Semantic Web 0 (2023) 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] *ICREA and Universitat Pompeu Fabra, c/Roc Boronat 148, Barcelona, Spain*
E-mail: jorge.lobo@upf.edu
[c] *Departamento de Ciencias de la Computación, Universidad de Chile and IMFD, Beauchef 851, Santiago, Chile*
E-mail: folmedo@dcc.uchile.cl

Abstract. *Differential privacy* is a framework that provides formal tools to develop algorithms to access databases and answer statistical queries with quantifiable accuracy and privacy guarantees. The notions of differential privacy are defined independently of the data model and the query language at steak. Most differential privacy results have been obtained on aggregation queries such as counting or finding maximum or average values, and on grouping queries over aggregations such as the creation of histograms. So far, the data model used by the framework research has typically been the relational model and the query language SQL. However, effective realizations of differential privacy for SQL queries that required joins had been limited. This has imposed severe restrictions on applying differential privacy in RDF knowledge graphs and SPARQL queries. By the simple nature of RDF data, most useful queries accessing RDF graphs will require intensive use of joins. Recently, new differential privacy 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 results carry over to RDF and SPARQL. In this paper we provide a positive answer to this question by presenting an algorithm that can answer counting queries over a large class of SPARQL queries that guarantees differential privacy, if the RDF graph is accompanied with semantic information about its structure. We have implemented our algorithm and conducted several experiments, showing the feasibility of our approach for large graph 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 privacy breaches, studied their general characteristics, and concluded, that electronic privacy breaches always ended with giving an attacker the ability to identify (using public data) whether an individual 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.

[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 Olmedo was also supported by ANID - Millennium Science Initiative Program - Code ICN17_002

For the public good, such as the advance of public health, or the fair distribution of government resources, such 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. Especially in such cases, society expects a 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 perfect 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 [6] were developed but they were shown to have weak privacy guarantees [7].

In spite of its limitations [8], but because of its formal properties, a privacy notion that has gained a lot of acceptance 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 quantifiable utility guarantees to the data published about the class. The basic principle is simple. Given a universe $\mathbb{D}$ of all possible datasets 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 $\mathcal{R}$, f is said to be differentially private if it yields indistinguishable results when applied to *similar* datasets. Differential privacy uses randomized algorithms to answer queries, typically by adding noise to the true query results. This noise is calibrated according to query sensitivity —how much the query result varies between similar datasets—, turning the task of query sensitivity computation essential for the endeavor of differential privacy. In practice, calculating the exact sensitivity of a query is not trivial and approximations are used instead [9].

Even though the notion of differential privacy is in principle independent of the data model and query language at steak, so far most practical, automated implementations over well-established languages have been in the context of relational databases, over SQL, and have been restricted to aggregation queries or grouping. Aggregations are queries such as counting, finding maximum or average values over a certain data subset; grouping is the creation of histograms based on aggregations.

Furthermore, to allow for reasonable approximations of sensitivity, the support of these implementations for queries with *joins* has been rather limited [10]. It was only in 2018, when Johnson *et al.* [11] introduced a new approach to approximate sensitivity that can be applied to a wider class of SQL joins, with reasonable results.

In the past decades, graph data models have enjoyed a growing adoption in comparison to the more traditional relational model. One such notable example is the RDF data standard, queried over by the SPARQL language, which have become extremely popular, in particular, for their role in the development of the Semantic Web. By the simple nature of RDF, it can be stored using binary relations [12] and most interesting queries will require operations equivalent to joins. This raises the question whether Johnson *et al.*'s approach [11] can also be applied to RDF and SPARQL.

In this paper we provide a positive answer to this question by presenting an algorithm that can answer counting queries over a large class of SPARQL queries that guarantees differential privacy. This result has been made possible by introducing the notion of a *differential privacy schema* that allows redefining 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 a database administrator to define.

We demonstrate the applicability of our approach 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 [13].

The rest of the paper is organized as follows: in Section 2 we introduce the readers to the fundamental concepts of differential privacy. In Section 3, we present the core concepts of SPARQL used within the paper, including the notion of differential privacy schema. In Section 5 we prove the correctness of our proposed approximation to sensitivity and in Section 6 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 7, and we conclude the paper in Section 8.

2. Preliminaries about Differential Privacy

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 by the scientific community.

2.1. Definition

Intuitively, a randomized algorithm [14] 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 (a mild generalization of) the notion of distance used for defining *bounded* differential privacy [15], which quantifies (only) over pairs of datasets of the same size. In the remainder, 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 *neighbors*, 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 neighbor 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 $\mathcal{A}$ lies within S. The smaller the ϵ and δ, the closer these two probabilities are, and therefore, the less likely that an adversary can tell D and 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 $D_1, \ldots, D_n$ will contain (tagged) data points belonging to $D_1, \ldots, D_n$ 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 D_i. In the context of relational databases, the D_i's correspond to relational tables.

2.2. Realization via Global Sensibility

Establishing differential privacy for numeric queries of limited sensitivity is relatively simple. The Laplacian mechanism [16] 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 neighbor datasets; formally, $\mathsf{GS}_f = \max_{D,D' \mid D \sim D'} |f(D) - f(D')|$.

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 a 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* [7]. 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 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 [17]. Their approach relies on the notion of local sensitivity.

Definition 2. *The* local sensitivity $\mathsf{LS}_f(D)$ *of a numeric query* $f\colon \mathbb{D} \to \mathbb{R}$ *on dataset* $D \in \mathbb{D}$ *is defined as*

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

The local sensitivity $\mathsf{LS}_f^{(k)}(D)$ at distance $k \in \mathbb{N}_0$ of D *is defined as*

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

Observe that $\mathsf{LS}_f^{(0)}(D)$ coincides with $\mathsf{LS}_f(D)$. Similar to global stability, a dataset transformer $T\colon \mathbb{D} \to \mathbb{D}$, is *$\alpha$-local-stable* for a dataset D if $d(T(D), T(D')) \leqslant \alpha\, d(D, D')$ for every $D' \in \mathbb{D}$. And as with global sensitivity, $\mathsf{LS}_{f \circ T}(D) \leqslant \alpha\, \mathsf{LS}_f(D)$ whenever T is α-local-stable for D.

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 [11]. 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* $\mathcal{U}_f\colon \mathbb{D} \to \mathbb{R}_{\geqslant 0}$ *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. $\mathcal{U}_f(D) \geqslant \mathsf{LS}_f(D)$ *for all dataset D, and*
2. $\mathcal{U}_f(D) \leqslant e^{\beta}\, \mathcal{U}_f(D')$ *for all neighbor 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 [18, Corollary 2.4].

Theorem 2. *Let* $f\colon \mathbb{D} \to \mathbb{R}$ *be a numeric query and let* $\mathcal{U}_f\colon \mathbb{D} \to \mathbb{R}_{\geqslant 0}$ *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*

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

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 itself, 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.* [11], "it requires running the query on every possible neighbor 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* $\mathcal{U}_f^{(k)}$ *is a pointwise upper bound of the local sensitivity* $\mathsf{LS}_f^{(k)}$ *of* f *at distance k, that is,*

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

Then,

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

is a β-smooth upper bound of the local sensitivity $\mathsf{LS}_f(D)$ of f on D, where $size(D)$ denotes the number of rows (in all the tables) in D.

The goal of Section 5 is to apply the differential privacy mechanism from Theorem 2 to SPARQL counting queries. To do so, we will use Lemma 1 to derive smooth upper bounds of the local sensitivity of queries. In turn, this requires constructing upper bounds for the local sensitivity of queries at fixed distances, for which we will leverage *local* stability properties of SPARQL dataset transformers.

3. Toward Differential Privacy over RDF Graphs

In this section we examine the semantic information that is necessary considering over RDF graphs, in order to answer counting queries in a differentially private manner. This comprises a data schema and upper bounds on the predicate multiplicities.

3.1. Privacy schema

Motivation

As mentioned earlier, the goal of differential privacy is to protect the (possibly sensible) contribution of each individual within a dataset when publicly releasing aggregate information about the dataset —in our case, the result of counting queries. In the relational model, individuals are typically *identified with rows* of the database which significantly simplifies all the technical development. For instance, if the database at stake consists of a *single* table, we consider two instances of the database neighboring, *i.e.* differing in the contribution of a single individual, if they differ in a single row. On the other hand, if the database consists of *multiple* tables, we consider two database instances neighboring if they differ in a row of some of the tables (see paragraph *Multi-table datasets* in Section 2). The underlying assumption behind this is that each table groups attributes of individuals in a particular entity type, *e.g.* people, political parties or companies, or part thereof, whose identities must be protected.

To be able to apply differential privacy to a dataset in the form of an RDF graph, we must thus begin by identifying the different types of entities present in the graph, and the set of individuals in each type. Consider, for instance, the RDF graph $\mathcal{G}$ in Figure 1, which will be the running example of our presentation. This graph contains information about three types of entities: people, companies and cites. In particular, it contains information about two people (depicted in blue), two companies (depicted in red) and two cities (depicted in green). Said otherwise, there are two individuals of each entity type, adding up to six individuals in all. When querying the graph, we will be interested in protecting the contribution of all these individuals, and when applying differential privacy techniques to this end, we will then consider as a neighbor any other graph that differs in the contribution of either of them.

We refer to the semantic information necessary to identify the individuals in an RDF graph $\mathcal{G}$ as a *differential privacy schema*. More formally, its goal is to partition $\mathcal{G}$ as a set $\{g_1, \ldots, g_n\}$ of sub-graphs, where each g_i represents the contribution of an individual, and $\mathcal{G} = \biguplus_i g_i$ is the disjoint union of all these sub-graphs. For example, the graph in Figure 1 is decomposed as the disjoint union of the pair of sub-graphs in blue, the pair of sub-graphs red and the pair of sub-graphs green. Observe that in the relational model, this corresponds to nothing more than understanding a (set of) table(s) as the disjoint union of its (their) rows. Here, our interest is to protect the contribution of the individuals represented by each g_i. In Figure 1, this means, for example, the data related to Alice and to the Walt Disney company.

Formal definition

We briefly review some basic RDF terminology following standard notation used in the literature [19, 20], where more details can be found. The RDF language assumes the existence of an infinite set U (of URI references), an infinite set B (of blank nodes), and an infinite set L (of RDF literals). An RDF *triple* is a term of the form $(v_1, v_2, v_3) \in (U \cup B) \times U \times (U \cup B \cup L)$. An RDF dataset is a finite set of RDF triples. RDF triples are interpreted as labeled arcs or edges in a directed graph from a vertex v_1, called the triple *subject*, to a vertex v_3, called the triple *object*, and label v_2, called the triple *predicate*. Figure 1 shows an example of an RDF graph $\mathcal{G}$. We denote by $voc(\mathcal{G})$ the finite subset of elements from $(U \cup B \cup L)$ that appear in $\mathcal{G}$. More importantly, because of the nature of the aggregation queries under consideration, we will restrict ourselves to graphs without blank nodes, *i.e.* graphs where $B = \emptyset$.[2] We also require a restricted version of the concept of *triple patterns*, which, similarly

[2] Calculation of query sensitivity will depend on the the domain of blank nodes which can change under different contexts and implementations. This is a topic of future research.

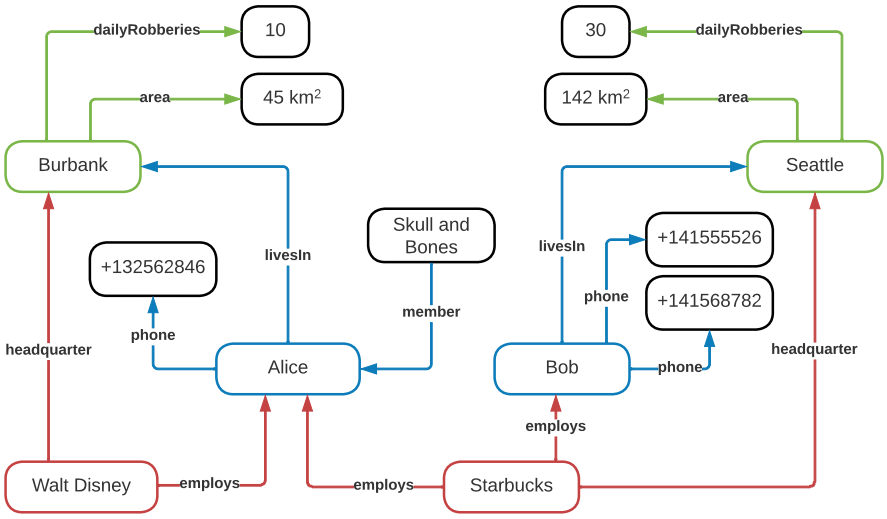

Figure 1. RDF graph $\mathcal{G}$ containing information about three types of entities: people, companies and cites.

to RDF triples, are terms of the form $(v_1, v_2, v_3) \in (U \cup B \cup V) \times U \times (U \cup B \cup L \cup V)$, with V an infinite set of variables, and *basic graph patterns* (BGPs) which are finite sets of triple patterns.[3]

To hint how we can formally define entities within an RDF graph, observe first the six colored sub-graphs identified in Figure 1: two green, two blue and two red. All have a star shape [21], consisting of a "center" with outgoing and/or incoming edges, *i.e.* predicates. Sub-graphs representing individuals of the same entity type are built from the same set of predicates. For example, both (blue) sub-graphs, representing people, are built from predicates phone, livesIn and member. We can characterize entity types through a set of triple patterns (*i.e.* a BGP) that share a common or "join" vertex. Formally, a *join vertex* in a BGP is a variable that appears either as a subject or as an object multiple times in the BGP. The type of BGP's that we need have further restrictions, which are captured by the notion of *star BGP* below:

Definition 4 (Star BGP)**.** *A BGP is called a* star *if*

1. *both the subject and the object of all its triple patterns are variables,*
2. *all triple patterns have different predicates, and*
3. *it consists of either*
   (a) a single triple pattern with no join vertex, i.e. a triple pattern whose subject and object are distinct variables, or
   (b) multiple triple patterns with a single join vertex, which appears once and only once in every triple pattern

Example 3.1 (Star BGP)**.** *The three stars employed to identify the different entities in our running example (Figure 1) are (modulo variable renaming):*

$$S_1 = \{(?\mathsf{c}_1, \mathsf{livesIn}, ?\mathsf{o}_1), (?\mathsf{c}_1, \mathsf{phone}, ?\mathsf{o}_2), (?\mathsf{s}_2, \mathsf{member}, ?\mathsf{c}_1)\}$$

$$S_2 = \{(?\mathsf{c}_2, \mathsf{employs}, ?\mathsf{o}_3), (?\mathsf{c}_2, \mathsf{headquarter}, ?\mathsf{o}_4)\}$$

$$S_3 = \{(?\mathsf{c}_3, \mathsf{area}, ?\mathsf{o}_5), (?\mathsf{c}_3, \mathsf{dailyRobberies}, ?\mathsf{o}_6)\}$$

Note that in each S_i, the vertex denoted by $?\mathsf{c}_i$ plays the role of the star center, joining all the triple patterns in S_i. Furthermore, there are no common predicates across S_1, S_2 and S_3. Formally, we define the *center* of a star as the join vertex if the star contains multiple patterns, or the variable appearing in the subject of the triple pattern in case it consists of a single triple pattern.[4] We say that a set of stars is *pairwise predicate-*

[3] The more general definition of triple patterns allows also for variables in the predicate component of triple patterns.

[4] The notion of *star BGP* that we use here is similar to that of *star query* from [21], except that in a star query the center of the star must always appear as subject.

disjoint (or simply *pairwise disjoint* when no ambiguity arises), if no pair of stars in the collection share a common predicate. They thus define what we call a differential privacy schema:

Definition 5 (Dp-schema)**.** *A* differential privacy schema *(*dp-schema*, for short) $\mathcal{P}$ is a finite pairwise predicate-disjoint set of stars.*

The set $\{S_1, S_2, S_3\}$ is a dp-schema. as well as its sub-set $\{S_1, S_2\}$. However, this second schema falls short of describing the whole graph $\mathcal{G}$, as it leaves out the information related to cities. To formally capture this completeness condition, we require the notion of *induced sub-graph*, whose formal definition is based on (solution) mappings. In SPARQL, a (solution) mapping is a partial mapping from variables to URI references or blank nodes, $\mu : V \to U \cup B$. For a triple pattern tp, $\mu(tp)$ denotes the triple obtained after replacing the variables in tp according to μ. If μ is defined for all variables in tp, $\mu(tp)$ is an RDF triple. Given a graph $\mathcal{G}$, the solution mappings of a triple pattern tp over $\mathcal{G}$, denoted by $[\![tp]\!]_{\mathcal{G}}$, is defined by the set $\{\mu \mid \mu(tp) \in \mathcal{G}\}$. Now, we are ready to define the concept of *induced sub-graph*:

Definition 6 (Induced sub-graphs)**.** *Let $\mathcal{G}$ be an RDF graph and $?x$ the center of a star BGP S. Given some $y \in U$, let μ_y denote the mapping $\{?x \to y\}$. The sub-graph of $\mathcal{G}$ induced by S and y, denoted by $ind(S)^y_{\mathcal{G}}$, is defined by the set of* RDF *triples $\{\mu(\mu_y(tp)) \mid \mu \in [\![\mu_y(tp)]\!]_{\mathcal{G}} \wedge tp \in S\}$, and the subgraphs of $\mathcal{G}$ induced by S, denoted by $S(\!|\mathcal{G}|\!)$, is the set of* RDF *sub-graphs $\{ind(S)^y_{\mathcal{G}} \mid y \in voc(\mathcal{G})\}$.*

Example 3.2 (Induced sub-graph)**.** *The star S_1 induces two sub-graphs g_1 and g_2 over $\mathcal{G}$, i.e. $S_1(\!|\mathcal{G}|\!) = \{g_1, g_2\}$. These correspond to the blue sub-graphs in Figure 1, which are formally defined as:*

$$g_1 = \{(\text{Alice}, \text{livesIn}, \text{Burbank}), (\text{Alice}, \text{phone}, +132562846), (\text{Skull and Bones}, \text{member}, \text{Alice})\}$$

$$g_2 = \{(\text{Bob}, \text{livesIn}, \text{Seattle}) (\text{Bob}, \text{phone}, +141555526), (\text{Bob}, \text{phone}, +141568782)\}$$

Likewise, $S_2(\!|\mathcal{G}|\!) = \{g_3, g_4\}$, where

$$g_3 = \{(\text{Walt Disney}, \text{employs}, \text{Alice}), (\text{Walt Disney}, \text{headquarter}, \text{Burbank})\}$$

$$g_4 = \{(\text{Starbucks}, \text{employs}, \text{Alice}), (\text{Starbucks}, \text{employs}, \text{Bob}), (\text{Starbucks}, \text{headquarter}, \text{Seattle})\}$$

Intuitively, the set of induced sub-graphs $S(\!|\mathcal{G}|\!)$ can be recovered by evaluating S over $\mathcal{G}$, but assuming that the triple patterns in S are optional. This assumption is already exposed by the example, where the triple pattern $(?\mathsf{s}_2, \text{member}, ?\mathsf{c}_1)$ belongs to S_1, but it is not materialized in g_2. From the privacy point of view, the values of the star center are the unique identifiers of the entities contributing the data in each sub-graph, and their values must be kept confidential.

The notion of induced sub-graphs naturally extends from single stars, to dp-schemas, *i.e.* sets of stars. Concretely, we let $\mathcal{P}(\!|\mathcal{G}|\!) = \bigcup_{S \in \mathcal{P}} S(\!|\mathcal{G}|\!)$ be the set of sub-graphs of $\mathcal{G}$ induced by dp-schema $\mathcal{P}$. In our example, $\{S_1, S_2\}(\!|\mathcal{G}|\!) = \{g_1, g_2, g_3, g_4\}$.

Now we have all the prerequisites to define the core concept of this section:

Definition 7 (Dp-schema compliance)**.** *We say that an* RDF *graph $\mathcal{G}$* complies *with a dp-schema $\mathcal{P}$ iff $\mathcal{G}$ coincides with the graph induced by $\mathcal{P}$ in $\mathcal{G}$, i.e. if $\mathcal{G} = \bigcup_{g \in \mathcal{P}(\!|\mathcal{G}|\!)} g$.*

Example 3.3 (Dp-schema compliance)**.** *Our running example $\mathcal{G}$ complies with dp-schema $\{S_1, S_2, S_3\}$. In contrast, it does not comply with dp-schema $\{S_1, S_2\}$.*

In summary, if a graph $\mathcal{G}$ complies with a dp-schema $\mathcal{P}$, the schema partitions the graph into a finite set $\mathcal{P}(\!|\mathcal{G}|\!)$ of sub-graphs, which intuitively model the different individuals in the graph. The fact that sub-graphs are disjoint follows from the definition of dp-schema, which requires stars in the schema to be pairwise predicate disjoint, and that all the occurrences of variables in a star are different except for the center of the star. The fact that the set of sub-graphs cover the entire graph follows from the definition of compliance.

Discussion

We now address a few key points about dp-schemas.

No shared attribute between entities. At first sight, the requirement that stars in a dp-schema be predicate pairwise disjoint might seem a limitation, as it requires

that each attribute belong to a single entity type. For instance, it might seem natural to consider that predicate employs should be part of both the employer and the employee, and it should be thus present in both S_2 (which identifies companies) and S_1 (which identifies people). However, for the sake of protection it makes no difference to which star it belongs, since our application of differential privacy will protect the contribution of all individuals, regardless of its type.

Existence of dp-schemas. RDF graphs always admit compliant dp-schemas. In particular, every graph complies with a *trivial* dp-schema comprising the union of all singleton BGP's of the form $\{(?x, p, ?y)\}$, where p ranges over the set of predicates appearing in $\mathcal{G}$. Intuitively, this schema indicates that each RDF triple is the contribution of a different individual. Even though this is a valid dp-schema, it will yield very weak privacy guarantees. In general, database administrators should aim to provide dp-schemes with a maximal number of triple patterns per star since, as we will see later, it will allow better approximations of queries' local sensitivity, and thus, better privacy guarantees.

Compliance verification. Checking whether an RDF graph complies with a given schema $\mathcal{P}$ is algorithmically straightforward as it amounts to verifying that all predicates in the graph appear also in (some star of) the schema (which, using a hashing algorithm, can be done in $O(n)$ steps, n being the number of predicates in the graph).

Dp-schema provision. For practical purposes, we assume that the database administrator of the RDF graph at stake is responsible for designing the dp-schema the graph shall comply with, and for ensuring the compliance as the graph evolves. In this latter regard, observe that removing an RDF triple from the graph always preserves the dp-schema compliance, and adding a triple also preserves compliance provided the predicate in the triple already appears in the dp-schema. We believe this is a natural assumption, as in the relational data model this would correspond to changing the schema of the database by adding a new attribute to a relation if the new predicate is incorporated into an existing star of the dp-schema or creating a new table if the predicate is added to the dp-schema as a new star.

3.2. Predicate multiplicity

Automatic approaches to answer dataset queries in a differentially private manner are typically obtained by adding noise to the query results, calibrated according to their sensitivity. Thus, a prerequisite to apply differential privacy to counting queries over RDF graphs is that they have bounded sensitivity. Unfortunately, this does not occur in the general case.

To see this, consider graph $\mathcal{G}$ of our running example and query "How many phone numbers are currently in use?". If we evaluate the query over $\mathcal{G}$, the answer is 3. Now assume we consider a neighboring graph $\mathcal{G}'$, where Bob's contribution (*i.e.* sub-graph g_2) is replaced by somebody else's contribution. The query answer over this neighboring graph can certainly be any integer $n \geqslant 1$, since a priori we do not know how many phone numbers this new individual might have. Therefore, the sensitivity of the query becomes unbounded.

This problem arises because of the presence of predicates that are not *one-to-one*. To recover bounded sensitivities, we have to restrict ourselves to predicates that have bounded multiplicity. For instance, if the administrator of graph $\mathcal{G}$ requires that individuals declare at most 5 phone numbers, then the above query will have a local sensitivity of at most 5 (recall that if $n_1, n_2 \leqslant \alpha$, then $|n_1 - n_2| \leqslant \alpha$). This approach was already taken by other authors [9, 22] to bound the sensitivity of counting queries (in the presence of *joins*), and is the price one has to pay to apply differential privacy over RDF graphs.

On the formal level, we associate such bounds to triple patterns rather than to predicates. This is because in the presence of compliant dp-schemas, predicates are identified with triple patterns (every predicate within a dp-schema occurs in a single triple pattern, in a single star).

Definition 8 (Triple-pattern multiplicity)**.** *Let $\mathcal{G}$ be an* RDF *graph that complies with a dp-schema $\mathcal{P}$. A* multiplicity bound *κ associates to each triple pattern tp in a star S of $\mathcal{P}$ an integer $\kappa(tp)$ that upper-bounds the number of solution mappings $\mu \in [\![tp]\!]_{\mathcal{G}}$ with the same image for the center of S.*

Many predicates (or equivalently, triple patterns) would have a natural multiplicity bound of 1. For instance, a city has a unique area and a unique number of dailyRobberies. Likewise, we can assume that a person livesIn a single city (at least for formal purposes). On the other hand, if a predicate does not admit a natural bound for its multiplicity, think *e.g.* of predicate friend, we can either a) choose an upper bound that covers most of the cases in practice or b) establish an upper bound in accordance with the size of the graph that the administrator is willing to support.

Henceforth, in the remainder we assume the system administrator provides a dp-scheme $\mathcal{P}$ and that every graph $\mathcal{G}$ is compliant with the dp-scheme $\mathcal{P}$. Furthermore, we assume that the administrator establishes an upper bound $\kappa(tp)$ for each triple pattern tp in $\mathcal{P}$. Hence, the graph space that we consider for the purposes of differential privacy will be that of graphs that comply with both $\mathcal{P}$ and κ.

For bounding the local sensitivity of queries in Section 5, it will suffices a coarser notion of multiplicity, at the level of stars rather than triple patterns. The required generalization is straightforward:

Definition 9 (Star multiplicity)**.** *Let $\mathcal{G}$ be an* RDF *graph that complies with a dp-schema $\mathcal{P}$ and has a* multiplicity bound κ. *We call the* multiplicity *of a star $S \in \mathcal{P}$, by notation convenience written $\kappa(S)$, to the product of the multiplicity bound of the triple patterns in S, i.e. $\kappa(S) = \prod_{tp \in S} \kappa(tp)$.*

Example 3.4 (Star multiplicity)**.** *Assume that a graph administrator adopts dp-schema $\mathcal{P} = \{S_1, S_2, S_3\}$ and requires that each individual* liveIn *(at most) a single city, declare at most five phone numbers and be member of at most 3 secret societies. Then the star S_1 will have multiplicity $\kappa(S_1) = 1 \times 5 \times 3 = 15$.*

4. Queries

In this section we describe the subset of queries over RDF graphs for which we provide differential privacy, and show how dp-schemes enable a decomposition result for the evaluation of such queries.

4.1. Supported queries

We develop differential privacy for counting queries over the SPARQL fragment of basic graph patterns with filter expressions, also known as *constrained basic graph pattern* (CBGP) [21]. In this fragment, a query is denoted by a pair

$$\bar{B} = \langle B, F \rangle \ ,$$

where B is a finite set of triple patterns, *i.e.* a BGP, and $F = \{f_1, \ldots, f_n\}$ is a finite (possibly empty) set of filter expressions. $\bar{B}$ represents the SPARQL graph pattern

$$P = ((\ldots(B \ \mathsf{FILTER}\ f_1)\ldots)\ \mathsf{FILTER}\ f_n) \ ,$$

and its meaning over an RDF graph $\mathcal{G}$, denote by $[\![\bar{B}]\!]_{\mathcal{G}}$, is the multiset of solution mappings $[\![P]\!]_{\mathcal{G}}$ as defined by the standard semantics of SPARQL queries [20].

For simplicity, we consider only CBGPs that are semantically valid. We also assume that in a graph $\mathcal{G}$ that complies with a dp-schema $\mathcal{P}$, all predicates appearing in triple patterns of a CBGP also appear in $\mathcal{P}$.

Example 4.1. *Take* RDF *graph $\mathcal{G}$, which complies with dp-schema $\mathcal{P} = \{S_1, S_2, S_3\}$. Now assume we want to know how many people have a coworker in a company with headquarters in a city with over 20 daily robberies? The query can be cast in terms of the CBGP $\bar{B} = \langle B, F \rangle$, where*

$$\begin{aligned} B \ &= \ \{(?\mathsf{x}, \mathsf{employs}, ?\mathsf{p}_1),\ (?\mathsf{x}, \mathsf{employs}, ?\mathsf{p}_2), \\ &\qquad (?\mathsf{p}_2, \mathsf{livesIn}, ?\mathsf{c}),\ (?\mathsf{c}, \mathsf{dailyRobberies}, ?\mathsf{n})\} \\ F \ &= \ \{?\mathsf{n} \geqslant 20\} \end{aligned}$$

Triple patterns in an RDF graph compliant with a dp-schema $\mathcal{P}$ naturally inherit the notion of center from the star they "belong to". Specifically, for a star $S \in \mathcal{P}$ and a triple pattern $tp = (s, p, o)$ such that (X, p, Y) belongs to S, for some variables X, Y, we let $center(tp, \mathcal{P}) = s$ if X is the center of S; otherwise $center(tp, \mathcal{P}) = o$. For instance, in the above example we have

$$\begin{aligned} center((?\mathsf{x}, \mathsf{employs}, ?\mathsf{p}_1), \mathcal{P}) \ &= \ ?\mathsf{x} \\ center((?\mathsf{x}, \mathsf{employs}, ?\mathsf{p}_2), \mathcal{P}) \ &= \ ?\mathsf{x} \ , \end{aligned}$$

since star S_2 which contains the triple pattern

$$(?\mathsf{c}_2, \mathsf{employs}, ?\mathsf{o}_3)$$

has center $?\mathsf{c}_2$. Note that center is a well-defined function because the predicate of any triple pattern can only appear in single star of the dp-schema. In the remainder, we write $center(tp)$ for $center(tp, \mathcal{P})$ when $\mathcal{P}$ is understood from the context (*e.g.* when referring to the dp-schema established by the administrator of the RDF graph at stake).

Finally, a *user query* is a CBGP $\bar{B}$ wrapped by either of the two following aggregation operations:

1. $\mathsf{COUNT}^{?x}(\bar{B})$, whose semantics $[\![\mathsf{COUNT}^{?\mathsf{x}}(\bar{B})]\!]_{\mathcal{G}}$ is defined as the cardinality of the multiset $[\![\bar{B}]\!]_{\mathcal{G}}$.
2. $\mathsf{COUNT}^{\mathsf{DISTINCT}\ ?x}(\bar{B})$, whose semantics $[\![\mathsf{COUNT}^{\mathsf{DISTINCT}\ ?\mathsf{x}}(\bar{B})]\!]_{\mathcal{G}}$ is defined as the cardinality of the set $\{\mu(?x) \mid \mu \in [\![\bar{B}]\!]_{\mathcal{G}}\}$.

3. $\mathsf{COUNT}^{?x}_{?x_1 \ldots ?x_n}(\bar{B})$, where $?x, ?x_1, \ldots, ?x_n$ are variables appearing in $\bar{B}$, and whose semantics $[\![\mathsf{COUNT}^{?x}_{?x_1,\ldots,?x_n}(\bar{B})]\!]_{\mathcal{G}}$ is defined by grouping the solution mappings from $[\![\bar{B}]\!]_{\mathcal{G}}$ according to (the values they assign to) variables $?x_1 \ldots ?x_n$ and returning the number of mappings within each of the resulting groups. Loosely speaking, this corresponds to a histogram over tuples grouped by keys created by the different combinations of the values assigned to variables $?x_1 \ldots ?x_n$ by the solution mappings from $[\![\bar{B}]\!]_{\mathcal{G}}$.

Example 4.2. *The query from the previous example can be expressed as* $\mathsf{COUNT}^{\mathsf{DISTINCT}\ ?\mathsf{p}_1}(\bar{B})$ *using the previous CBGP.*

4.2. Evaluation decomposition

Continuing with the previous example, assume we want to evaluate B (from Example 4.1) over $\mathcal{G}$ (from Figure 1), that is, to compute $[\![B]\!]_{\mathcal{G}}$ (observe that if we are interested in obtaining $[\![\langle B, F\rangle]\!]_{\mathcal{G}}$ instead, we simply add a FILTER operation on top of the evaluation of $[\![B]\!]_{\mathcal{G}}$). We can do this in a compositional fashion, leveraging the partition that dp-schema $\mathcal{P} = \{S_1, S_2, S_3\}$ induces on $\mathcal{G}$. Concretely, we can split B as

$$\begin{aligned} B_1 &= \{(?\mathsf{x}, \mathsf{employs}, ?\mathsf{p}_1)\} \\ B_2 &= \{(?\mathsf{x}, \mathsf{employs}, ?\mathsf{p}_2)\} \\ B_3 &= \{(?\mathsf{p}_2, \mathsf{livesIn}, ?\mathsf{c})\} \\ B_4 &= \{(?\mathsf{c}, \mathsf{dailyRobberies}, ?\mathsf{n})\} \end{aligned} \tag{1}$$

Hence, for any query $\bar{B} = \langle B, F\rangle$, we can formally define a split $B_{\div\mathcal{P}}$ of B from a dp-schema $\mathcal{P}$, as follows: $B_{\div\mathcal{P}} = \{B_1, \ldots, B_n\}$ iff the following two conditions hold

1. every $B_i \in B_{\div\mathcal{P}}$ i) is a maximal subset of B for which there exists a star $S \in \mathcal{P}$ such that $pred(B_i) \subseteq pred(S)$, and ii) has no predicate repetitions;
2. for any two triple patterns $tp, tp' \in B_i$, $center(tp) = center(tp')$.

Because the stars in $\mathcal{P}$ are predicate disjoint and the B_i's in $B_{\div\mathcal{P}}$ are maximal, the split $B_{\div\mathcal{P}}$ is unique. In the context of Condition 1 above, we call S the *covering star* of B_i. Moreover, we call a BGP B *elementary* if $|B_{\div\mathcal{P}}| = 1$, and, by construction, all members of a split will be elementary.

Example 4.3. *For the CBGP from Example 4.1, B is split into four elementary BGPs by dp-schema $\mathcal{P} = \{S_1, S_2, S_3\}$, i.e. $B_{\div\mathcal{P}} = \{B_1, B_2, B_3, B_4\}$ as defined in Equation 1 above. Note that B_1 and B_2 have both the same covering star S_2, but they are consireded different elementary BGPs because they share predicate* employs. *The covering stars of B_3 and B_4 are S_1 and S_3, respectively.*

If B were extended, e.g., with triple pattern

$$tp = (\mathsf{Skull\ and\ Bones}, \mathsf{member}, ?\mathsf{p}_1)\ ,$$

the splitting $B_{\div\mathcal{P}}$ would contain a fifth, member $B_5 = \{tp\}$. In this case, B_3 and B_5 share the same covering star, S_1, but remain different members of $B_{\div\mathcal{P}}$ because their triple patterns have different centers ($?\mathsf{p}_2$ is the center of triple patterns in B_3 and $?\mathsf{p}_1$ the center of the triple pattern in B_5). Alternatively, if B were extended, e.g., with triple pattern

$$tp' = (?\mathsf{x}, \mathsf{headquarter}, \mathsf{Burbank})\ ,$$

the number of elementary BGPs does not change but B_1 and B_2 should be both augmented with tp' due to the maximality condition of each B_i. (In this case, B_1 and B_2 remain being covered by star S_2.)

Note also that the elementary BGP where a triple pattern belongs to is determined by the center of the triple pattern, all triple patterns in the same split must share the same variable or RDF term as their center. The interest in $B_{\div\mathcal{P}} = \{B_1, \ldots, B_n\}$ resides in that it lets us isolate the fragment of the graph necessary to answer each B_i. Assume we denote by $\mathcal{G}_{S_i}$ the subgraph induced by star S_i, *i.e.* $\mathcal{G}_{S_i} = \bigcup_{g \in S_i(\!|\mathcal{G}|\!)} g$.

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

$$[\![B_i]\!]_{\mathcal{G}} = [\![B_i]\!]_{\mathcal{G}_{S_i}}$$

Then, following the terminology defined in [23] for joins between multisets of solution mappings, we can extend the lemma to B as follows:

Lemma 3.

$$[\![B]\!]_{\mathcal{G}} = [\![B_1]\!]_{\mathcal{G}_{S_1}} \bowtie ([\![B_2]\!]_{\mathcal{G}_{S_2}} \bowtie \cdots \bowtie ([\![B_{n-1}]\!]_{\mathcal{G}_{S_{n-1}}} \bowtie [\![B_n]\!]_{\mathcal{G}_{S_n}}))$$

We have already observed that $\bar{B}$ is such that $[\![B]\!]_{\mathcal{G}}$ can be evaluated using only equi-joins. Therefore, there must exist an ordering of the elements in $B_{\div\mathcal{P}}$

such that $[\![B_i]\!]_{\mathcal{G}_{S_i}} \bowtie [\![B_{i+1}]\!]_{\mathcal{G}_{S_{i+1}}}$ can also be done with equi-joins. In other words, an ordering where $|var(B_i) \cap var(B_{i+1})| = 1$ for all $1 \leqslant i < n$. We call this order a *normal ordering* of $B_{\div\mathcal{P}}$, and without loss of generality denote by $?x_i$ the variable in the equi-join $[\![B_i]\!]_{\mathcal{G}_{S_i}} \bowtie [\![B_{i+1}]\!]_{\mathcal{G}_{S_{i+1}}}$. For convenience, in the remainder we assume that the indexing used for $B_{\div\mathcal{P}}$ follows a normal order. Note that this is already the case for the splitting from Example 4.3 (B_1 and B_2 share variable p_2, and B_2 and B_3 share variable c).

We are now in a position to establish differential privacy for SPARQL count and histogram queries.

5. Towards Differential Privacy for SPARQL

In this section we develop all the prerequisites to extend Lemma 1 from the relational model to the graph model, in terms of the SPARQL queries over RDF graphs described in the previous section.

5.1. Preliminary notions

We begin defining the notion of size and distance between RDF graphs. These are straightforward adaptations of the relational case, where the induced subgraphs play the role of table rows. Concretely, the size of a graph refers to the number of individuals present in it. Formally, given an RDF graph $\mathcal{G}$ that complies with a dp-schema $\mathcal{P} = \{B_i\}_{i\in I}$, we define the *size* of $\mathcal{G}$ w.r.t. $\mathcal{P}$ as $size(\mathcal{G})_{\mathcal{P}} = \sum_{i\in I} |B_i(\!|\mathcal{G}|\!)|$.

Moreover, we say that two graphs are k *far apart* if one can be obtained from the other by replacing k of its induced sub-graphs. Formally, given a pair of RDF graphs $\mathcal{G}_1, \mathcal{G}_2$ that comply with a dp-schema $\mathcal{P} = \{B_i\}_{i\in I}$ and such that $size(\mathcal{G}_1)_{\mathcal{P}} = size(\mathcal{G}_2)_{\mathcal{P}}$, their *distance* is defined as the size of their difference, *i.e.* $d(\mathcal{G}_1, \mathcal{G}_2)_{\mathcal{P}} = size(\mathcal{G}_1 \setminus \mathcal{G}_2)_{\mathcal{P}}$. Note that in the general case where the sizes of $\mathcal{G}_1$ and $\mathcal{G}_2$ need not coincide, their distance is defined as the size of the their symmetrical difference $(\mathcal{G}_1 \setminus \mathcal{G}_2) \cup (\mathcal{G}_2 \setminus \mathcal{G}_1)$, but when the sizes coincide, this reduces to the size of either their differences, making distance commutative as expected.

Finally, this notion of distance between RDF graphs readily induces a notion of *local sensitivity* (*at distance* k) $\mathsf{LS}_Q(\mathcal{G})$ ($\mathsf{LS}_Q^{(k)}(\mathcal{G})$) of SPARQL query Q over RDF graph $\mathcal{G}$, as given by Definition 2.

In order not to clutter the presentation, we usually omit the underlying dp-schema when referring to the size of an RDF graph, the distance between a pair of RDF graphs, and the local sensitivity of a SPARQL query if the dp-schema is understood from the context.

5.2. Elastic sensitivity

Our next step is, given a user query Q and an RDF graph $\mathcal{G}$ that complies with a dp-schema $\mathcal{P}$, to compute an upper bound of the local sensitivity $\mathsf{LS}_Q^{(k)}(\mathcal{G})$ (denoted by $\mathcal{U}_Q^{(k)}(\mathcal{G})$ in Lemma 1).

To this end, observe that the naive approach of evaluating the query on every neighbor (at distance k) of $\mathcal{G}$ is not a feasible solution, since the number of neighbors can be extremely large. To address this problem in the relational setting, Johnson *et al.* [11] has introduced the notion of *elastic sensitivity*, which leverages (maximum) frequency values of the join keys (that can be precomputed or statistically estimated) to provide more efficient upper bounds for the local sensitivity of queries with joins.

In the remainder of the section we adapt Johnson *et al.*'s approach to the case of RDF graphs and SPARQL. Intuitively, our notion of elastic sensitivity of a SPARQL query Q at distance k of a concrete graph $\mathcal{G}$ (that complies with the dp-schema $\mathcal{P}$) regards the evaluation of Q as the composition of successive transformations applied to $\mathcal{G}$, and is defined in terms of the stability properties of such transformations.

In our case, these transformations are given by the CBGP of user queries, more concretely, by their BGP part. We thus introduce the auxiliary notion of BGP *elastic stability*. A key property of this notion is that it allows bounding the local sensitivity of counting queries: Given a BGP B, its elastic stability at distance k with respect to a graph $\mathcal{G}$ that complies with a dp-schema $\mathcal{P}$, bounds the local stability for any graph $\mathcal{G}'$ that also complies with $\mathcal{P}$ and is at distance k of $\mathcal{G}$. Hence, it bounds the local sensitivity at distance k of $\mathsf{COUNT}^{?x}(\bar{B})$ (for any $\bar{B} = \langle B, F\rangle$) over $\mathcal{G}'$.

The formal definition of elastic stability relies on the frequency of most popular values. More precisely, if $?x$ is a variable occurring in an elementary BGP B, we use $mpv(?x, B, \mathcal{G})$ to denote the frequency of the most popular value to which $?x$ is mapped to, when evaluating B over $\mathcal{G}$. We can use SPARQL itself to determine $mpv(?x, B, \mathcal{G})$ through the query

$$\mathsf{SELECT}\ (\mathsf{COUNT}(?x)\ as\ ?c)\ \mathsf{WHERE}\ B$$
$$\mathsf{GROUP\_BY}\ ?x\ \mathsf{ORDER\_BY}\ ?c\ \mathsf{DESC\ LIMIT}\ 1$$

Loosely speaking, this corresponds to first evaluating $\mathsf{COUNT}^{?x}(B)$, and then selecting the value with the

largest count. This is an upper bound for the frequency of the most popular value yielded by $?x$ within a CBGP query $\bar{B} = \langle B, F \rangle$, regardless of the filter F, since it can only reduce the size of the result. Alternatively, we could also use the query

$$\text{SELECT (COUNT}(?x)\ as\ ?c\text{) WHERE } tp \\ \text{GROUP\_BY } ?x \text{ ORDER\_BY } ?c \text{ DESC LIMIT } 1\ ,$$

where tp is obtained from a triple pattern in B that 1) has $?x$ as one of its non-predicate (*i.e.* subject or object) components; 2) $?x$ is participating in a join of the full query $\bar{B}$; and 3) it has the other non-predicate component replaced by a fresh variable.[5]

The counting on the latter query will be greater than (or equal to) the one on the former query and, therefore, a valid (possibly looser, though) upper bound for $mpv(?x, B, \mathcal{G})$. Nevertheless, the benefit is that since the second query is merely a variable renaming of a $tp' \in S$, the values can be pre-computed for all $tp' \in S$ within a dp-schema, and simply retrieved during (differentially private) query evaluation. This is possible because the dp-schema of an RDF graph is defined with a set of predicates that includes all the predicates appearing the graph. In contrast, if the former query is used to determine *mpv*, the sensitivity is likely to be more accurate (*i.e.* tighter approximations) resulting in better privacy guarantees (smaller ϵ's), but precomputations cannot be done, possibly impacting system performance.

To compute the the elastic stability of BGP B at distance k of graph $\mathcal{G}$, written $\mathcal{S}_B^{(k)}(\mathcal{G})$, we start by applying Lemma 3 to decompose B as a sequence of elementary BGPs. Once we fix a normal ordering, we have $B_{\div\mathcal{P}} = B_1 \bowtie (B_2 \bowtie (\ldots \bowtie B_n)\ldots)$. This decomposition allows estimating the frequency of most popular values of graphs k far apart from $\mathcal{G}$. Formally, the frequency of the most popular values for variable $?x$ in a BGP B for graphs at distance k of $\mathcal{G}$, written $mpv^{(k)}(?x, B, \mathcal{G})$, is defined by induction on the length $|B_{\div\mathcal{P}}|$ as follows:

Base case: If $|B_{\div\mathcal{P}}| = 1$, we let

$$mpv^{(k)}(?x, B, \mathcal{G}) \ = \ mpv(?x, B, \mathcal{G}) + k \times \kappa(S),$$

where S is the covering star of B_1 (recall that, in this base case, $B_{\div\mathcal{P}} = \{B_1\}$).

Inductive case: If $|B_{\div\mathcal{P}}| > 1$, we let

$$mpv^{(k)}(?x, B, \mathcal{G}) \ = \ mpv^{(k)}(?x_1, B_{\div\mathcal{P}} \setminus \{B_1\}, \mathcal{G}) \\ \times\ mpv^{(k)}(?x, B_1, \mathcal{G})\ ,$$

where $?x_1$ is the common variable shared by B_1 and $(B_2 \bowtie (\ldots \bowtie B_n)\ldots)$, used for their equi-join.

The intuition behind the base case is easy to grasp. For $k = 1$, we take a subgraph from $\mathcal{P}(\!|\mathcal{G}|\!)$, induced by a star BFP S in the schema, and replace it with a different one. The maximal difference between the new and the old value on the count of the most popular mapping value of $?x$ is $\kappa(S)$. This is an upper bound of all the mappings that can be produced by the instance due to the triple-pattern multiplicity, if the instance removed didn't have an instance of the value and a new instance of the same value is added. Hence, k changes will at most increment the count by $k \times \kappa(S)$. For the inductive case, we need to worry about the most frequent mapping value of $?x_1$ in $B_{\div\mathcal{P}} \setminus \{B_1\}$ since for every mapping of $?x$ obtained from B_1, if this mapping maps $?x_1$ in B_1 to the same value of the most frequent value of x_1 in $B_{\div\mathcal{P}} \setminus \{B_1\}$, the value of $?x$ will be repeated as many times in the combined mapping of B. Hence, the most frequent value of $?x$ in $[\![B_1]\!]_{S_1}$ can be duplicated, in the worst case, as many as $mpv^{(k)}(?x_1, B_{\div\mathcal{P}} \setminus \{B_1\}, \mathcal{G})$ times in the multiset of solution mappings $[\![B]\!]_{\mathcal{G}}$, giving us a safe upper bound for the count. Importantly, observe that because the multiplication operation is commutative, the frequency is not affected by the selected normal ordering.

We are now ready to define the elastic stability of a BGP B at distance k of graph $\mathcal{G}$, denoted by $\mathcal{S}_B^{(k)}(\mathcal{G})$. The definition also proceeds by induction on the cardinality of a fixed normal ordering of $B_{\div\mathcal{P}} = B_1 \bowtie (B_2 \bowtie (\ldots \bowtie B_n)\ldots)$:

Base case: If $|B_{\div\mathcal{P}}| = 1$, we let

$$\mathcal{S}_B^{(k)}(\mathcal{G}) \ = \ \kappa(S),$$

where S is the covering star of B_1.

Inductive case: If $|B_{\div\mathcal{P}}| > 1$, 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 $B_j \in B'$, we let

[5] Observe that there might be multiple such triple patterns in B, all of them yielding valid upper bounds for $mpv(?x, B, \mathcal{G})$. If we are interested in obtaining tighter privacy guarantees, we should choose the most precise bound.

$$\mathcal{S}_B^{(k)}(\mathcal{G}) \;=\; \max\Big\{ mpv^{(k)}(?x_1, B_1, \mathcal{G}) \times \mathcal{S}_{B'}^{(k)}(\mathcal{G}), \\ mpv^{(k)}(?x_1, B', \mathcal{G}) \times \mathcal{S}_{B_1}^{(k)}(\mathcal{G}) \Big\} \quad (2)$$

If the covering star S of B_1 is also the covering star of another $B_j \in B'$, we let

$$\mathcal{S}_B^{(k)}(\mathcal{G}) \;=\; mpv^{(k)}(?x_1, B_1, \mathcal{G}) \times \mathcal{S}_{B'}^{(k)}(\mathcal{G}) + \\ mpv^{(k)}(?x_1, B', \mathcal{G}) \times \mathcal{S}_{B_1}^{(k)}(\mathcal{G}) + \\ \mathcal{S}_{B_1}^{(k)}(\mathcal{G}) \times \mathcal{S}_{B'}^{(k)}(\mathcal{G})$$

Loosely speaking, this definition captures the amount of changes that the transformations (i.e. the joins) within the query, add to the final result, when modifying a single element in the induced schema.

As so defined, the local stability bounds the local sensitivity of counting queries:

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

$$\mathcal{S}_B^{(k)}(\mathcal{G}) \;\geqslant\; \mathsf{LS}^{(k)}_{\mathsf{COUNT}^{?x}(\bar{B})}(\mathcal{G})\,.$$

The main intuition behind the proof is that changes made to a graph to get a new graph at distance 1, are limited to a sub-graph $\mathcal{G}_i$, that must be covered by a single star pattern S in $\mathcal{P}$. Then the maximum number of RDF tuples that can change to get the graph at distance 1 is limited by the multiplicity of the predicates in S. Therefore, the change in the number of mappings obtained from the new graph of an elementary BGP covered by S is bounded by $\kappa(S)$. If these RDF triples contribute in the result mappings of a join vertex, the number of new mappings can increase by as much as the frequency of the most popular result mapping of the joining triple pattern. For example, if $B = \{(?\mathsf{v}_0, \mathsf{p}, ?\mathsf{u}), (?\mathsf{u}, \mathsf{p}', ?\mathsf{v}_1)\}$, and the triple $(\mathsf{s}, \mathsf{p}, \mathsf{o})$ is part of $\mathcal{G}_1$ and o happens to be the most popular result mapping for $(?\mathsf{u}, \mathsf{p}', ?\mathsf{v}_1)$, then there will be at most $mpv^{(1)}(?\mathsf{u}, (?\mathsf{u}, \mathsf{p}', ?\mathsf{v}_1), \mathcal{G})$ new mappings in the result.

Proof. The proof of this lemma follows the same strategy as the proof in [11, Lemma 2], and is by induction on the length of $B_{\div\mathcal{P}}$.

- **Case** $|B_{\div\mathcal{P}}| = 1$. Let S be the covering star of B. Hence, its elastic stability is $\kappa(S)$, a parameter given by the DBA. Thus, we have

$$\kappa(S) \;=\; \mathcal{S}_B^{(k)}(\mathcal{G}) \;\geqslant\; \mathsf{LS}^{(k)}_{\mathsf{COUNT}^{?x}(\bar{B})}(\mathcal{G})$$

  since the local sensitivity at distance k is calculated as the max of the sensitivities of all graphs at distance 1 of all graphs at distance k or less of $\mathcal{G}$, meaning the modification of a single star, which may change by at most $\kappa(S)$ tuples and the filter in $\bar{B}$ doesn't affect its local stability.
- **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 that can change in graphs $\mathcal{G}'$ at distance k of $\mathcal{G}$ to get a graph at distance 1, based on the star multiplicities. First, let's assume S_1 is not the covering star of any other B_i in B'. Hence, changes can happen in either $\mathcal{G}'_{S_1}$ or in a graph $\mathcal{G}'_{S'}$ induced by a star S' different from S_1 that covers some other other $B_i \in B'$, but not in both graphs since the new graph must be at distance 1 from $\mathcal{G}'$. Thus, either $\mathcal{S}_{B_1}^{(k)}(\mathcal{G}) = 0$ or $\mathcal{S}_{B'}^{(k)}(\mathcal{G}) = 0$:

  1. When $\mathcal{S}_{B_1}^{(k)}(\mathcal{G}) = 0$, the changes in $\mathcal{G}'_{S'}$, by induction hypothesis using Eq (2), produce at most $mpv^{(k)}(?x_1, B_1, \mathcal{G}) \times \mathcal{S}_{B'}^{(k)}(\mathcal{G})$ changed mappings since one change in $\mathcal{G}'_{S'}$ might affect at most $mpv^{(k)}(?x_1, B_1, \mathcal{G})$ triplets in the same join when applied to a graph at distance 1 of $\mathcal{G}'$.
  2. In the symmetric case, when $\mathcal{S}_{B'}^{(k)}(\mathcal{G}) = 0$, $\mathcal{G}'_{S_1}$ may contain $\mathcal{S}_{B_1}^{(k)}(\mathcal{G}) = \kappa(S)$ changed triplets, producing at most $mpv^{(k)}(?x_1, B', \mathcal{G}) \times \mathcal{S}_{B_1}^{(k)}(\mathcal{G})$ changed mappings in the joined SPARQL pattern.

  We chose the largest of the two values when calculating $\mathcal{S}_B^{(k)}(\mathcal{G})$. On the other hand, if S_1 also covers another $B_i \in B'$, a change in $\mathcal{G}'_{S_1}$ can also imply changes in $\mathcal{G}'_{S'}$. This can cause, in the worst case, $mpv^{(k)}(?x_1, B_1, \mathcal{G}) \times \mathcal{S}_{B'}^{(k)}(\mathcal{G})$ changed mappings for the change happening in $\mathcal{G}'_{S'}$, plus $mpv^{(k)}(?x_1, B', \mathcal{G}) \times \mathcal{S}_{B_1}^{(k)}(\mathcal{G})$ changed mappings caused by the change in $\mathcal{G}_{S_1}$, which may contain $\mathcal{S}_{B_1}^{(k)}(\mathcal{G}) = \kappa(S)$ changed triplets. We also need to consider that the change may cause new joins between then new triplets in both $\mathcal{G}'_{S_1}$ and $\mathcal{G}'_{S'}$, for a total of $\mathcal{S}_{B_1}^{(k)}(\mathcal{G}) \times \mathcal{S}_{B'}^{(k)}(\mathcal{G})$ changed mappings in the joined SPARQL pattern. The sum of these three values is what the definition of $\mathcal{S}_B^{(k)}(\mathcal{G})$ uses. □

Now we have all the prerequisite to define the *elastic sensitivity* of user queries (at fixed distances of a given graph):

$$\begin{aligned}
\mathsf{ES}^{(k)}_{\mathsf{COUNT}^{?x}(\bar{B})}(\mathcal{G}) &= \mathcal{S}^{(k)}_{B}(\mathcal{G}) \\
\mathsf{ES}^{(k)}_{\mathsf{COUNT}^{\mathsf{DISTINCT}\ ?x}(\bar{B})}(\mathcal{G}) &= \mathcal{S}^{(k)}_{B}(\mathcal{G}) \\
\mathsf{ES}^{(k)}_{\mathsf{COUNT}^{?x_0}_{?\bar{x}}(\bar{B})}(\mathcal{G}) &= 2\mathcal{S}^{(k)}_{B}(\mathcal{G})
\end{aligned}$$

And as in Johnson *et al.* [11], the above lemma readily leads us to the desired bound for (the three kind of) user queries:

Lemma 5. *For any user query Q, and any graph $\mathcal{G}$ compliant with schema $\mathcal{P}$:*

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

Proof. By case analysis on the type of user query Q:

- For plain counting queries ($Q = \mathsf{COUNT}^{?x}(\bar{B})$): 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.
- For plain unique counting queries ($Q = \mathsf{COUNT}^{\mathsf{DISTINCT}\ ?x}(\bar{B})$): it can be noted that DISTINCT reduces the elastic stability of the elementary BGP, B', containing $?x$ from $\kappa(S')$ to 1, where S' is the star covering B'.
- For counting queries after grouping ($Q = \mathsf{COUNT}^{?x_0}_{?\bar{x}}(\bar{B})$): During the grouping, each changed triple can affect *two* result mappings in the query since one modified triple may generate a mapping that will fall into a new group, and at the same time the old mapping is dropped from another group. □

Lemma 5 readily establishes our main result, which allows applying differential privacy to SPARQL queries over RDF graphs:

Theorem 3. *Assume that our universe of (valid)* RDF *graphs is composed by the graphs that comply with dp-schema $\mathcal{P}$ and multiplicity bound κ and let $\mathcal{G}$ be any of those graphs. Let Q be a user query and let*

$$\mathcal{U}_Q(\mathcal{G}) = \max_{0 \leqslant k \leqslant size(\mathcal{G})_{\mathcal{P}}} e^{-\beta k}\, \mathsf{ES}^{(k)}_{Q}(\mathcal{G}) \,,$$

where $\epsilon > 0$, $0 < \delta < 1$, $\beta \leqslant \frac{\epsilon}{2\ln(2/\delta)}$ and the elastic sensitivity $\mathsf{ES}^{(k)}_{Q}(\mathcal{G})$ of Q is computed, as previously described, from multiplicity bound κ and the frequencies of most popular values mapped to the variables in Q as specified by function $mpv^{(k)}$. Then, the randomized algorithm

$$\mathcal{A}(\mathcal{G}) = [\![Q]\!]_{\mathcal{G}} + \mathsf{Lap}\Big(\frac{2\,\mathcal{U}_Q(\mathcal{G})}{\epsilon}\Big)$$

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

The theorem follows immediately from Theorem 2 and Lemmas 1 and 5.

6. 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.

Setup We conducted our evaluation on a 2018 Macbook Pro with 16 GB of RAM memory having installed a Fuseki instance on a 2 AMD Opteron server with an SSD drive and 64GB of RAM memory. We used Java 1.17 to implement our proof of concept. We also 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[6], an essential component for ensuring the correctness of our privacy guarantees algorithm. The code and all the queries used for this evaluation are available in GitHub [7].

Data In the evaluation we used real world data and queries from Wikidata [13]. 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[8]). We selected a subset of the Wikidata Truthy from 2021-06-23, which has all but direct properties (i.e http://www.wikidata.org/prop/direct/P*) removed [24, 25]. The data is available to download from Google

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

[7] Repository https://github.com/cbuil/DPSparql

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

Drive[9]. We also provide the scripts to generate this Wikidata version in our Github repository[10]. We use the following prefixes from Wikidata along this section:

```
wdt: <http://www.wikidata.org/prop/
    direct> # prefix for referring to
    properties
wd: <http://www.wikidata.org/entity> #
    prefix for referring to data
```

6.1. Privacy schema

Our dp-schema is defined based on three pairwise disjoint stars, $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, representing data from Humans, Organizations, and Professions, built around Wikidata's `P31` property (the *instance of* property). We extracted URIs from all the instances of the classes `Human`, `Organization` and `Profession` using queries like:

```
SELECT ?center WHERE {
  ?center wdt:P31 wd:Q5
# wd:Q5 represents the Human class
}
```

This gathers the instances of the class `Human`. Star instances were formed by selecting a subset of properties of the three classes using star queries centered in the URIs (each instance representing either a Human, an Organization or a Profession) to define three subgraphs covered by three stars, $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$. In other words, we used the mappings of `?center` from the initial queries as star centers and for each center we retrieved a few of their properties and used the resulting RDF graph as the basis for our queries. Differential privacy is applied to protect the privacy of the centers. We present the schema with all the properties we used for each star in the following example.

Evaluation Privacy Schema The three stars employed to identify the different entities in our evaluation schema are (modulo variable renaming):

$$
\begin{aligned}
S_1 &= \{(?\mathsf{c}_1, \mathsf{P569}, ?\mathsf{o}_1), (?\mathsf{c}_1, \mathsf{P570}, ?\mathsf{o}_2), (?\mathsf{c}_1, \mathsf{P106}, ?\mathsf{c}_3)\} \\
&\quad (?\mathsf{c}_1, \mathsf{P108}, ?\mathsf{c}_3), (?\mathsf{c}_1, \mathsf{P2002}, ?\mathsf{o}_2), \ (?\mathsf{c}_1, \mathsf{P21}, ?\mathsf{o}_3)\} \\
&\quad (?\mathsf{c}_1, \mathsf{P40}, ?\mathsf{c}_3)\} \\
S_2 &= \{(?\mathsf{c}_3, \mathsf{P1963}, ?\mathsf{o}_5), (?\mathsf{c}_3, \mathsf{P101}, ?\mathsf{o}_6), (?\mathsf{c}_3, \mathsf{P425}, ?\mathsf{o}_6)\} \\
S_3 &= \{(?\mathsf{c}_2, \mathsf{P106}, ?\mathsf{o}_3), (?\mathsf{c}_2, \mathsf{P178}, ?\mathsf{o}_4), (?\mathsf{c}_2, \mathsf{P112}, ?\mathsf{o}_5)\}
\end{aligned}
$$

These three stars shouldn't be used directly as a privacy schema because S_1 and S_3 share a property, `P106`. Nevertheless, the sets of instances of the property in the sub-graphs induced by S_1 and S_3 are disjoint because instances of $?c_1$ (human UIRs) and $?c_2$ (organization URIs) are disjoint. Hence, for the purpose of our evaluation, we consider this partition of `P106` as two different properties that we refer to as `P106 Profession` in S_1 and `P106 Occupation` in S_3, keep S_3 as it is, and rename them $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ respectively.[11]

The Wikidata properties that allow joining data from two different stars are `P108 Employer` from Humans to Organizations, `P106 Profession` from Humans to Professions, `P106 Occupation` from Organizations to Professions and `P112 Founded by` from Organizations to Humans. Table 1 shows statistics about the instances of the stars in our data, including star size (on top of the table).

6.2. Queries

We selected 26 queries from the query logs in [24, 25] containing the predicates used in our dp-schema. Since the amount of COUNT queries in the query logs is small [26] for each of these queries we added the `COUNT` keyword and we removed the triples from the query that were not accessing our schema. In addition, these queries were modified to get a diverse set of query results and types.

We consider star queries which are queries covered by a single star from the dp-schema. These queries can only add filters and remove triple patterns from the star. Therefore, a star query is centered around a single join vertex $?x_0$, corresponding to the center of the star (Figure 2). We also have linear queries describing a path that must include a join variable that appears in a place that is not the center of any star from

[9] https://drive.google.com/u/0/uc?id=1oDkrHT68_v7wfzTxjaRg40F7itb7tVEZ

[10] https://github.com/MillenniumDB/benchmark/blob/master/src/database_generation/filter_direct_properties.py

[11] Note that this observation suggests a more subtle definition of privacy scheme would be useful.

Humans: 9,181,487				
P569	P570	P106	P108	P2002
5,109,648	2,511,719	6,446,811	1,085,617	159,194
P21	P40			
7,223,891	707,747			

Professions: 7,786		
P1963	P101	P425
97	95	3,018

Organizations: 72,879		
P106	P178	P112
50	16	2,789

Table 1

Table showing key statistics about the data in our privacy schema (the largest schema is by far the Humans star)

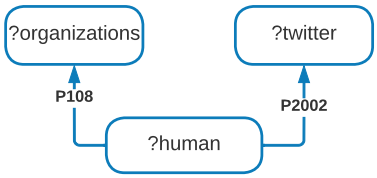

Figure 2. Star-shaped query Q_{16} accessing the Humans privacy star

the dp-schema (Figure 3), and snowflake queries (Figure 4), a concatenation through a join variable of a star query with other queries of different shapes, as defined in [21]. The queries are listed in Appendix B and they can also be found in the companion Github repository for this article. Table 3, also in Appendix A, summarizes their characteristics. We show Q_3 in Figure 4, which queries the Humans star, accessing sex, birth and death dates for each human as well as their professions. We use the `?professions` variable for connecting to the Professions star. From that star we retrieve each profession's field of work (property `P425`), obtaining a snowflake-shaped query.

6.3. Results

We report the results of our evaluation in Table 2, showing the actual count output by the queries and the average result to these queries with added noise (calculated applying the method described in Section 5). We followed the query schema introduced in Section 5.2, that uses the BGP part of each query to calculate the initial most popular values (*mpv*). We report the results using two values for ϵ, 0.1 and 1.0. We also report the median error percentage for $\epsilon = 1.0$ in Figure 5 between the real counts vs. the counts with added noise segregated by the type of query. To calculate the error we used the following Equation:

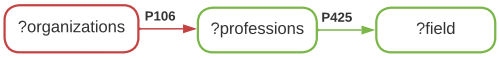

Figure 3. Linear query Q_{15} accessing the Humans privacy star

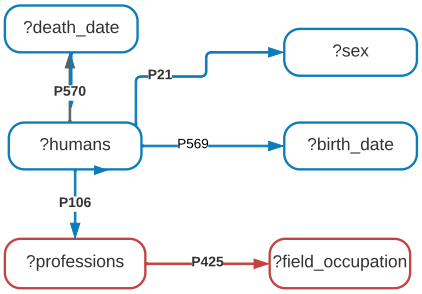

Figure 4. Snowflake query Q_3 accessing the Humans and Organization privacy schemas

$$median\left(\frac{(ActualCount - NoiseResult_i) * 100}{ActualCount}_{i=1...100}\right)$$

We use $\delta = n^{-\epsilon \ln n}$ where n is the size of the dataset (i.e. the sum of all the different RDF resources in the schema that the queries access) and $\beta = \epsilon/(2 \times \log(2/\delta))$ for the β parameter.

(Blue) squares in Figure 5 represent star queries (typically accessing a single star within the dp-schema), have a very low error (and thus a high utility) compared to the other two types of queries that involved at least one "join" operation across stars.

However, the smaller the result from the `COUNT`, the larger the error introduced. (Red) triangles represent path queries and thus queries with join operations. Only query Q_9 is close to the 10% error threshold to be considered a query with enough utility. That query is only two triple patterns that retrieve all data from the Professions and Organizations stars. (Grey) circles represent Snowflake queries, which are queries

that join two or more stars in the schema, and also access several properties from each star pattern. Only those queries returning a result greater than 1,000,000 (queries $Q_{3,6}$) have a high utility result.

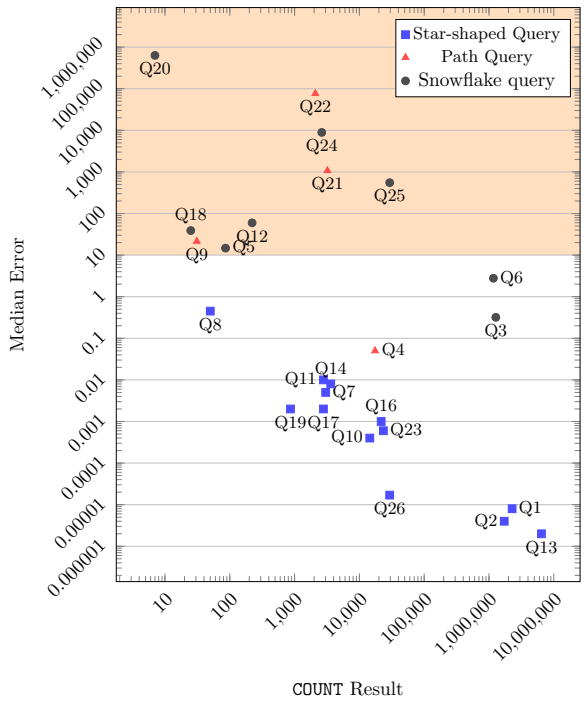

Figure 5. This plot shows that star-shaped queries (blue squares) have the greatest utility, since they are likely not to have joins between stars in the schema. It also shows that queries accessing large amounts of data have high utility. Notice that $Q15$ does not appear in the plot since its result is 0.

The *sensitivity* and *stability* columns in Table 2 clearly show that the larger the degree in the stability polynomial the greater the query sensitivity, and thus, the higher the error in the result. Note that in most cases the derived queries produce smaller results than the simpler queries, thus the errors are larger. The only queries where this is not the case are queries $Q_{1,2}$. However, the errors in these queries are so small that much more data should be collected to really establish if they are statistically different. A class of queries with large errors are queries with small outputs and at least one join. See, for example, queries $Q_{9,18,20}$. In general, the more joins in the query and the smaller the result size, the worse results. Joins directly affect the stability polynomial, more joins imply larger degree. The effect is exacerbated if the value representing the most popular mapping is large. Large amounts of noise are introduced to results of the three queries associated with polynomials of degree 2, $Q_{20,21,22}$, which access data from 3 different stars. Even though the results of the original queries are not small (>2,000), the error introduced is very high. Compare that to the single join query Q_9 that produces a small result and high errors, but it is much smaller than the errors of Q_{21} and Q_{22}. Results from queries accessing a single star from the dp-schema are good as expected, since without joins they have low query sensitivity and, hence, small errors.

7. Related Work

The study of how to guarantee the privacy of individuals contributing personal data to datasets 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 guarantee privacy over the results returned by SPARQL queries. We briefly look over these works in this section.

7.1. Privacy over SPARQL

There have been several approaches to address privacy concerns related queries to RDF data. A good survey can be found in [27]. There is a basic anonymization protection that a SPARQL engine must provide to queries that directly return individuals, as opposed to aggregated data. Similar to the case of relational databases where attribute values are anonymized using nulls, the work presented in [28] uses blank nodes to hide sensitive data. Delanoux et al. [29] introduce a more general framework with formal soundness guarantees for privacy policies that describe information that should be hidden as well as utility policies that describe information that should be available. The framework checks whether policies are compatible with each other, and based on a set of basic update queries that use blank nodes and deletions of triples, automatically derives from the policies candidate sets of anonymization operations that guarantee to transform any input dataset into a dataset satisfying the required policies. However, their soundness guarantees do not imply any formal privacy guarantees. Two early methods developed for privacy protection when answering queries about classes in a dataset are k-anonymity and l-diversity. In particular, k-anonymity

Query Id	Actual Result	Average Private Result using Epsilon = 0.1	Average Private Result using Epsilon = 1.0	Sensitivity	Stability $S^{(k)}(Q_i, ?x)$
Q1	2,275,177	2,275,176	2,275,176	1.0	1
Q2	1,717,945	1,717,940	1,717,945	1.0	1
Q3	1,274,788	1,189,636	1,270,649	290,415	(x+290,863) * 1
Q4	17,440	17,464	17,319	245	(x+18) * 1
Q5	86	110.8	203.1	245	(x+18) * 1
Q6	1,170,315	3,860,469	1,137,687	400,363	(x+400,981) * 1
Q7	3,018	3019	3,017	1.0	1
Q8	50	57	50	1.0	1
Q9	31	1577	37	241.3	(x+8) * 1
Q10	14,477	14,472	14,477	1.0	1
Q11	2,789	2,792	2,789	1.0	1
Q12	221	1,144	89	1,162.2	(x+1,164) * 1
Q13	6,446,811	6,446,812	6,446,810	1.0	1
Q14	3,615	3,613	3,614	1.0	1
Q15	0	71	8	250.8	(x+33) * 1
Q16	21,683	21,682	21,682	1.0	1
Q17	2,789	2,788	2,788	1.0	1
Q18	25	465	34	248.5	(x+27) * 1
Q19	865	864	865	1.0	1
Q20	7	7,087,181	44,254	1,656,501	(x+6,189) * (x+8) * 1
Q21	3,213	1,739,549	31,357	1,651,883	(x+6) * (x+6,189) * 1
Q22	2,092	377,638,493	1,600,610	408,594,207	(x+7) * (x+1,694,747) * 1
Q23	23,450	23,449	23,449	1.0	1
Q24	2,626	1,071,418	231,278	628,018	(x+628,987) * 1
Q25	29,352	1,593,488	42,317	628,018	(x+628,987) * 1
Q26	29,352	29,350	29,351	1.0	1

Table 2

Results of the execution of Wikidata queries using our differential privacy method. Those queries with sensitivity "1.0" are star queries since the sensitivity of a COUNT query over a single star schema is 1 (a COUNT query over a table), and their elastic stability is "x" as described in Section 5.2 If there are joins between star BGPs, the sensitivity increases based on the stability polynomial, calculated according to Theorem 3.

is used in [30, 31] to answer queries in RDF datasets. Unfortunately, it is well-known these methods, in contrast to differential privacy, do not provide formal guarantees for privacy.

The only work known to us that directly applies differential privacy to SPARQL queries is [27]. But surprisingly, differential privacy is realized through local sensitivity alone without the use of a smoothing function necessary for correctness [8]. A privacy-preserving query language for RDF streams is introduced in [32]. Limiting queries to that language servers can continuously release privacy-preserving histograms (or distributions) from online streams. Han et al. [33] provide differentially-private variants of the algorithms TransE and RESCAL, aimed at constructing knowledge graph embeddings in the form of real-valued vectors. While the authors show that these encodings allow performing some analyses with a reasonable privacy-utility tradeoff, inlcuding clustering and link prediction, it is an open question whether this generalizes to further analyses or counting queries as addressed in the current article.

7.2. Privacy in Social Graphs

A central task to the development of any practical differentially private analysis tool is finding appropriate approximations and alternatives to global sensitivity: it should be easy to calculate, and, at the same time, close enough to the real sensitivity to allow the computation of statistically useful results. A well-known approach is to rely on the concept of restricted sensitivity [9]. Restricted sensitivity is tailored to provide privacy guarantees assuming datasets come from a specific subgroup of the universe of all possible datasets, and it was introduced in the context of social-graph data analysis. There are two natural notions from which one can define adjacency of graphs: differences on edges and differences on vertices. The distance between two graphs, $\mathcal{G}_1$ and $\mathcal{G}_2$, can be then given by the smallest number of changes (either on edges or vertices) needed to transform $\mathcal{G}_1$ and $\mathcal{G}_2$ into the same graph, giving rise to two definitions of restricted sensitivity. Blocki et al. [9] provide efficient algorithms to calculate approximations of these sensitivities for a class of social graph queries that involve only one type of join: aggregations over properties of a node and its neighbors (the specific subgroup of interest). Proserpio, Goldberg and McSherry extended the edge-based definition of restricted sensitivity to include weighted datasets [34]. Briefly, the intent was to increase the utility of the answer by considering weights associated with edges during the calculation of noise. Our notion of dp-schema sensitivity can be seen as a vertex-based sensitivity if each induced subgraph $g_i \in \mathcal{P}(\!|\mathcal{G}|\!)$ is interpreted as a single node with its attributes. Our proposed elastic sensitivity can be then interpreted as a generalization to handle multiple joins. The polynomial to calculate the selectivity of a query with a single join is always of degree 1. Furthermore, social graphs are essentially defined using a single relationship type. Using our terminology, this implies that the dp-schema would consist of a single star BGP. Hence, Blocki et al. [9] argue that, in practice, the value of x in the degree-1 polynomial (*i.e.* the frequency of the most popular value) can be bounded by a constant (which would be provided by the RDF data administrator). This is closely akin to our predicate multiplicity. Elastic sensitivity, on the other hand, uses a variable selectivity (the values of x are obtained directly from the dataset), and generalizes to multiple joins. Other works such as [35] proposed differential privacy methods 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.

8. Conclusions

In this paper we have introduced a framework to develop differential privacy tools for RDF data repositories. 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 or dp-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. In our algorithm, we adapt the concept of elastic sensitivity of SQL queires from [11] to SPARQL.

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. 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. As in the case of SQL queries in relational databases, in order to keep the

noise in SPARQL queries under a single percentage digit, query results should have over 1M tuples and $\epsilon = 1$. The evaluation shows though that we can safely evaluate star queries.

There are many pending issues to address. 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. From the more practical point of view, more operations need to be implemented. We can consider the approaches described in [11] for SQL to add aggregation functions like sum and averages to our framework. There are also issues to consider 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 a dp-schema. It might also be possible to find a more accurate approximation for the elastic sensitivity of $DISTINCT$ queries to make them independent or partially independent of predicate multiplicity. We should point out that most queries that will require privacy protection will be $DISTINCT$ (e.g. how many human exists with properties A and B that work in company C?).

Acknowledgements

We thank the reviewers for their thorough work revising this paper, which improved the overall quality of the paper. C. Buil-Aranda and J. Lobo were partially funded by Fondecyt Iniciación 11170714; C Buil-Aranda and Federico Olmedo were supported by ANID – Millennium Science Initiative Program – Code ICN17_002.

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] N. Li, T. Li and S. Venkatasubramanian, t-closeness: Privacy beyond k-anonymity and l-diversity, in: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, IEEE, 2007, pp. 106–115.

[7] C. Dwork, Differential privacy: A survey of results, in: *International Conference on Theory and Applications of Models of Computation*, Springer, 2008, pp. 1–19.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] R. Motwani and P. Raghavan, Randomized algorithms, *ACM Computing Surveys (CSUR)* **28**(1) (1996), 33–37.

[15] 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.

[16] 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/.

[17] 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.

[18] 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.

[19] C. Gutierrez, C. Hurtado and A.O. Mendelzon, Foundations of semantic web databases, in: *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2004, pp. 95–106.

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

[21] 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.

[22] M. Arapinis, D. Figueira and M. Gaboardi, Sensitivity of Counting Queries, in: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani and D. Sangiorgi, eds, LIPIcs, Vol. 55, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 120–112013.

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

[24] A. Hogan, C. Riveros, C. Rojas and A. Soto, A Worst-Case Optimal Join Algorithm for SPARQL, in: *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I*, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I.F. Cruz, A. Hogan, J. Song, M. Lefrançois and F. Gandon, eds, Lecture Notes in Computer Science, Vol. 11778, Springer, 2019, pp. 258–275. doi:10.1007/978-3-030-30793-6_15. https://doi.org/10.1007/978-3-030-30793-6_15.

[25] D. Vrgoc, C. Rojas, R. Angles, M. Arenas, D. Arroyuelo, C.B. Aranda, A. Hogan, G. Navarro, C. Riveros and J. Romero, MillenniumDB: A Persistent, Open-Source, Graph Database, *CoRR* **abs/2111.01540** (2021). https://arxiv.org/abs/2111.01540.

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

[27] 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.

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

[29] 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.

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

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

[32] 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.

[33] X. Han, D. Dell'Aglio, T. Grubenmann, R. Cheng and A. Bernstein, A framework for differentially-private knowledge graph embeddings, *J. Web Semant.* **72** (2022), 100696. doi:10.1016/j.websem.2021.100696. https://doi.org/10.1016/j.websem.2021.100696.

[34] 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.

[35] 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.

Appendix A. Query Characteristics

In this Section we present the characteristics of the queries we used in Section 6. The table presents the query ID (which refer to queries Section B in this Appendix), the stars within the Privacy Schema we defined in 6, the query shape, the amount of tripe patterns in the queries as well as the number of variables, the join variables when applicable, including the amount of mappings for each join variable, and data about the Privacy Schema stars in the query.

Query ID	Schemes	Shape	Triple Patterns	Join Variables	Star Info
Q1	Humans	Star	3 tp, 4 vars	No join variables	1,717,255 Humans with Professions, 10,337 Professions at Humans star, 2,991 distinct Professions at Professions star
Q2	Humans	Star	4 tp, 5 vars	No join variables	
Q3	Humans and profs	Snowflake	5 tp, 6 vars	COUNT(?professions) = 3018 (from profs star)	
Q4	Humans	Path	1 tp, 2 vars	COUNT(?humans) = 3615 (from Professions star) COUNT(?humans) = 1,648,629 (from Humans Star)	1,648,629 distinct humans educated at (P69), and 3,615 Humans have Organizations (Q_2b), 2,789 professions (Q_2c 6 developer field)
Q5	Humans and Orgs	Snowflake	2 tp, 3 vars	COUNT(?humans) = 3,615 from Professions Star	
Q6	Humans and Profs	Snowflake	3 tp, 4 vars	COUNT(?professions) = 7,764 from Humans, COUNT(?professions) = 3,018 from professions	
Q7	Profs	Star	1 triple, 2 vars	No join variables	3018 distinct profs 50 distinct organizations with professions, 3018 distinct professions
Q8	Orgs	Star	1 triple, 2 vars	No join variables	
Q9	Orgs and Profs	Path	2 tp 4 vars	COUNT(?professions) = 54 from organizations, ?professions = 3,356 Professions	
Q10	Humans	Star	1 triple 2 vars	No join variables	14,382 distinct Humans, 2,789 distinct Organizations
Q11	Orgs	Star	1 triple 2 vars	No join variables	

Q12	Humans and Orgs	Snowflake	3 tp 6 vars	COUNT(?organizations) = 33,423 from humans star, COUNT(?organizations) = 3,814 form Organizations star	
Q13	Humans	Star	1 triple 2 vars	No join variables	6,446,811 distinct Humans, 2,789 distinct Organizations
Q14	Orgs	Star	1 triple 2 vars	No join variables	
Q15	Humans and Orgs	Path	2 tp 4 vars	COUNT(?organizations) = 8,501,245 at Humans star, COUNT(?organizations) = 3,814 at Organizations star	
Q16	Humans	Star	2 tp 4 vars	No join variables	21,651 Humans with Twitter accounts and a employer, 13,814 distinct Organizations
Q17	Orgs	Star	1 triple 2 vars	No join variables	
Q18	Humans and Orgs	Snowflake	3 tp 4 vars	COUNT(?organizations) = 35,419 from Humans Star, COUNT(?humans=3814) from Organizations star	
Q19	Organizations	Star	2 tp	No join variables	5 distinct organizations
Q20	Profs, Humans and Orgs	Snowflake	3 tp 6 vars	No join variables	3,018 distinct profs, 90,341 Organizations at Humans star, 31 distinct Organizations at Organizations star
Q21	Humans, Occupations, Professions	Path	3 tp, 4 vars	COUNT(?occupationsStar) = 90,341 (humans side), COUNT(?opccupationsStar) = 15,524 (occupationsStar side), 3,018 professions (Professions side)	6,446,860 occupationsStar, 3,018 Professions, 1,448,232 Humans with occupations at Humans Star

Q22	Organizations, Occupations, Professions	Path	3 tp, 4 vars	COUNT(?occupationsStar) = 3,615 (organizations side), 15524 (occupationsStar side), 3018 professions (professions side)	6,446,860 occupationsStar, 2,789 Organizations Star, 3,018 Professions Star
Q23	Humans	Star	2 tp, 3 vars		23,450 Humans with Athlete profession.
Q24	Humans and Profs	Snowflake	2 tp, 3 vars	?COUNT(?professions) = 23,451 form Humans Star, ?COUNT(?professions) = 4 form Professions Star,	
Q25	Profs, Humans	Snowflake	5 tp, 3 different vars	?COUNT(?professions) = 15,359	1 profession, 4,969,877 humans
Q26	Humans	Star	2 tp, 2 vars	?COUNT(?humans) = 29,352 humans (distinct)	

Table 3: table showing the main characteristics of each query to the privacy schema

Appendix B. Queries

In this Section we present the queries we used in Section 6. There are 11 base queries with several variations, totaling 26 SPARQL COUNT queries.

Query Q_1:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
?humans wdt:P21 ?v1 .
?humans wdt:P569 ?v4 .
?humans wdt:P570 ?v2 .
}
```

Query Q_2:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) where {
?humans wdt:P21 ?v1 .
?humans wdt:P569 ?v4 .
?humans wdt:P570 ?v2 .
?humans wdt:P106 ?occupation .
}
```

Query Q_3:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) where {
  ?humans wdt:P31 wd:Q5 .
  ?humans wdt:P21 ?v1 .
  ?humans wdt:P569 ?v4 .
  ?humans wdt:P570 ?v2 .
  ?humans wdt:P106 ?professions .
  ?professions wdt:P31 wd:Q28640 .
  ?professions wdt:P425 ?
      field_occupation
}
```

Query Q_4:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
    ?humans wdt:P69 ?v2 .
    ?organizations wdt:P112 ?humans .
}
```

Query Q_5:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
    ?humans wdt:P69 ?v2 .
    ?organizations wdt:P178 ?developer
        .
    ?organizations wdt:P112 ?humans
}
```

Query Q_6:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
    ?humans wdt:P69 ?v2 .
    ?humans wdt:P106 ?professions .
    ?professions wdt:P425 ?field
}
```

Query Q_7:

```
SELECT (COUNT(DISTINCT ?professions) as
    ?count) WHERE {
  ?professions wdt:P425 ?var6 .
}
```

Query Q_8:

```
SELECT (COUNT(DISTINCT ?organizations)
    as ?count) WHERE {
  ?organizations wdt:P106 ?professions
      .
}
```

Query Q_9:

```
SELECT (COUNT(DISTINCT ?organizations)
    as ?count) WHERE {
  ?professions wdt:P425 ?var6 .
  ?organizations wdt:P106 ?professions
      .
}
```

Query Q_{10}:

```
Select (COUNT (DISTINCT ?humans) as ?
    count) where {
  ?humans wdt:P40 ?child .
  ?humans wdt:P108 ?organizations .
}
```

Query Q_{11}:

```
Select (COUNT (DISTINCT ?organizations)
    as ?count) where {
  ?organizations wdt:P112 ?founded_by
}
```

Query Q_{12}:

```
SELECT (COUNT (DISTINCT ?humans) as ?
    count) WHERE {
  ?humans wdt:P40 ?child . #with at
      least one P40 (child) statement
  ?humans wdt:P108 ?organizations .
  ?organizations wdt:P112 ?founded_by
}
```

Query Q_{13}:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) where {
?humans wdt:P106 ?organizations .
}
```

Query Q_{14}:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) where {
?organizations wdt:P112 ?humans
}
```

Query Q_{15}:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) where {
?humans wdt:P106 ?organizations .
?organizations wdt:P112 ?founded_by
}
```

Query Q_{16}:

```
SELECT (COUNT (DISTINCT ?humans) as ?
    count) WHERE {
  ?humans wdt:P2002 ?twittter . #with
      at least one P40 (child) statement
  ?humans wdt:P108 ?organizations .
}
```

Query Q_{17}:

```
SELECT (COUNT (DISTINCT ?organizations)
    as ?count) WHERE {
  ?organizations wdt:P31 wd:Q43229 .
  ?organizations wdt:P112 ?founded_by
}
```

Query Q_{18}:

```
SELECT (COUNT (DISTINCT ?humans) as ?
    count) WHERE {
  ?humans wdt:P2002 ?twittter . #with
      at least one P40 (child) statement
  ?humans wdt:P108 ?organizations .
  ?organizations wdt:P112 ?founded_by
}
```

Query Q_{19}:

```
SELECT (COUNT (DISTINCT ?organizations)
    as ?count) WHERE {
#  ?organizations wdt:P31 wd:Q43229 .
  ?organizations wdt:P112 ?founded_by .
  ?organizations wdt:P106 ?professions
      .
}
```

Query Q_{20}:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
  ?professions wdt:P425 ?var6 .
  ?organizations wdt:P106 ?professions
      .
  ?organizations wdt:P31 wd:Q43229 .
  ?humans wdt:P108 ?organizations .
  ?humans wdt:P31 wd:Q5 .
}
```

Query Q_{21}:

```
SELECT (COUNT (DISTINCT ?humans) as ?
    count) WHERE {
  ?humans wdt:P108 ?occupationsStar .
  ?occupationsStar wdt:P106 ?
      professions .
  ?professions wdt:P425 ?var6 .
}
```

Query Q_{22}:

```
SELECT (COUNT (DISTINCT ?organizations)
    as ?count) WHERE {
  ?organizations wdt:P112 ?
      occupationsStar . # humans link
  ?occupationsStar wdt:P106 ?
      professions .
  ?professions wdt:P425 ?var6 .
}
```

Query Q_{23}:

```
SELECT (COUNT(DISTINCT ?humans) as ?cnt)
    WHERE {
  ?humans wdt:P106 wd:Q2066131 .
  ?humans wdt:P21 ?v2 .
}
```

Query Q_{24}:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
  ?humans wdt:P106 ?professions .
  ?humans wdt:P21 ?v2 .
  ?professions wdt:P425 wd:Q349 .
}
```

Query Q_{25}:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
  ?humans wdt:P106 wd:Q901 .
  ?humans wdt:P21 ?v2 .
}
```

Query Q_{26}:

```
SELECT (COUNT(DISTINCT ?humans) as ?
    count) WHERE {
  ?humans wdt:P106 ?professions .
  ?humans wdt:P21 ?v2 .
  ?professions wdt:P425 wd:Q336 .
}
```