Semantic Web 0 (2020) 1–0
IOS Press

Differential Privacy and SPARQL

1

Carlos Buil-Aranda [a], Jorge Lobo [b] and Federico Olmedo [c]

[a] *Departamento de Informática, Universidad Técnica Federico Santa María and IMFD Chile Avda España 1680, Valparaíso Chile*
E-mail: cbuil@inf.utfsm.cl
[b] *Universidad Pompeu Fabra, ICREA, Roc Boronat 148, Barcelona, Spain*
E-mail: jorge.lobo@icrea.cat
[c] *Departamento de Ciencias de la Computación, Universidad de Chile and IMFD Chile, Chile, Beaucheff, Santiago Chile*
E-mail:

Abstract. Differential privacy is a framework that provides formal tools to develop algorithms to access databases and answer numerical and statistical queries with quantifiable accuracy and privacy guarantees. The notions of differential privacy are defined independent of the data model and the query language. Most results have been on aggregation queries such as counting or finding maximum or average values, and on grouping queries over aggregations such as the creation of histograms. The data model has been typically the relational model and the query language SQL. However, good realizations of deferential privacy for queries that required joins had been limited. This has imposed severe restrictions on applying differential privacy in RDF knowledge graphs and SPARQL. By the simple nature of RDF data, most interesting queries accessing RDF graphs will require intensive use of joins. Recently though, new techniques have been developed that can be applied to many types of joins in SQL with reasonable results. This opened the question of whether these new definitions can be transferred to RDF and SPARQL. In this paper we provide a positive answer to this question by presenting an algorithm that can answer count queries over a large class of SPARQL queries that guarantees differential privacy, if the RDF graph is accompanied with some natural semantic information about its structure. We have implemented our algorithm and conducted several experiments, showing the feasibility of our approach for large databases. Our aim has been to present an approach that can be used as a stepping stone towards extensions and other realizations of differential privacy for SPARQL and RDF.

Keywords: Differential Privacy, SPARQL

1. Introduction

As many social norms, privacy, or the right to privacy, is an evolving term that is invoked in many contexts as eloquently described by Louis Menand in [1]: "Privacy is associated with liberty, but it is also associated with privilege (private roads and private sales), with confidentiality (private conversations), with nonconformity and dissent, with shame and embarrassment, with the deviant and the taboo (...), and with subterfuge and concealment".

In order to get some formal underpinning of privacy in the context of electronic data collection and publishing, Li et al [2] have looked at recent privacy breaches, studied their general characteristics, and concluded, perhaps not surprising, that an electronic privacy breaches always ended with giving an attacker the ability to identify using public data whether an in-

[1]Carlos Buil was supported by Fondecyt Iniciacion 11170714 and by ANID - Millennium Science Initiative Program - Code ICN17_002. Jorge Lobo was partially supported by the Spanish Ministry of Economy and Competitiveness under Grant Numbers: TIN-2016-81032-P, MDM-2015-052, and the U.S. Army Research Office under Agreement Number W911NF1910432. Federico Olmerdo was also supported by ANID - Millennium Science Initiative Program - Code ICN17_002

dividual is member of a set or class that had been intended to be anonymous (e.g., the class of individuals with high cholesterol). Hence, they define preservation of privacy as *avoiding privacy breaches* in the sense of not disclosing set memberships of individuals.

For the public good, such the advance of public health, or the fair distribution of government resources, data is frequently made public. There are also situations in which governmental and commercial organizations collect and analyze data to improve or provide new services. But even in such cases, society expects certain level of privacy on the way these organizations use the data. Publishing data with perfect privacy means that no assumption can be made about the prior knowledge an attacker may have about the supposedly anonymous set. Under this assumption, there would be little utility in published data if prefect privacy is expected [2, 3]. Therefore, the research community has looked at weaker definitions of "acceptable" privacy. Useful concepts like k-anonymity [4], l-diversity [5] and t-closeness were developed but they were shown to have weak privacy guarantees.

In spite of its limitations, a privacy notion that has gained a lot of acceptance because of its formal properties is *Differential Privacy*. We will present precise definitions later in the paper, but informally, differential privacy tries to hide the identity of individuals that are members of a particular class, while still providing some guarantees about the utility of the published data about the class. The basic principle is simple. Given a universe $\mathbb{D}$ of all possible data sets and a query $f : \mathbb{D} \to \mathcal{R}$ that can be applied to a dataset D in $\mathbb{D}$ and results in a value of an abstract domain $\mathcal{R}$, the result $f(D)$ is deferentially private if it is indistinguishable from the query applied to *similar* datasets, D'. Instances of differential privacy use randomized algorithms to answer queries, indistinguishability is guaranteed with a specific minimal probability, and the utility of the answer is assessed by how diverse answers among similar datasets must be in order to maintain the differential privacy required. Smaller the diversity required, the better the utility. This is called in the literature *sensitivity* of the query. Calculating sensitivity is not trivial and approximations are used.

The notions of differential privacy are defined independently from the data model and query language, but most results have been on aggregation queries and grouping over aggregations in data sets in the context of relational databases and the query language SQL. Aggregations are queries such as counting, finding maximum, minimum or average values of members of a subset of the data that has certain property. Grouping is the creation of histograms based on aggregations. However, until recently, in order to get reasonable approximations of sensitivity, good realizations of deferential privacy for queries that required joins were limited [6]. This put severe limitations on applying differential privacy in RDF repositories and SPARQL. By the simple nature that data in RDF repositories is stored in binary relations, most interesting queries will require operations equivalent to joins. Nonetheless, in 2018, Johnson et al [7] introduced a new approximation of sensitivity that can be applied to many types of SQL joins with very reasonable results. This opened up the question of whether this new definition can be transferred to RDF and SPARQL. In this paper we provide a positive answer to this question by presenting an algorithm that can answer count queries over a very large class of SPARQL queries that guarantees differential privacy. This result has been possible by introducing the notion of a *Differential Privacy Schema* that allows us to redefine Johnson et al's sensitivity approximation of SQL queries in the appropriate terms for answering SPARQL queries. A Differential Privacy Schema groups sets of RDF tuples into sub-graphs that can be then used as single units for privacy protection. Examples show that this type of schema naturally arises from the semantics of the data stored in the tuples, and it should not be difficult for an administrator to define. We demonstrate the applicability of our proposed sensitivity approximation by implementing a Differential Privacy query engine that uses the approximation to answer counting and grouping SPARQL queries, and evaluate the implementation running simulations using the Wikidata knowledge base. We also run simulations using the Watdiv evaluation framework to find classes of queries that can be answered with reasonable utility in smaller datasets.

The rest of the paper is organized as follows: in Section 2 we introduce the readers with the fundamental concepts of Differential Privacy. We present in Section 3 the core concepts of SPARQL used within the paper. It is in this section where we introduce the concept of Differential Privacy Schema. In Section 4 we prove the correctness of our proposed approximation to sensitivity and in Section 5 we evaluate the effectiveness of our proposed approximation in an implementation that we apply to both synthetic and real world datasets and queries. We present related work in Section 6, and we conclude the paper in Section 7.

2. Background

We now describe the framework of differential privacy, the problem that arises when applying differential privacy to SQL queries with general joins and how it has been addressed.

2.1. Definition

Intuitively, a randomized algorithm is differentially private if it behaves similarly on similar input datasets. To formalize this intuition, the framework of differential privacy relies on a notion of *distance* between datasets. We model datasets as a *multiset of tuples* and we say that two datasets are *k-far apart* if one can be obtained from the other by changing the value of k tuples. Formally, this corresponds to the notion of *bounded* differential privacy [8], where all datasets separated by a finite distance share the same number of tuples. In the reminder, we let $\mathbb{D}$ be the set of all possible datasets, and use $d(D, D') = k$ to denote that $D, D' \in \mathbb{D}$ are k-far apart. In particular, two datasets $D, D' \in \mathbb{D}$ that are 1-far apart are called *neighbours*, written $D \sim D'$.

Definition 1. *Let $\epsilon, \delta \geqslant 0$. A randomized algorithm $\mathcal{A}$ is* (ϵ, δ)-differentially private *if for every pair of neighbour datasets $D, D' \in \mathbb{D}$ and every set $S \subseteq \mathsf{range}(\mathcal{A})$,*

$$\mathsf{Pr}[\mathcal{A}(D) \in S] \;\leqslant\; e^{\epsilon}\,\mathsf{Pr}[\mathcal{A}(D') \in S] + \delta\,.$$

This inequality establishes a quantitative closeness condition between $\mathsf{Pr}[\mathcal{A}(D) \in S]$ and $\mathsf{Pr}[\mathcal{A}(D') \in S]$, the probabilities that on inputs D and D', the outcome of $\mathcal{A}$ lays within S. The smaller the ϵ and δ, the closer these two probabilities are, and therefore, the less likely that an adversary can tell $\mathcal{A}(D)$ and $\mathcal{A}(D')$ apart. In other words, parameters ϵ and δ quantify the privacy guarantees of the randomized algorithm.

Multi-table datasets Our notions of dataset and distance between datasets can be extended to collections of datasets as follows: A dataset formed by multiple sets $T_1, \ldots, T_n$ will contain (tagged) data points belonging to $T_1, \ldots, T_n$ and the distance between two such datasets D, D' will reduce to $d(D, D') = \sum_{i=1}^{n} d(\pi_i(D), \pi_i(D'))$, $\pi_i(D)$ representing the subset of data points from D belonging to T_i. In the context of relational databases, the T_is correspond to relational tables.

2.2. Realization via Global Sensibility

Establishing differential privacy for numeric queries of limited sensitivity is relatively simple. The Laplacian mechanisms [9] says that we can obtain a differentially private version of query $f\colon \mathbb{D} \to \mathbb{R}$ by simply perturbing its output: On input D, we return $f(D)$ plus some noise sampled from a Laplacian distribution. The noise must be calibrated according to the *global sensitivity* GS_f of f, which measures its maximum variation upon neighbour datasets; formally, $\mathsf{GS}_f = \max_{D,D' | D \sim D'} |f(D) - f(D')|$.

Theorem 1. *Given a numeric query $f\colon \mathbb{D} \to \mathbb{R}$ of global sensitivity GS_f, the randomized algorithm*

$$\mathcal{A}(D) \;=\; f(D) + \mathsf{Lap}\Big(\tfrac{\mathsf{GS}_f}{\epsilon}\Big)$$

is an $(\epsilon, 0)$-differentially private version of f.

Here $\mathsf{Lap}(\lambda)$ represents sample from the *Laplacian distribution* with parameter λ, a symmetric distribution with probability density function $pdf(x) = \frac{1}{2\lambda} e^{-|x|/\lambda}$, mean 0 and variance $2\lambda^2$. Parameter λ measures how concentrated the mass of the distribution is around its mean 0: The smaller the λ, the less noise we add to the true query result and therefore, the more faithful the mechanism becomes. In the realm of differential privacy, this "faithfulness" property is referred to as the mechanism *utility*. An important point here is that utility and privacy are always conflicting requirements: adding more noise results in more private and—at the same time—less useful mechanisms.

In practice, when implementing the Laplacian mechanism we approximate the global sensibility of queries by exploiting their structures: Numeric queries are typically constructed by first transforming the original dataset using some standard transformers and by returning as final result some aggregation on the so obtained dataset. For example, we join two tables, filter the result (dataset transformations) and return the count (aggregation) of the obtained table. The global sensitivity of such a query can be estimated from the so-called stability properties of the involved transformers. Intuitively, a stable transformer can increase the distance between nearby datasets at most by a multiplicative factor. Formally, we call a dataset transformer $T\colon \mathbb{D} \to \mathbb{D}$ *α-globally-stable* if $d(T(D), T(D')) \leqslant \alpha\, d(D, D')$ for every $D, D' \in \mathbb{D}$. Transformers with bounded global stability yield bounded global sensitivities: $\mathsf{GS}_{f \circ T} \leqslant \alpha\, \mathsf{GS}_f$ whenever T is α-globally-stable.

Conversely, the use of transformers with unbounded stability might result in queries of unbounded sensitivity. A prominent example of a transformer exhibiting this problem is join. Assume we join two tables, say t_1 and t_2, by matching a pair of their attributes. A modification in a mere tuple from t_1 may result in the addition and/or deletion of an unpredictable number of tuples in the result of the join, leaving elementary queries such as counting the number of tuples in a join already out of the scope of the Laplacian mechanism. The applicability of differential privacy approaches based on query *global* sensitivity is thus rather limited.

2.3. Realization via Local Sensibility

To handle queries that involve transformers of unbounded stability, such as joins, we require the use of more advanced techniques. The Laplacian mechanism calibrates noise according to the query, overlooking the fact that queries are done on concrete datasets, hence the employed noise could be potentially customized for each dataset. Nissim *et al.* show how to exploit this idea of instance-based noise [10, 11]. Their approach relies on the notion of local sensitivity.

Definition 2. *The* local sensitivity $\mathsf{LS}_f^{(k)}(D)$ *at distance* $k \in \mathbb{N}_0$ *of a numeric query* $f\colon \mathbb{D} \to \mathbb{R}$ *on dataset* $D \in \mathbb{D}$ *is defined for* $k = 1$ *as*

$$\mathsf{LS}_f^{(1)}(D) \;=\; \max_{D' \,\mid\, d(D,D')=1} \big|f(D) - f(D')\big| \,.$$

And for $k \geqslant 2$ *as*

$$\mathsf{LS}_f^{(k)}(D) \;=\; \max_{D' \,\mid\, d(D,D')\leqslant k} \mathsf{LS}_f^{(1)}(D') \,.$$

When $k = 1$ *we simply write* LS_f *for* $\mathsf{LS}_f^{(1)}$ *and refer to it as the* local sensitivity *of* f.

For answering a query f on dataset D, we cannot simply use noise calibrated according to $\mathsf{LS}_f(D)$ because the noise level itself may reveal information about D [7]. Instead, we should use an approximation of LS_f that is insensitive to small variations of its input dataset. This is captured by the notion of *smooth* upper bound.

Definition 3. *A function* $\mathcal{U}\colon \mathbb{D} \to \mathbb{R}_{\geqslant 0}$ *is called a* β-smooth upper bound *of the local sensitivity* $\mathsf{LS}_f\colon \mathbb{D} \to \mathbb{R}_{\geqslant 0}$ *of query* $f\colon \mathbb{D} \to \mathbb{R}$ *if it satisfies the following requirements:*

1. $\mathcal{U}(D) \geqslant \mathsf{LS}_f(D)$ *for all dataset D, and*
2. $\mathcal{U}(D) \leqslant e^{\beta}\mathcal{U}(D')$ *for all neighbour datasets D and* D'.

We can readily achieve differential privacy by adding noise calibrated according to a smooth upper bound of the query local sensitivity [11, Corollary 2.4].

Theorem 2. *Let* $f\colon \mathbb{D} \to \mathbb{R}$ *be a numeric query and let* $\mathcal{U}\colon \mathbb{D} \to \mathbb{R}_{\geqslant 0}$ *be a* β*-smooth upper bound of its local sensitivity* LS_f. *Moreover, let* $\delta \in (0,1)$ *and let* $\beta \leqslant \frac{\epsilon}{2\ln(2/\delta)}$. *Then, the randomized algorithm*

$$\mathcal{A}(D) \;=\; f(D) + \mathsf{Lap}\Big(\tfrac{2\,\mathcal{U}(D)}{\epsilon}\Big)$$

is an (ϵ, δ)*-differentially private version of* f.

The benefits of this mechanism are twofold. On the one hand, it allows handling queries that fail to have a bounded *global* sensitivity, but *do* have a bounded *local* sensitivity. These include *e.g.* the query we considered earlier, consisting of the count of the join between two tables. On the other hand, it does not require computing the local sensitivity of the queries themself, but only a smooth upper bound thereof. This is key for its practical adoption since calculating the local sensitivity of queries is computationally prohibitive: As observed by Johnson *et al.* [7], "it requires running the query on every possible neighbour of the original dataset".

To apply the mechanism from Theorem 2, we must provide a smooth upper bound for the local sensitivity of queries. We can construct the smooth upper bound using approximations for the local sensitivity at fixed distances.

Lemma 1. *Let* $f\colon \mathbb{D} \to \mathbb{R}$ *be a numeric query and assume that* $\mathcal{U}^{(k)}$ *is a pointwise upper bound of the local sensitivity* $\mathsf{LS}_f^{(k)}$ *of* f *at distance k, that is,*

$$\mathcal{U}^{(k)}(D) \;\geqslant\; \mathsf{LS}_f^{(k)}(D) \qquad \text{for all } D \in \mathbb{D} \,.$$

Then,

$$\mathcal{U}(D) \;=\; \max_{0\leqslant k\leqslant size(D)} e^{-\beta k}\,\mathcal{U}^{(k)}(D)$$

is a β*-smooth upper bound of the local sensitivity* $\mathsf{LS}_f(D)$ *of* f *on D.*

The goal of Section 4 is to apply the differential privacy mechanism from Theorem 2 to SPARQL count queries. We will study how to efficiently construct uppers bounds for the local sensitivity of queries at fixed distances. To do so, we will in turn leverage *local* stability properties of SPARQL dataset transformers.

3. SPARQL Preliminaries

In this Section we introduce briefly the SPARQL1.1 structural features we consider most important in the application of Differential Privacy. These features are those that will affect the query sensibility as presented in Section 2. These features are divided in

1. data-driven features and
2. query structural features.

We assume the reader is familiar with the RDF data model [12] and with SPARQL [13].

3.1. Data-driven features

In the relational database model, Differential Privacy protects the disclosure of the presence or absence of a single tuple in the database.

In our context, where a dataset is an RDF graph G, what we would like to protect is the disclosure of the presence or absence of a sub-graph of G that represents the contribution of a single individual or entity to G. We aim to protect the identity of the individual or the entity. Hence, we want to characterize G as a set $\{g_1, \ldots, g_n\}$ of sub-graphs such as each g_i represents the contribution of entity i to the dataset $G = \bigcup g_i$. Similarly to the case of relational databases, we define graphs at distance 1 as graphs that are represented by sets of sub-graphs of the same cardinality which differ in a single sub-graph. In general, there might be parts of G that are public and we don't care to protect. We will discuss later how we can use the information that some portions of the graph are public to improve the upper bounds of local sensitivity.

It will be through semantic information provided by the administrator of the RDF graph G, that we will be able to identify the sub-graphs that characterize G. This semantic information will be a finite set of basic graph patterns (BGPs) plus some additional constraints that we will later formally define as a Differential Privacy schema. There is only a subclass of BGPs that can be members of a Differential Privacy schema. To identify those BGPs, we need the concept of a join vertex from [14]. A *join vertex* in a BGP B is a variable that appears either as a subject or as an object multiple times in B. We call B a *star* BGP, if it is either a single triple pattern with no join vertex, or it is a BGP with (1) a single join vertex appearing once and only once in every triple pattern in B, and (2) no repeated predicates. We call the star *simple* if both the subject and the object in every triple pattern in the star are variables. In a simple start we call the *center* of the star the join vertex if there are multiple patterns in the star, or the subject of the triple pattern in case that the star is a singleton set.

Note that start BGPs are similar to Star queries [14], except that in a star query the center of the star must always appear as a subject. This is a constraint that we do not need to impose, but it is important for every triple patter in a star to share either its subject or its object with the other triple patterns. A graphical representation of a star BGP is a star with the join vertex in the center with all property arrows either coming into or going out from the vertex.

We say that two star BGPs are *disjoint* if they don't have predicates in common. Figures 3 and 4 show two disjoint star BGPs with central vertex $?v0$. Now we can formally say:

Definition 4. *A* Differential Privacy schema $\mathcal{P}$ *is a finite set of pair-wise disjoint simple stars.*

It is obvious that because stars in a differential privacy schema do not share predicates, the evaluation of any two BGPs over an RDF graph will use disjoint sets of RDF triples to find the mappings. Hence, given an RDF graph G, we will call the (RDF) *graph induced* by a star BGP B from G the set of RDF triples used to obtained the solution mappings $[\![B]\!]_G$ as defined by the standard SPARQL query semantics [13], and the *graph induced* by a Differential Privacy schema $\mathcal{P}$, the union of the graphs induced by each of the star BGPs in the schema.

We say that a graph G *complies* with a Differential Privacy schemes $\mathcal{P}$ iff G coincides with the graph induced by $\mathcal{P}$ from G.

Example 3.1. *Take the RDF graph in Figure 1. The graph complies with the schema*

$$\mathcal{P} = \{\{(?s1, type.?o1)\},\ \{(?s2, lives, ?o2), (?s2, enemy, ?o3)\},\ \{(?s4, hires, ?o4), (?s4, has, ?o5)\}\ \}$$

This graph is partitioned into the subgraphs

$g_1 = \{(Joker, type, Clown)\}$,
$g_2 = \{(Gotham, type, City)\}$,
$g_3 = \{(Joker, enemy, Batman), (Joker, lives, Gotham)\}$,
$g_4 = \{(Batman, enemy, Joker), (Batman, lives, Gotham)\}$,
$g_5 = \{(Gotham, hires, Clown), (Gotham, has, Police)\}$

Administrators must provide a Differential Privacy schema for their RDF graphs. Such a schema always exists for every graph since the union of all singleton BGPs of the form $\{(?X, p, ?Y)\}$ that can be defined using all predicates p that appear in G is a Differential

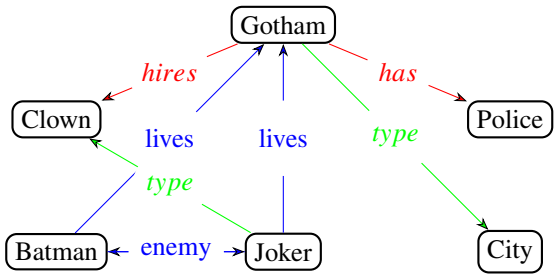

Figure 1. RDF graph with a three star schema

Privacy schema. This schema indicates that each RDF triple is the contribution of a different entity and needs privacy protection, and although this is possible, it is not usually the case. In many situations, an entity possesses a collection of properties that are collected in a set of RDF triples, all having a term in common, and hence, captured by a simple star BGP, that contains: 1) a triple pattern $(?X, p, ?Y)$, for each predicate in the set of RDF triples, and 2) its center is determined by the position the common term that appears in all the triples of the collection. Figure 1 shows an example. This is why *Star* triple patterns are common SPARQL queries. Administrators should then provide Differential Privacy schemes with the largest stars possible since, as we will see later, it will allow better approximations of local sensitivity.

There is a second characteristic typical of a schema compliant graph G that we can exploit: if $tp = (?X, p, ?Y)$ is a triple pattern in a simple star B of a schema centered in $?X$, the solution mappings $\mu \in [\![tp]\!]_G$, are such that the number of mappings with the same mapping for $?X$ can be bound to be smaller or equal to a constant κ. In many cases $\kappa = 1$. Take for example, employees identified by a company *id* as subjects of RDF triples with a predicate that identifies an employee salary, or age, or seniority status, all have $\kappa = 1$. An employee might have several phone numbers, but the administrator can constrain the graph to a maximum of, let's say, 5 numbers per employee and set $\kappa = 5$. We call κ the *sensitivity* of the triple pattern. The definition is similar if $?Y$ is the center. For star pattern S, the *sensitivity* of S, denoted $sen(S)$ is the multiplication of the sensitivities of the triple patterns in S.

Henceforth, we assume we have an RDF graph G that complies with a Differential Privacy schema $\mathcal{P}$, and in addition, we also know the sensitivity $sen(S)$ of all its triple patterns. With $\mathcal{P}$, we can get the set of sub-graphs $\{g_1, \ldots, g_n\}$ of G. A graph g belongs to the set iff there is a star pattern $S \in \mathcal{P}$ with center $?X$, a solution assignment $\mu \in [\![S]\!]_G$ and g is the graph induced by the SPARQL pattern S FILTER $?X = \mu(?X)$. In this case, we say that g *belongs* to S and denote the set of all sub-graphs, $\{g_1, \ldots, g_n\}$, by $\mathcal{P}(G)$. Note that for any distinct pair of sub-graphs, g_i and g_j, their intersection is empty.

3.2. SPARQL query structural features

For queries, we continue to follow the notations from [14] and develop differential privacy over the SPARQL fragment of basic graph patterns with filter expressions (CBGP). In this fragment, a query is denoted by a pair $\bar{B} = \langle B, F \rangle$, where B is a *BGP* and $F = \{f_1, \ldots, f_n\}$ is a finite set of filter expressions. $\bar{B}$ represents the SPARQL graph pattern $P = ((\ldots(B \texttt{ FILTER } f_1)\ldots)\texttt{FILTER } f_n)$, and its meaning, denote by $[\![\bar{B}]\!]_G$, is the multiset of solution mappings $[\![P]\!]_G$ as defined by the standard semantics of SPARQL queries [13]. Filter expressions will have no disjunctions, no conjunctions, and no equalities involving two variables since these equalities can be removed from the filters by variable renaming.[2] Note that negation of equalities are allowed. We also assume that in a graph G that complies with a Differential Privacy schema $\mathcal{P}$, all predicates appearing in triple patterns of a CBGP also appear in $\mathcal{P}$.

A user query is a CBGP $\bar{B}$ that will be always wrapped by one of the following two aggregation operations:

1. $\texttt{COUNT}(\bar{B})$ for which its semantics $[\![\texttt{COUNT}(\bar{B})]\!]_G$, is defined as the cardinality of the multiset $[\![\bar{B}]\!]_G$.
2. $\texttt{COUNT}_{?X_1 \ldots ?X_n}(\bar{B})$, where $?X_1, \ldots, ?X_n$ are variables that appear in $\bar{B}$, and for which its semantics $[\![\texttt{COUNT}_{?X_1 \ldots ?X_n}(\bar{B})]\!]_G$, is defined as grouping solution mappings from $[\![\bar{B}]\!]_G$ according to one or more variable expressions $?X_1 \ldots ?X_n$.

Example 3.2. *For the schema $\mathcal{P}$ of the RDF graph in Figure 1, let*
$B = \{(?s, lives, ?p), (?s, enemy, ?e), (?e, lives, ?p), (?s, type, ?c)\}$
$F = \{?c \neq Cat\}$)
Then $\langle B, F \rangle$ is a CBGP.

Let $var(E)$ be the set of variables that appear in the expression E, where E can be either a CBGP, a BGP or a Differential Privacy schema.

[2] Observe that F is essentially a conjunction of filters.

For a simple star S and a triple pattern $tp = (s, p, o)$ such as $(?x, p, ?y) \in S$, let $center_S(tp) = s$ if $?x$ is the center of S; otherwise $center_S(tp) = o$.

For any query $\bar{B} = \langle B, F \rangle$, we can naturally define from a Differential Privacy schema $\mathcal{P}$, a split $B_{\div\mathcal{P}}$ of B as follows: $B_{\div\mathcal{P}} = \{B_1, \ldots, B_n\}$ iff every $B_i \in B_{\div\mathcal{P}}$ is a maximal subset of B for which there exists a simple star $S \in \mathcal{P}$ such that $pred(B_i) \subseteq pred(S)$ and for any two triple patterns, $tp, tp' \in B_i$, $center_S(tp) = center_S(tp')$. Because the stars in $\mathcal{P}$ are disjoint and the B_i's are maximal, this split is unique. We will call S the *covering star* of B_i.

Example 3.3. *From our previous CBGP example, $B_{\div\mathcal{P}}$ has three members:*

$B_1 = \{(?s,\ lives,\ ?p), (?s,\ enemy,\ ?e)\}$
$B_2 = \{(?s,\ enemy,\ ?e), (?e,\ lives,\ ?p)\}$
$B_3 = \{(?s, type, ?c)\}$

The interest of $B_{\div\mathcal{P}}$ is that it lets us isolate the part of the graph necessary to answer each B_i. Let denote by G_{S_i} the graph induced by S_i

Lemma 2. *If S_i is the covering star of B_i then*

$$[\![B_i]\!]_G = [\![B_i]\!]_{G_{S_i}}$$

Then, following the terminology defined in [15] for joins between multisets of solution mappings, we can extend the lemma to B as follows:

Lemma 3.

$$[\![B]\!]_G = [\![B_1]\!]_{G_{S_1}} \bowtie [\![B_2]\!]_{G_{S_2}} \bowtie \cdots [\![B_n]\!]_{G_{S_n}}$$

Recall that $\bar{B}$ is such that $[\![B]\!]_G$ can be evaluated using only equi-joins. This implies that there is an ordering of the elements in $B_{\div\mathcal{P}}$ such as $[\![B_i]\!]_{G_{S_i}} \bowtie [\![B_{i+1}]\!]_{G_{S_{i+1}}}$ can also be done with equi-joins. In other words, $var(B_i) \cap var(B_{i+1}) \neq \emptyset$. We call this order a normal ordering of $B_{\div\mathcal{P}}$. For national convenience, we will assume that the indexing we use for $B_{\div\mathcal{P}}$ follows a normal order.

We are now in place to define Differential Privacy for SPARQL count and histogram queries.

4. Towards Differential Privacy for SPARQL

The goal of this section is to provide the basic notions to assure privacy over SPARQL counting queries. As we said before, a randomized algorithm is differentially private if it behaves similarly on similar input datasets. That notion is formalized by defining a *distance* between graphs. However, as we saw in the previous sections, the distance should be selected carefully, since different distance measures may give different privacy properties.

For the rest of the paper, we assume that there is a fixed Differential Privacy schema $\mathcal{P}$, and all RDF graphs are compliant with this schema. The distance between two graphs, $d(G_1, G_2)$ is defined when $|\mathcal{P}(G_1)| = |\mathcal{P}(G_2)|$, and it will be the cardinality of the difference, $\mathcal{P}(G_1) \setminus \mathcal{P}(G2)$, between G_1 and G_2. Because the number of sub-graphs is the same in $\mathcal{P}(G_1)$ and $\mathcal{P}(G2)$, distance is commutative as expected.

Based on Lemma 1, we develop in this section, for a user query Q and an RDF graph G that complies with a Differential Privacy schema $\mathcal{P}$, an upper bound $\mathcal{U}_Q^{(k)}$ of the local sensitivity $\mathsf{LS}_Q^{(k)}$, to get a β-smooth upper bound $\mathcal{U}_Q(G)$ of the local sensitivity $\mathsf{LS}_Q(G)$ of Q on G. With that in mind, we turn to the concept *elastic sensitivity* of Johnson *et al.* [7] to define such upper bounds.

4.1. Elastic sensitivity

The main goal of Elastic Sensitivity is to provide a sensitivity measure without losing utility in excess. This is an open research question in the database and privacy communities and the authors of [7] introduced Elastic Sensitivity as a practical method to provide Differential Privacy within Uber data analysis using relational databases and SQL.

We will redefine their notions in order to apply them to RDF graphs and SPARQL. Thus, elastic sensitivity of a query Q at distance k from the true graph G is denoted by $\mathcal{S}^{(k)}(Q, G)$, and it will be defined in terms of another function that tries to capture the idea of local stability but for CBGP queries. We will call this function the *elastic stability* of SPARQL transformations defined by the BGP part of CBGPs. Given a BGP B, its elastic stability, denoted $\mathcal{S}^{(k)}_{RDF}(B, G)$, bounds the local sensitivity a distance k of $\texttt{COUNT}(B)$.

To define elastic stability we need to count the most popular result mapping (maximum frequency) for any variable $?x$ that appears in a CBGP $\bar{B}$ given a graph G. This can be calculated using the query

$$SELECT\ (COUNT(?v0)\ as ?c)\ WHERE\ \{?v0\ ?p\ ?o\}$$

$$ORDER\ BY\ ?v0.\ DESC(?c)\ LIMIT\ 1$$

We denote this value by $mp(?X, \bar{B}, G)$. we will also need to estimate the most popular result mapping from graphs at distance k from G, denoted by $mp_k(?X, \bar{B}, G)$. mp_k is defined inductively on the cardinality of $B_{\div\mathcal{P}} = \{B_1, \ldots, B_n\}$:
For $|B_{\div\mathcal{P}}| = 1$: $mp_k(?X, \bar{B}, G) = mp(?X, \bar{B}, G) + k * sen(S)$, where S is the covering star of B_1.

For $|B_{\div\mathcal{P}}| = n + 1$: Let $B_1 \in B_{\div\mathcal{P}}$, and $V = var(B_{\div\mathcal{P}} \setminus \{B_1\}) \cap var(B_1)$. Then,
$mp_k(?v, \bar{B}, G) =$
$$\max\left\{\begin{array}{l}\arg\max_{?u\in V} mp_k(?v, B_{\div\mathcal{P}} \setminus \{B_1\}, G) \times mp_k(?u, B_1, G) \\ \arg\max_{?u\in V} mp_k(?u, B_{\div\mathcal{P}} \setminus \{B_1\}, G) \times mp_k(?v, B_1, G)\end{array}\right\}$$
We extend the definitions to a set of variables V as follows:

$$mp(V, \bar{B}, G) = \arg\max_{?v\in V} mp(?v, \bar{B}, G)$$
$$mp_k(V, \bar{B}, G) = \arg\max_{?v\in V} mp_k(?v, \bar{B}, G)$$

For variables that don't appear in $\bar{B}$, $mp(?v, \bar{B}, G) = mp_k(?v, \bar{B}, G) = 0$. We will also define the elastic stability of a CBGP $\bar{B}$ by induction on the cardinality of $B_{\div\mathcal{P}} = \{B_1, \ldots, B_n\}$, and we will use the fact that the B_i's are indexed in a normal order.

For $|B_{\div\mathcal{P}} = 1|$: $\mathcal{S}^{(k)}_{RDF}(B, G) = sen(S)$, where S is the covering star of B_1.

For $|B_{\div\mathcal{P}}| = n + 1$: Let $B' = B_{\div\mathcal{P}} \setminus \{B_1\}$. We have two cases. If the covering star S of B_1 is not the covering star of any other $B_i \in B'$, $\mathcal{S}^{(k)}_{RDF}(B, G) =$
$$\max\left\{\begin{array}{l} mp_k(var(B_1) \cap var(B'), B_1, G) \times \mathcal{S}^{(k)}_{RDF}(B', G) \\ mp_k(var(B_1) \cap var(B'), B', G) \times \mathcal{S}^{(k)}_{RDF}(B_1, G)\end{array}\right\}$$
If the covering star S of B_1 is also the covering star of another $B_i \in B'$,
$$\begin{aligned}\mathcal{S}^{(k)}_{RDF}(B, G) = {} & mp_k(var(B_1) \cap var(B'), B_1, G) \times \mathcal{S}^{(k)}_{RDF}(B', G) + \\ & mp_k(var(B_1) \cap var(B'), B', G) \times \mathcal{S}^{(k)}_{RDF}(B_1, G) + \\ & \mathcal{S}^{(k)}_{RDF}(B_1, G) \times \mathcal{S}^{(k)}_{RDF}(B', G)\end{aligned}$$
Note that a BGP is a CBGP with F empty.

Lemma 4. *For any CBGP query $\bar{B} = \langle B, F\rangle$, and any graph G compliant with schema $\mathcal{P}$:*

$$\mathcal{S}^{(k)}_{RDF}(B, G) \geqslant \mathsf{LS}^{(k)}_{COUNT(B)}(G)$$

The main observation is that when G changes in one of its sub-graphs g_i, and that sub-graph belongs to the star pattern S, then the number of RDF tuples in G can change in the worst case by $sen(S)$. If these RDF triples contribute as result mappings of a join vertex, the number of new mappings can increase by as much as the most popular result mapping of the joining triple pattern. If this triple pattern is outside the scope of S. For example, if $B = \{?v0, p, ?u), (?u, p', ?v1)\}$, and the triple (s, p, o) is part of g_1 and o happens to be the most popular result mapping for $(?u, p', ?v1)$, then there will be at most $mp(?u, (?u, p', ?v1), G)$ new mappings in the result if $(?u, p', ?v1)$ doesn't belong to S. If it does, there can be more because the same new triple can participate twice in the same join. Formally,

Proof. The proof of this lemma follows the same strategy that the proof in [7, Lemma 2] and it is by induction on the structure of B.

- **Case** $|B_{\div\mathcal{P}}| = 1$ where S is the covering star of B, the sensibility is $sen(S) = \kappa$, a parameter given by the DBA.
- **Case** $|B_{\div\mathcal{P}}| = n + 1$: we have a covering star S_1 for partition B_1 and a set with n covering stars for partitions $B' = \{B_2, \ldots B_{n+1}\}$. We want to bound the number of RDF triples in G that can change in the worst case by $sen(S)$. First, let's assume S_1 is not the covering star of any B_i in B'. In this case either G_{S_1} or $G_{S'}$ from a covering star $S' \in \{S_2 \ldots S_n\}$ changes but not both, thus, either $\mathcal{S}^{(k)}_{RDF}(B_1, G) = 0$ or $\mathcal{S}^{(k)}_{RDF}(B_i, G) = 0$ where $B_i \in B'$ (we are modifying the triples affecting only one privacy partition schema):

  1. When $\mathcal{S}^{(k)}_{RDF}(B_1, G) = 0$, B', induction hypothesis, may contain $\mathcal{S}^{(k)}_{RDF}(B', G)$ changed result mappings, producing at most $mp_k(var(B') \cap var(B_1), B, G) \times \mathcal{S}^{(k)}_{RDF}(B', G)$ changed mappings in the joined SPARQL pattern.
  2. In the symmetric case, when $\mathcal{S}^{(k)}_{RDF}(B', G) = 0$, B_1 may contain $\mathcal{S}^{(k)}_{RDF}(B_1, G) = sen(S_1)$ changed rows, producing at most $mp_k(var(B_1) \cap var(B'), B, G) \times \mathcal{S}^{(k)}_{RDF}(B_1, G)$ changed rows in the joined SPARQL pattern.

  We chose the maximum between the two values when calculating $\mathcal{S}^{(k)}_{RDF}(B, G)$. On the other hand, if S_1 also covers another $B_i \in B'$, a change in B_1 can also imply changes in B'. The changes in B' can affect the join for RDF tuples in the original G. This, in the worst case, can cause $mp_k(var(B') \cap var(B_1), B, G) \times \mathcal{S}^{(k)}_{RDF}(B', G)$

changed mappings in the joined SPARQL pattern. In addition, the change in B_1, which may contain $\mathcal{S}^{(k)}_{RDF}(B_1, G) = sen(S_1)$ changed rows, can cause $mp_k(var(B_1) \cap var(B'), B, G) \times \mathcal{S}^{(k)}_{RDF}(B_1, G)$ changed rows in the joined SPARQL pattern from original tuples in G. We also need to consider that the change allows new joins between new rows in both B_1 and B', for a total of $\mathcal{S}^{(k)}_{RDF}(B_1, G) \times \mathcal{S}^{(k)}_{RDF}(B', G)$ changed mappings in the joined SPARQL pattern. The sum of these three values is what the definition of $\mathcal{S}^{(k)}_{RDF}(B, G)$ uses.

□

As in Johnson *et al.* [7], this lemma leads us to our main theorem.

Theorem 3. *For any user query Q, and any graph G compliant with schema $\mathcal{P}$:*

$$\mathcal{S}^{(k)}(Q, G) \geqslant \mathsf{LS}^{(k)}_{Q}(G)$$

Proof. We assume that Q are SPARQL *Count* and *Histogram* (*Count* with **GROUP_BY**) queries.

- Case *Count* queries: the result follows directly from Lemma 4 since the result of the counting query is given by the application of the sensitivity calculation for the *CBGPs* in the query.
- Case *Histogram* (with **GROUP_BY**) queries. In a **GROUP_BY** query each changed triple affects two result mappings in the query since one modified triples will generate a mapping that may fall into one group and losing that mapping another group. Thus $2\mathcal{S}^{(k)}(Q, G)$.

□

Our implementation of an (ϵ, δ)-differential privacy algorithm $\mathcal{A}$ from Theorem 2 uses $\mathcal{S}^{(k)}_{RDF}$ to implement $\mathcal{U}$ according to Lemma 1.

5. Evaluation

Having characterized formally how an algorithm can be implemented to enforce Differential Privacy on SPARQL queries based on privacy schemes, in this section, we present an empirical evaluation of how the algorithm would behave in real scenarios. Aluç *et al.* [14] have identified three main types of SPARQL queries, namely, *Star*, *Snowflake* and *Path* (or *Linear*) shaped queries. Analysis of real SPARQL query logs had confirmed that this classification covers most user queries in practice. Our evaluation presents the results based on this query classification. We devised our evaluation in two parts. First, we used data from the Wikidata project [16] using its query logs. We also taken 5 queries from the Wikidata examples found at its SPARQL endpoint[3] (the Wikidata examples, besides of being in the log files, are clear examples of the query types we describe). For the second part, we used the Watdiv evaluation framework where we have direct control on the generate RDF triples in relation to the query types. In Watdiv, we can also control the size of the graph which allows us to show the effect the graph size has on the query sensitivity For each query we calculate the Elastic Sensitivity and run simulations of two kinds. In one, we assume the URI to be private and hence subject to Differential Privacy. But since this URI will be public in many cases, we also run simulations ignoring the calculation of Differential Privacy for the URI. The evaluation shows that this simple observation has a significant impact in sensitivity.

Setup We did our analysis on a 2016 Macbook Pro with 8 GB of RAM memory. The implementation was done using Java 1.8 and the SymJava library for symbolic-numeric computation [17]. We used the `SecureRandom` Java class to generate the random numbers to calculate the Laplacian probability distribution since that class implements a well-tested random number generator[4], an essential component for ensuring the correctness our privacy guarantees algorithm. The code and all the queries used for this evaluation are available in GitHub [5].

5.1. Data & Queries

Wikidata For evaluating our approach we use real world data and queries from Wikidata [16]. Wikidata is a collaboratively edited knowledge base hosted by the Wikimedia Foundation. It is a common source of data for Wikimedia projects such as Wikipedia, and it has been made available to the general public under a public domain license. Wikidata stores 86,671,701 items (RDF resources), and 1,084,935,969 statements (triples[6]). For queries, we randomly se-

[3] https://query.wikidata.org

[4] https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html

[5] Repository https://github.com/cbuil/PrivacyTest1

[6] https://tools.wmflabs.org/wikidata-todo/stats.php

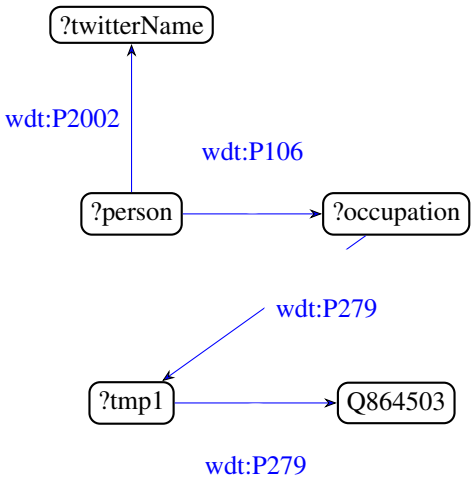

Figure 2. Snowflake query shape

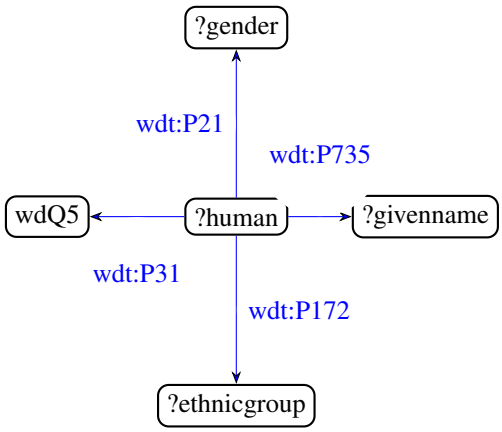

Figure 3. Star query shape

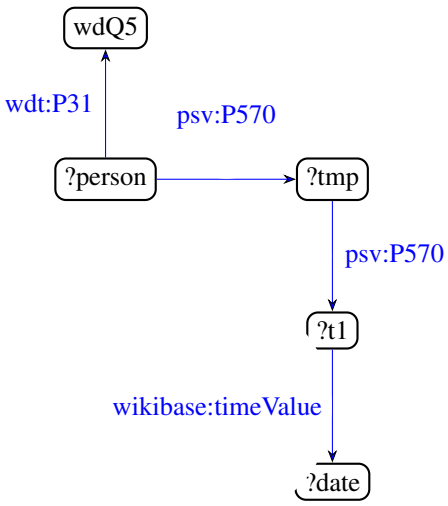

Figure 4. Linear query shape

lected 10 queries from the latest query log files available [7] (from 2018-02-26 to 2018-03-25) and 5 examples from the Wikidata endpoint (F1, F5, S1, S3 and L5 – see naming description below). In total we selected 5 queries for each type: Star, Snowflake and Linear. Star queries were selected with varying forms of triple patterns that restrict the amount of intermediate results within the query (listed as S1 to S5 in the figures and tables below). An example of a Star query is depicted in Figure 3. This corresponds to Query S3 of the Wikidata list. Figure 4 is an example of a Path query and it corresponds to Query L5 in our list (L1 to L5). Figure 2 depicts a Snowflake query corresponding to Query F5 (Queries F1 to F5 plus two derived queries F2′ and F5′). More details will be discussed later.

We highlight that these query shapes are similar to those identified by Bonifatti *et al.* [18]. The patterns identified there, Chains, Trees, and Flowers from Wikidata queries, are similar to the Snowflake, Path and Star patterns from Watdiv. We follow the Watdiv name conventions in this paper. Our query selection is based on 17 different types of queries from the Watdiv templates.

Watdiv We complement our evaluation using the Watdiv evaluation framework. Although Watdiv is a stress testing evaluation suite, they provide a suit of queries that cover the most common SPARQL join types. As we described in Section 2, when applying Differential Privacy to a dataset it is of utmost importance to understand the transformations happening to the data during the query. The use of Watdiv helps us in that regard, since it provides a complete classification of the different join types in SPARQL. We use the 100 million triples dataset from the Watdiv Test Suite, one order of magnitude smaller than the Wikidata set.

As mentioned earlier, our Watdiv tests use 17 different types of queries from the Watdiv templates (Snowflake, Linear and Star queries with variations that include different number of join variables, different number of triple patterns per query, and cardinalities). The query naming corresponds to the queries in the Watdiv companion site [8]. As with the Wiki-

[7] https://iccl.inf.tu-dresden.de/web/Wikidata_SPARQL_Logs/en

[8] https://dsg.uwaterloo.ca/watdiv/basic-testing.shtml

data query set, Queries S1 to S5 are Star queries, L1 to L5 are Linear queries and F1 to F5, F2' and F5' are Snowflake queries. Syntactically, all these queries have one term that looks like a variable except that instead of being prefixed by a question mark, "?", the first symbols is "%". This term is used by the Watdiv tool as a placeholder for an URI that will be generated at query evaluation time. Different URIs will produce different mappings.

A Differential Privacy schema was designed for each query using the following procedure. Each variable (or URI placeholder term) that appeared as a subject of triple pattern of the query was designated as the center of a star BGP of the schema for the query. Then, any triple pattern with the same variable in the subject was made member of the same star BGP. For example. the schema associated with Wikidata's S4 query (Figure 3) has a single BGP, $\{((?human, P31, ?gender),$ $(?human, P172, ?ethnic\_group),$ $(?human, P735, ?given\_name),$ $(?human, P734, ?family\_name)\}$. The schema associated with Query F5 has three BGPs, $\{(?person, P2002, ?twitterName)\}$, $\{(?person, P106, ?occupation),$ $(?occupation, P279, ?tmp1)\}$, and $\{(?tmp1, P279, ?Q864503)\}$. To calculate the elastic sensitivity of a query Q given a Differential Privacy schema $\mathcal{P}$, we need to calculate the size $\mathcal{P}(G)$, given G, the graph that Q will access. This is done by adding the number of tuples in the mappings generated by each each of the BGPs in $\mathcal{P}$.

5.2. Results

5.2.1. Wikidata Experiments

Table 1 shows the results of the Wikidata query evaluations. Queries showing a polynomial in the Sensitivity column are evaluated using Elastic Sensitivity, while the non polynomial queries are evaluated using Restricted Sensitivity by applying a fixed value for the most popular join value within the star.As discussed before, the variable x in the expression must be chosen to maximize $\mathcal{S}^{(k)}(,)$.

Looking at the results in Table 1 and in Figure 5 the first thing to notice is that those queries with an error under 1% are all Star queries. This confirms our hypothesis (also reflected in the Watdiv simulations below) that star queries behave well. All queries with error greater than 1% are linear or snowflake queries. We can also notice several exceptions to this general rule: query F3 reported one of the lowest errors in the experiment: this is because in that query we find the *type* (property `P31` in Wikidata), resulting in a large query graph and thus lowering the overall query sensitivity. We see a similar behavior in queries L1, L3, and F2, none of them being Star queries and all of them having low error rates due to the large graph generated by the `P31` predicate. On the other hand, as we can see in Table 2, the query sensitivity of F5 is calculated by a second degree polynomial with large constants, $11077112403 + 545676x + x^2$, causing a query error beyond 6 orders of magnitude (Figure 5). The answer to this query is meaningless. That particular query is asking for the Twitter accounts of biologists, involving three Star patterns connected by two joins forming a path. These three join operations amplify the high sensitivity of querying a specific type of person (biologists, which generates a high count for the "most popular" value) result in large constants in the polynomial, and thus a high query sensitivity. In practice, an implementation should calculate the sensitivity of a query first, and if the result is higher than certain threshold, then the system should automatically reject the query for privacy concerns. If users do not have restrictions in the kind of queries they can ask to a public site (e.g., not having a fix set or predefined query templates), there will always be queries with high sensitivity that should not be answered. The last observation is about Queries F3′ and F5′. Both queries are modified versions of queries F3 and F5 respectively. These queries have a high error (specially query F5) since they are accessing very specific information. Query F5 contains a triple pattern of the form $?xwdt : P279, wd : Q864503$ which is accessing the Twitter accounts of biologists while query F3 asks for participants in a mass shooting event $(?x, wdt : P279, wd : Q21480300)$. In both F3′ and F5′ we removed the sensitive URIs (wd:Q21480300 and wd:Q864503) making more generic these queries and immediately decreasing the amount of error returned.

5.2.2. Watdiv

The most important conclusion of the Watdiv experiments is the effect of the graph size in sensitivity. Table 2 shows that all the Watdiv queries that need to join BGPs from their Differential Privacy schema have high sensitivity. It is important to notice that query graphs in WatDiv are up to three orders of magnitude smaller than in Wikidata, making each query sensitivity higher than Wikidata sensitivities. Recall that the variable x in the expression must be chosen to maximize $\mathcal{S}^{(k)}(,)$ Some of the errors are very high too. This

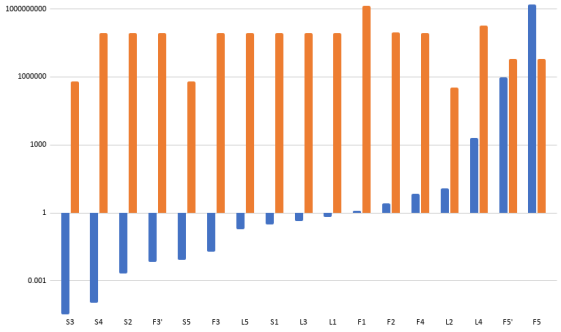

Figure 5. In this chart we plot the median error (blue) for the 17 evaluation queries from Wikidata and the size of the graph from which the COUNT was calculated. Notice that the graph size (in log scale) for each query is at least one order of magnitude greater, and thus the results for Snowflake and Linear shaped queries have more utility.

Query	% error	Graph Size	Sensitivity	Elastic Stability
S1	0.303	85M	1	x
S2	0.002	85M	1	x
S3	3.24E-05	620,322	1	x
S4	0.0001	85M	1	x
S5	0.008	647,282	1	x
L1	0.679	88M	986,303	999,613 + x
L2	12.22	342793	19,899	20,282 + x
L3	0.432	86M	986,284	999,618 + x
L4	2085	189M	28M	(1 + x)*(999,375 + x)
L5	0.19	88M	3,052	$(1 + x)^2$
F1	1.269	1343M	31.8	1 + x
F2	2.559	91M	29M	(6 + x)*(998,728 + x)
F3	0.018	88M	59,047	59,843 + x
F3'	0.006	88M	59,047	59,843 + x
F4	6.924	88M	88,109	89,298 + x
F5	1630M	6,305,923	10,905M	(21,117 + x)*(524,559 + x)
F5'	938784	6,305,909	10,904M	(21,115 + x)*(524,559 + x)

Table 1
Wikidata Queries used .

is mostly due to the method Watdiv uses to generate queries: it fixes a URI so that it can be used to control the cardinalities for the rest of the variables. That is to

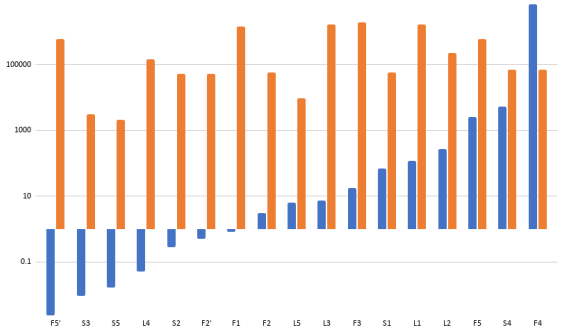

Figure 6. In this chart we plot the median error (blue) for the 17 evaluation queries from Watdiv and the size of the graph from which the COUNT was calculated. We see how in all queries but those *Snowflake* shaped queries we obtain the desired privacy parameters. For *Snowflake* queries the privacy highly depends on the semantic information provided by the administrator in the Differential Privacy schemes to identify most repeated attributes.

say, it can control the size of the intermediate results that are required to compute the joins. As with Wikidata, we modified queries F2 and F5 to not depend on the privacy of the URI as follows. We dropped the URI from the queries and introduced a selection over a numeric attribute. For example, in Query F5, depicted in Figure 2, we drop $(wsdbm/Retailer9247, offers, ?v0)$ and add the filter $(?v3 > 30)$. Now one can see that even though the sensitivity is still high, the errors are under %1. Figure 6 clearly contrasts queries with joins between BGPS and queries without joins. Except for F2′ and F5′, all queries without joins are listed before the queries with joins.

Table 3 shows a subset of the queries but this time using $\epsilon = 0.1$ to calculate the sensitivity. This is too low for most of the queries involving joins since the errors are high for the size of the graph. We left though F2′ and F5′. The error for F2′ is high but the error for F5′ is under $\sim$1%.

Errors over $\sim$1% are usually unacceptable. This means that our method, as it is in the case of relational databases, will be appropriate for large graphs when queries required several joins over the BGPs in a schema. Nevertheless, queries that are within a BGP will have sensitivity 1, and will have manageable noise.

Our plan is to look for other types of semantic properties that can be incorporated into the Differential Privacy schema to improve the calculation of sensitivity. One possible direction is to use some kind of restricted sensitivity (see [19, 20] that introduces a gen-

Query	% error	Graph Size	Sensitivity	Elastic Stability
F1	0.85	1.4M	23	5 + x
F2	3.17	58,622	17.03	1 + x
F2'	0.5	53,674	16.9	1 + x
F3	0.018	88M	59,047	59,843 + x
F4	6.8M	71,853	42.5M	(1 + x)*(2 + x)*(35,640 + x)
F5	1637.2	609,672	1662	(1 + x)*(2 + x)
F5′	0.002	609,672	20.8	2 + x
L1	118.4	1.6M	21.9	1 + x
L2	277.6	224,985	1869.7	1,906 + x
L3	7.37	1.6M	1	x
L4	0.05	146,964	1	x
L5	6.49	9,862	2612	2,681 + x
S1	68.85	58,746	17.8	3 + x
S2	0.28	53,237	1	x
S3	0.009	3,189	1	x
S4	5,386	70,010	1014	1,036 + x
S5	0.01	2,164	1	x

Table 2

Watdiv queries used in the evaluation spanning different query shapes, cardinalities, join variables and triple patterns, which are the parameters most affected in the calculation of the noise added to the COUNT operation. We set the Epsilon parameter to 1.0 obtaining error rates of maximum 17% for most of the queries (but S1, S4 and L2).

Query	% error	Sensitivity
F2	388.97	167.02
F2′	14.92	165.72
F4	19.46	219.98
F5	1887	217.94
F5′	0.101	201.85
L4	0.31	1
S2	8.95	1
S3	0.06	1
S5	0.34	1

Table 3

Watdiv queries evaluated using the privacy parameter Epsilon = 0.1. Only those queries not having a join or a FILTER present below 17% error percentages.

eral framework to incorporate semantic information into the calculation of sensitivity approximations).

6. Related Work

The study of how to guarantee the privacy of individuals contributing personal data to a dataset is a long studied problem. In this work we have focused on how to guarantee this privacy in RDF data graphs accessed through SPARQL queries using Differential Privacy. The related work can be roughly classified into those that provide some privacy guarantees to accesses to data stored in (social) graphs and those that guaranty privacy over the results returned by SPARQL queries. We briefly look over these works in this section.

6.1. SPARQL state of the art

In the context of the Semantic Web and the Web of Linked Data there have been several approaches to address privacy concerns related queries to RDF data. A good survey can be found in [21]. There is the basic privacy protection that a SPARQL engine must provide from queries that directly return individuals' data. Similar to the case of relational databases where this kind of protection can be done replacing some attribute values with nulls, the work presented in [22] and [23] describes how to use blank nodes to hide sensitive data. But, simply replacing URIs with blank nodes does not necessarily prevent a user to get private information about. Class information, such as size of a class, or the distribution of values over partitions of the class, which is often be made public can also reveal individual private data.

All the effort directed to provide tools to efficiently access Linked Data in the semantic Web is to make as much data as possible easily available. This has to be balanced with the need to protect the privacy of individuals. The aim of the work such as *k*-anonymity or *l*-diversity is to be able to provide answers to queries about classes; *k*-anonymity is used in [24, 25] to answer queries in RDF datasets. Unfortunately, it is well-known that *k*-anonymity does not provide formal guarantees for privacy. In contrast, Differential Privacy precisely prescribes how to characterize the privacy guaranties of data query answering algorithms. The only work known to us that directly uses Differential Privacy over RDF datasets is [21]. The authors propose a method for applying Differential Privacy to SPARQL queries. However, they provide a Differential Privacy realisation through local sensitivity without the use of a smoothing function, violating thus the privacy guarantees described in [26]. In [27] the authors provide a query language that processes RDF streams in a privacy-preserving fashion called SihlQL. Limiting queries to that language permits servers to continuously release privacy-preserving histograms (or distributions) from online streams.

6.2. Privacy in Social Graphs

One of the most well-known approaches to provide deferentially private queries over social graphs is by the use of Restricted Sensitivity [19]. The authors of Restricted Sensitivity define adjacency of graphs based on two notions: differences on edges and differences on vertices. The distance between two graphs, G_1 and G_2, is given by the smallest number of changes (either on edges or vertices) needed to transform G_1 and G_2 into the same graph, giving rise to two definitions of restricted sensitivity. The authors also provide efficient algorithms to calculate these sensitivities for a class of social graph queries that involve only one join. The authors of [20] extended the previous notion of edge-based restricted sensitivity by using weighted datasets. Briefly, the aim is to increase the utility of the answer by calculating noise with weights added to each edge that contributes to the solution. Our notion of Differential Privacy schema of sensitivity is a vertex-based sensitivity by observing that each $g_i \in G$ can be interpreted as a single node. The class of queries for which efficient implementations of Restricted Sensitivity exist corresponds to the execution of joins between Start patterns. Our proposed Elastic Sensitivity can be then interpreted as a generalization of Restricted Sensitivity. In Restricted Sensitivity, the polynomial associated to a query is always of degree 1 and the value of x is bounded by a constant (which is provided by the system administrator). Elastic Sensitivity is based on a variable selectivity (the values of x are obtained directly from the dataset), and generalizes to multiple joins. Other works such as [28] proposed a method to use Differential Privacy for subgraph counting queries with unrestricted joins (through node differential privacy), however, answering this type of queries is computationally difficult (NP-hard). The result is then more of theoretical interest and limited for general application.

7. Conclusions

In this paper we have introduced a framework to develop Differential Privacy tools for RDF Data repositories, and we have used the framework to develop an (ϵ, δ)-differential privacy SPARQL query engine for COUNT queries. A crucial component of our framework is the concept of Differential Privacy Schema. Without it, we would have not been able to develop a differential privacy preserving algorithm to publish data of acceptable quality. The concept is independent of the sensitivity approximation used and we hope that others can build on the concept to get better query answering algorithms.

We have implemented our algorithm and tested it using the Wikidata RDF database, queries from its log files and other example queries found at the Wikidata endpoint. We have also used the Watdiv framework to generate different types of queries and datasets. The simulations show the approach to be effective for queries over large repositories, such as Wikidata, and in many cases for queries within the 10 of thousands answers to aggregate. However, even though Elastic Sensitivity has been designed to bind the stability of joins, the sensitivity of a query with joins can still be very high. In the case of SQL queries in relational databases, in order to keep the noise under a single percentage digit, the databases should have over 1M tuples and $\epsilon = 1$. This is also the case for SPARQL queries. When working with the smaller Watdiv datasets, it is difficult to provide reasonable privacy guarantees for general queries because of the size of the query graph. The evaluation shows though that we can safely evaluate Star queries.

We can still apply several optimizations to our framework. For example, public graphs can be treated as public tables. If they participate in joins, we can directly use their most popular result mappings during calculation of the query sensitivity. We partially demonstrate the effect of such optimization when we assume one of the URIs in the queries to be public, resulting in lower sensitivities and fewer errors. We can also consider the approaches described in other state of the art works [7] to add aggregation functions like sum and averages to our framework.

There are many pending issues to address. From the more practical point of view, more operations need to be implemented. There are also issues about the impact that such algorithms will have on SPARQL query engines. From the more formal side, it is still important to keep searching for better approximations of local and global sensitivities as well as alternative definitions that are less onerous than differential privacy. One possibility is to find a way to apply Restricted Sensitivity to more types of queries by adding more semantic information to the Differential Privacy Schema.

References

[1] L. Menand, Why Do We Care So Much About Privacy?, *The New Yorker* **XCIV**(17) (2018), 24–29.

[2] N. Li, W. Qardaji, D. Su, Y. Wu and W. Yang, Membership privacy: a unifying framework for privacy definitions, in: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM, 2013, pp. 889–900.

[3] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov and M. Naor, Our data, ourselves: Privacy via distributed noise generation, in: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2006, pp. 486–503.

[4] L. Sweeney, k-anonymity: A model for protecting privacy, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **10**(05) (2002), 557–570.

[5] A. Machanavajjhala, J. Gehrke, D. Kifer and M. Venkitasubramaniam, l-diversity: Privacy beyond k-anonymity, in: *22nd International Conference on Data Engineering (ICDE'06)*, IEEE, 2006, pp. 24–24.

[6] F.D. McSherry, Privacy integrated queries: an extensible platform for privacy-preserving data analysis, in: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ACM, 2009, pp. 19–30.

[7] N. Johnson, J.P. Near and D. Song, Towards practical differential privacy for SQL queries, *Proceedings of the VLDB Endowment* **11**(5) (2018), 526–539.

[8] D. Kifer and A. Machanavajjhala, No Free Lunch in Data Privacy, in: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, ACM, New York, NY, USA, 2011, pp. 193–204. ISBN 978-1-4503-0661-4. doi:10.1145/1989323.1989345. http://doi.acm.org/10.1145/1989323.1989345.

[9] C. Dwork, Differential Privacy, in: *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, 33rd international colloquium on automata, languages and programming, part ii (icalp 2006) edn, Lecture Notes in Computer Science, Vol. 4052, Springer Verlag, 2006, pp. 1–12. ISBN 3-540-35907-9. https://www.microsoft.com/en-us/research/publication/differential-privacy/.

[10] K. Nissim, S. Raskhodnikova and A. Smith, Smooth Sensitivity and Sampling in Private Data Analysis, in: *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, ACM, New York, NY, USA, 2007, pp. 75–84. ISBN 978-1-59593-631-8. doi:10.1145/1250790.1250803. http://doi.acm.org/10.1145/1250790.1250803.

[11] K. Nissim, S. Raskhodnikova and A. Smith, Smooth Sensitivity and Sampling in Private Data Analysis, 2011, Draft full version v1.0. http://www.cse.psu.edu/~ads22/pubs/NRS07/NRS07-full-draft-v1.pdf.

[12] R. Cyganiak, D. Wood and M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, 2014.

[13] S. Harris and A. Seaborne, SPARQL 1.1 Query Language, W3C Last Call Working Draft http://www.w3.org/TR/2010/WD-sparql11-query-20101014/, 2012.

[14] G. Aluç, O. Hartig, M.T. Özsu and K. Daudjee, Diversified stress testing of RDF data management systems, in: *International Semantic Web Conference*, Springer, 2014, pp. 197–212.

[15] J. Pérez, M. Arenas and C. Gutierrez, Semantics and complexity of SPARQL, *TODS* **34(3)** (2009).

[16] D. Vrandečić and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Communications of the ACM* **57**(10) (2014), 78–85.

[17] Y. Liu, P. Zhang and M. Qiu, Fast Numerical Evaluation for Symbolic Expressions in Java, in: *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 2015, pp. 599–604. doi:10.1109/HPCC-CSS-ICESS.2015.19.

[18] A. Bonifati, W. Martens and T. Timm, Navigating the maze of wikidata query logs, in: *The World Wide Web Conference*, 2019, pp. 127–138.

[19] J. Blocki, A. Blum, A. Datta and O. Sheffet, Differentially Private Data Analysis of Social Networks via Restricted Sensitivity, in: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, ACM, New York, NY, USA, 2013, pp. 87–96. ISBN 978-1-4503-1859-4. doi:10.1145/2422436.2422449. http://doi.acm.org/10.1145/2422436.2422449.

[20] D. Proserpio, S. Goldberg and F. McSherry, Calibrating data to sensitivity in private data analysis: a platform for differentially-private analysis of weighted datasets, *Proceedings of the VLDB Endowment* **7**(8) (2014), 637–648.

[21] R.R.C. Silva, B.C. Leal, F.T. Brito, V.M. Vidal and J.C. Machado, A differentially private approach for querying RDF data of social networks, in: *Proceedings of the 21st International Database Engineering & Applications Symposium*, ACM, 2017, pp. 74–81.

[22] B.C. Grau and E.V. Kostylev, Logical foundations of privacy-preserving publishing of Linked Data, in: *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[23] R. Delanaux, A. Bonifati, M.-C. Rousset and R. Thion, Query-based Linked Data Anonymization, in: *International Semantic Web Conference*, Springer, 2018, pp. 530–546.

[24] F. Radulovic, R. García Castro and A. Gómez-Pérez, Towards the anonymisation of RDF data (2015).

[25] B. Heitmann, F. Hermsen and S. Decker, k-RDF-Neighbourhood Anonymity: Combining Structural and Attribute-based Anonymisation for Linked Data., in: *PrivOn ISWC*, 2017.

[26] C. Dwork, A. Roth et al., The algorithmic foundations of differential privacy, *Foundations and Trends® in Theoretical Computer Science* **9**(3–4) (2014), 211–407.

[27] D. Dell'Aglio and A. Bernstein, Differentially Private Stream Processing for the Semantic Web, in: *Proceedings of The Web Conference 2020*, WWW '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1977–1987. ISBN 9781450370233. doi:10.1145/3366423.3380265. https://doi.org/10.1145/3366423.3380265.

[28] S. Chen and S. Zhou, Recursive mechanism: towards node differential privacy and unrestricted joins, in: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ACM, 2013, pp. 653–664.