k])$. We note $\mathsf{Ans}_{\mathsf{sameAs}}(Q, G)$ the answer set of Q over G modulo* sameAs.

Hence, we extend Definition 8 to handle a set sameAs of `:sameAs` links.

Definition 12 (Safety modulo sameAs). *An anonymization instance $(G, O, \mathcal{P})$ is* safe modulo sameAs *iff for every RDF graph G', for every $P \in \mathcal{P}$ and for any tuple of constants $\bar{c}$, if $\bar{c} \in \mathsf{Ans}_{\mathsf{sameAs}}(P, O(G) \cup G')$ then $\bar{c} \in \mathsf{Ans}_{\mathsf{sameAs}}(P, G')$.*

O is safe modulo sameAs *for $\mathcal{P}$ if $(G, O, \mathcal{P})$ is safe modulo* sameAs *for every RDF graph G and for every set* sameAs *of `:sameAs` links.*

We first study how to build anonymization operations that are robust to explicit `:sameAs` links. Then, we focus on handling the case of inferred `:sameAs` links through knowledge.

Theorem 5 establishes that Algorithm 1 (and thus Algorithm 2) computes safe anonymizations even in presence of a set sameAs of explicit `:sameAs` links.

Theorem 5. *Let O be the result of applying Algorithm 1 to a set $\mathcal{P}$ of privacy queries: for any set* sameAs *of explicit* `:sameAs` *links, O* is safe modulo sameAs *for $\mathcal{P}$.*

We address two cases in which knowledge on properties may infer equalities. The first case occurs in the ontology axiomatization of the OWL language [6] when some of the properties are functional or inverse functional, as in Definition 13, where we model equalities by `:sameAs` links.

Definition 13. *A property p is* functional *iff for every $?x$, $?y_1$, $?y_2$:*
$(?x, p, ?y_1) \wedge (?x, p, ?y_2) \Rightarrow (?y_1, \texttt{:sameAs}, ?y_2)$.
A property p is inverse functional *iff for every $?x$, $?y_1$, $?y_2$:*
$(?y_1, p, ?x) \wedge (?y_2, p, ?x) \Rightarrow (?y_1, \texttt{:sameAs}, ?y_2)$.

For example, declaring that property `:bossOf` as inverse functional expresses the constraint that every person has only one boss. As shown in Example 4, exploiting this knowledge may lead to re-identifying blank nodes that have been produced by the previous anonymization algorithms.

Example 4. Let P be the following privacy query written in SPARQL syntax:

```
SELECT ?x  WHERE { ?x :seenBy ?y. ?x :bossOf ?z. }
```

Let G, $O(G)$ and G' be the following RDF graphs where O is an update operation returned by Algorithm 1:

```
G     = {:bob :seenBy :mary. :bob :bossOf _:b1. _:b1 :bossOf :ann.}
O(G)  = {_:b  :seenBy :mary. _:b  :bossOf _:b1. _:b1 :bossOf :ann.}
G'    = {:bob :bossOf :jim.  :jim :bossOf :ann.}
```

From $O(G) \cup G'$ and the inverse functionality of `:bossOf`, it can be inferred first (`:jim`, `:sameAs`, `_:b1`) and second (`:bob`, `:sameAs`, `_:b`). Consequently, `_:b` is re-identified as `:bob`, which is returned as answer of P over $O(G) \cup G'$ modulo sameAs, and the anonymization operation O is not safe.

One solution is to add a privacy query for each functional property p and for each inverse functional property q, respectively `SELECT ?x WHERE {?x p ?y.}` and `SELECT ?x WHERE {?y q ?x.}`. By doing so, the update queries returned by our algorithms will replace each constant in subject position of a functional property by a fresh blank node, and each constant in an object position of an inverse functional property by a fresh blank node. In the previous example, the constant `:ann` in (`_:b1`, `:bossOf`, `:ann`) would be replaced by a fresh blank node.

The second case that we consider may lead to infer equalities (modeled as `:sameAs` links) when a property is completely known, i.e., when its closure is available in an external RDF graph. For instance, suppose that the closure of the property `:seenBy` is stored in an extension of the external RDF graph G' containing the following triples:

[6] See OWL 2 RDF-Based Semantics, notably section 5.13. https://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/#Semantic_Conditions

```
:bob  :seenBy :mary.  :alice :seenBy :mary.
:john :seenBy :ann.   :tim   :seenBy :ann.
```

Knowing that G' is the complete extension of the `seenBy` predicate allows to infer (_:b, :sameAs, :bob) and thus to re-identify the blank node _:b.

One solution is to add a privacy query `SELECT ?x ?y WHERE { ?x p ?y }` for each property p for which we suspect that a closure could occur in the LOD. Then, the update queries returned by our algorithms will replace each constant in the subject or object position of such a property by a fresh blank node. For instance, in the Example 4, the constant :mary in (_:b, :seenBy, :mary) would be replaced by a fresh blank node.

7 Experimental evaluation

We have evaluated the runtime performance of the anonymization process produced by Algorithm 1 and the resulting loss of precision on three real RDF graphs for which we have designed a *reference privacy query* as a union of privacy conjunctive queries. Table 1 provides the indicators characterizing each RDF graph used in the experiments: $\#Triples$ (respectively $\#IRIs$ and $\#Blanks$) denotes the number of triples (respectively IRIs and blank nodes) in the graph, and $\#PrivQuery$ denotes the size of the reference privacy query (i.e., the sum of the triple patterns in each conjunctive privacy query). The reference privacy queries are reported in the companion appendix.

Table 1: RDF graphs and privacy queries used in our experiments.

RDF graph	#Triples	#IRIs	#Blanks	#PrivQuery
TCL	6,443,256	13,672,913	1,237,805	19
Synthetic transportation data				
Drugbank[7]	517,023	1,218,501	0	6
Real-world data about approved drugs				
(Swedish) Heritage[8]	4,970,464	12,421,192	0	6
Real world Europeana Swedish heritage data				

The experiments have been performed using Python 2.7, using RDF graphs stored on a Virtuoso version 07.20.3230 on a Linux Ubuntu 18.04.1 server, a 16GB RAM virtual machine with 2 VPCUs running in OpenStack. The source code of our prototype is openly available on GitHub and is provided along with the companion appendix.

7.1 Runtime performances

The average runtime of Algorithm 1 has been measured over 10 executions for each graph, using the reference privacy queries as input. Table 2 reports the

[7] http://wifo5-03.informatik.uni-mannheim.de/drugbank/
[8] General Europeana portal: https://pro.europeana.eu/page/linked-open-data

results: *T-Algo1* denotes the time in seconds for computing the sequence of anonymization operations by Algorithm 1 (which depends only of the reference privacy query) whereas *T-Anonym* denotes the time in seconds required for applying them on the different RDF graphs to compute their anonymized version, and *#UpdateQueries* is the number of update queries returned by Algorithm 1.

Table 2: Running time of anonymization process.

RDF graph	T-Algo1	#UpdateQueries	T-Anonym
TCL	0.207	16	3.5
Drugbank	0.012	6	1.7
Swedish Heritage	0.013	14	53.6

The anonymization time is reasonable in all cases. The cost for anonymizing the Swedish Heritage graph is due to a few update queries with graph patterns having many occurrences in the graph.

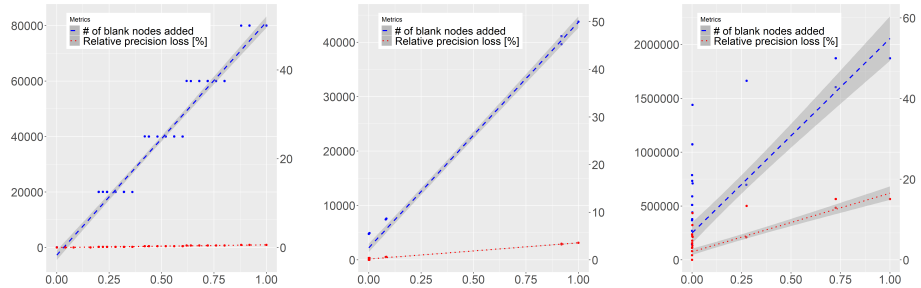
7.2 Evaluation of the precision loss

We have evaluated the *precision loss* and how it depends on the *privacy query specificity* defined relatively to the reference privacy query. We distinguish the *absolute precision loss* which is the *the number of blank nodes* introduced by the anonymization process, from the *relative precision loss* which the ratio of it with the total number of IRIs in the input graph.

From each reference privacy query P, we create more specific privacy queries by applying a set of random *mutations* that replace a variable by a constant in one of the privacy conjunctive queries. The specificity of a privacy query P' obtained by this mutation process is defined by $\mathsf{specif}(P') = |\,\mathsf{Ans}(P', G)|\,/\,|\,\mathsf{Ans}(P, G)|$.

By construction, $\mathsf{specif}(P')$ is a normalized value between 0 and 1, as any mutated privacy query P' is more specific than the reference privacy query P.

Results displayed on Figures 1a to 1c show that the precision loss grows linearly with the policy specificity: the less precise the privacy policy is in its selection of data, the more blank nodes will be inserted in the graph.



(a) TCL (b) Drugbank (c) Swedish Heritage

Fig. 1: Loss of precision depending on privacy query specificity for each graph.

We can also observe that precision is very dependent on the input: if the privacy policy only cover a specific part of the whole data (e.g. only the subscriptions in the data of whole transportation network) then its impact is quite small: for the TCL graph (Figure 1a), this precision value only drops marginally (99.9% to 99.4%). The trend is similar for other graphs: precision drops when the privacy policy gets more general. It drops to 85% in the case of the Swedish Heritage graph, and 96% for the Drugbank graph. This confirms that in general, using plausible privacy policy semantics, the number of IRIs lost in the anonymization process is not huge.

However, Figure 1c for the Swedish Heritage graph have a quite large spread on the $x = 0$ line. Indeed, the privacy policy forbids the disclosure of very general pieces of information such as the description of objects in the graph. Thus, this leads to many replacements by blank nodes in such a situation.

8 Conclusion

We have tackled the safety problem for Linked Open Data by providing a data-independent version and grounding it in a set of privacy queries (expressed in SPARQL). Our algorithms let seamlessly construct sequences of anonymization operations in order to prevent privacy leakages due to external RDF graphs along with explicit or inferred knowledge (under the form of sameAs links). We have proved the soundness of our anonymization algorithms and shown their runtime complexity. We have conducted experiments showing the quality of our anonymization and the performance of its operations.

Our approach can be seamlessly combined with existing privacy-preserving approaches. Once the RDF graph is transformed according to the operations generated by our algorithms, one could apply any other method to the obtained RDF graph. In particular, it could be verified whether the resulting anonymized RDF graph verifies some desired k-anonymity property. The adaptation of k-anonymity approaches for a more fine-tuned generalization of literals, see for instance [19], is planned as future work. Our approach can be also combined with ontology-based query rewriting for first-order rewritable ontological languages such as RDFS [2], DL-Lite [3] or EL fragments [12], by providing as input to Algorithm 1 the rewritings of the privacy queries.

We envision several other directions of future work. The first is to study the potential risk for re-identification of delegating the generation of fresh blank nodes to a standard SPARQL engine. Next, since the conditions provided in Theorem 1 for guaranteeing that a anonymization instance is safe are sufficient but not necessary, it would be beneficial to explore both sufficient and necessary conditions. We also plan to extend our safety model to handle additional knowledge, like for instance that some properties are equivalent. Finally, we plan to study whether considering the data-dependent version of the safety problem could lead to more specific anonymization operations while still guaranteeing safety.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Buron, M., Goasdoué, F., Manolescu, I., Mugnier, M.L.: Reformulation-based query answering for RDF graphs with RDF ontologies. In: ESWC, to appear (2019)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. Autom. Reasoning **39**(3), 385–429 (2007)
4. Delanaux, R., Bonifati, A., Rousset, M., Thion, R.: Query-based linked data anonymization. In: ISWC (1). Lecture Notes in Computer Science, vol. 11136, pp. 530–546. Springer (2018)
5. Deutsch, A., Papakonstantinou, Y.: Privacy in database publishing. In: ICDT. pp. 230–245 (2005)
6. Dwork, C.: Differential privacy. In: ICALP (2). LNCS, vol. 4052, pp. 1–12. Springer (2006)
7. Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: A survey of recent developments. ACM Comput. Surv. **42**(4), 14:1–14:53 (2010)
8. Goasdoué, F., Manolescu, I., Roatis, A.: Efficient query answering against dynamic RDF databases. In: EDBT. pp. 299–310 (2013)
9. Grau, B.C., Horrocks, I.: Privacy-preserving query answering in logic-based information systems. In: ECAI. pp. 40–44 (2008)
10. Grau, B.C., Kostylev, E.V.: Logical foundations of privacy-preserving publishing of linked data. In: AAAI. pp. 943–949. AAAI Press (2016)
11. Gutiérrez, C., Hurtado, C.A., Mendelzon, A.O.: Foundations of semantic web databases. In: PODS. pp. 95–106. ACM (2004)
12. Hansen, P., Lutz, C., Seylan, I., Wolter, F.: Efficient query rewriting in the description logic EL and beyond. In: IJCAI. pp. 3034–3040. AAAI Press (2015)
13. Heitmann, B., Hermsen, F., Decker, S.: k–RDF-neighbourhood anonymity: Combining structural and attribute-based anonymisation for linked data. In: PrivOn@ISWC. vol. 1951. CEUR-WS.org (2017)
14. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: ICDE. pp. 106–115. IEEE Computer Society (2007)
15. Machanavajjhala, A., He, X., Hay, M.: Differential privacy in the wild: A tutorial on current practices & open challenges. PVLDB **9**(13), 1611–1614 (2016)
16. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: *L*-diversity: Privacy beyond *k*-anonymity. TKDD **1**(1), 3 (2007)
17. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In: ISWC. pp. 376–394 (2018)
18. Miklau, G., Suciu, D.: A formal analysis of information disclosure in data exchange. Journal of Computer and System Sciences **73**(3), 507–534 (2007)
19. Radulovic, F., García-Castro, R., Gómez-Pérez, A.: Towards the anonymisation of RDF data. In: SEKE. pp. 646–651. KSI Research Inc. (2015)
20. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **10**(5), 557–570 (2002)
21. Vatsalan, D., Sehili, Z., Christen, P., Rahm, E.: Privacy-preserving record linkage for big data: Current approaches and research challenges. In: Handbook of Big Data Technologies, pp. 851–895. Springer (2017)