Graph rewriting primitives for semantic graph databases sanitization

Adrien Boiret, Cédric Eichler, Benjamin Nguyen, Sara Taki

To cite this version:

Adrien Boiret, Cédric Eichler, Benjamin Nguyen, Sara Taki. Graph rewriting primitives for semantic graph databases sanitization. Computer Science and Information Systems, 2024, 21 (3), pp.23. ⟨10.2298/csis230426026b⟩. ⟨hal-04631734⟩

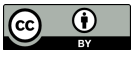
HAL Id: hal-04631734

https://hal.science/hal-04631734v1

Submitted on 2 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons CC BY 4.0 - Attribution - International License

Computer Science and Information Systems 21(3):1033–1054 https://doi.org/10.2298/CSIS230426026B

Graph Rewriting Primitives for Semantic Graph Databases Sanitization*

Adrien Boiret, Cédric Eichler, Benjamin Nguyen, and Sara Taki

INSA Centre Val de Loire, Laboratoire d'Informatique Fondamentale d'Orléans
88 Bd Lahitolle, 18000 Bourges, France
{adrien.boiret,cedric.eichler, benjamin.nguyen, sara.taki}@insa-cvl.fr

Abstract. Due to the rapid proliferation of data online, an important quantity of private or sensitive informations is being stored as linked data in graph databases (*e.g.*, represented as RDF). For such databases to be shared without jeopardizing privacy, they must first undergo a process known as database sanitization. During this process, databases are transformed following graph transformations that are usually described informally or through ad-hoc processes. However, a more thourough formalization of these transformations would aid in analysing the sanitization process, ensuring its correctness, and demonstrating the resulting privacy guarantees. This paper is an effort toward bridging the gap between the rigorous graph rewriting approaches and graph sanitization. We propose a graph transformation language to serve as a basis for constructing various sanitization mechanisms. This language relies on a set of elementary transformation operators formalized using a generic algebraic graph rewriting approach. Our language takes into account semantic and supports the equivalent of WHERE and EXCEPT clauses. As a proof of concept, we use these operators to implement two mechanisms from the literature, one generic (Local Differential Privacy) and one specifically introduced for semantic graph databases (sensitive attribute masking through anatomization). We propose an open-sourced tool implementing the elementary operators and the privacy mechanisms we derive from them relying on the Attributed Graph Grammar System (AGG) and its java API, providing a concrete tool implementing formal graph rewriting mechanisms to sanitize semantic graph databases. We present experimental results on this implementation regarding both proposed schemes and discuss its efficiency and scalability.

1. Introduction

In many collaborative data centric applications that collect personal data, such as car pooling, or smart metering, the data collectors (*i.e.*, the entity managing the application) need to publish and share raw data with various parties. These parties can range from internal developpers who need test data to data analysts in charge of producing predictive or explicative models, or even simply the scientific community. Since this database is composed of personal data, the European General Data Protection Regulation (GDPR) rules apply. In this paper, we consider the case where the database is sanitized (anonymized) prior to its release, which is compatible with the GDPR. Sanitization of a database means

* This paper is an extended version of [1]

that it is no longer possible (or at least very difficult, costly, and time consuming) to reidentify individuals in the dataset.

Nowadays, data are often organized as graphs with an underlying semantic to allow efficient querying and support inference engines. Such is the case in, for example, linked data and semantic web typically relying on RDF representation. Yet, while anonymization of tabular databases and untyped homogeneous graphs are well-researched, anonymization processes in such databases have been mildly studied. Most existing graph sanitization processes describe the underlying graph transformations in an informal, add-hoc, format-specific, and/or ambiguous way. We argue that such transformations should be 1) formalized, to ease the analysis of the process and the verification of both its correctness and the guarantees it provides; 2) generic and format-agnostic to improve the applicability of sanitization processes. In these regards, rigorous graph rewriting approaches appear to be perfect candidates.

In general, and despite the importance of graphs in databases and ontology representations, the use of formal graph rewriting techniques to model database evolutions is seldom studied. To the best of our knowledge, [1] is the only proposal using rigorous graph rewriting to model graph transformations for graph sanitization.

The present paper is an extension of [1]. We generalize this approach by formalizing a language of basic operators using attributed graph rewriting rules to serve as a basis for a greater number of graph sanitization mechanisms. We show this formalization in action on two examples privacy schemes, one classical (Local Differential Privacy (LDP)) already expressed in [1] and one specific to semantic graphs (anatomization) not explored in [1].

Contributions In this paper, we formalize and implement basic operators using AGG – The Attributed Graph Grammar System [22], one of the most mature development environments supporting the definition and application of typed graph rewriting systems [19]. These operators demonstrate the feasibility of the approach and should be enriched with other operators to build a library of operators supporting various anonymization schemes.

We choose to focus on eight basic operators. The seven first operators create or delete nodes, copy, cut, or merge edges, or randomize the targets of a relation. The eighth operator help better identify sets of nodes. Each of these operators can be represented as a single graph rewriting rule. Together, they can be combined into procedures expressive enough for our announced examples, randomization providing LDP guarantees and sensitive attribute anatomization.

We show that, given our formalization and implementation it is easy to describe sets of nodes or edges as the scope of our operators. This can be intuitively seen as clauses:

- WHERE allows restriction of the operator to a subset of nodes defined by a set of relations (*e.g.*, nodes that share a type or specific attribute).
- EXCEPT allows exclusion by the operator of a subset of nodes or relations defined by a set of relations.

The next Section presents the related work. In Section 3, we present the formalism of graph databases and graph rewriting we base our approach on, and the visual conventions used in the AGG rewriting tool. We also provide a running example for the rest of the paper. Section 4 describes the semantics and syntax of the eight basic operators, among which `JoinSet` that processes the keywords WHERE and EXCEPT for restriction and

exclusion in the scope of the other operators. Section 5 presents the description of two privacy procedures through our operators. Finally, Section 6 presents the results of a preliminary experimental evaluation made by applying one of the procedures of Section 5 to a sampling of the Sentiment140 dataset containing information about tweets.

2. Related Work

Private publication of semantic graph. A massive amount of work has focused on privacy in data presented as tables. They have resulted in multiple well-established models, such as k-anonymity [21] and differential privacy [9]. In this article, we consider *semantic databases*, such as RDF, which are *graph databases*. Such databases present the advantage of managing the semantics of the data, which we view as an advantage in the context of anonymization, since it helps the database designer better understand which data is sensitive, and how it can be modified. The aforementioned general anonymization concepts have been translated and applied to graph representations, but mainly in the context of homogeneous graphs such as social or computer networks [24,25].

Only a small batch of work has tackled the publication of semantic data-graphs. Radulovic et al. [17] introduced an anonymization framework for RDF that considers the specific characteristics of RDF specification. The framework aims to protect the privacy of particular entities of interest within RDF graphs. The underlying privacy concept is k-RDF anonymity, where an entity of interest within the graph is indistinguishable from k-1 others with regard to their quasi identifiers. They introduced different add-hoc anonymization operations that can be employed to implement such a model: generalization, suppression, atomization, and perturbation.

K-RDF-Neighborhood anonymity is an approach presented by Heitmann et al. [12]. Their work joins and builds upon other works conducted for homogeneous graphs, specifically by Zhou et al. [26], as well as on heterogeneous graphs, such as RDF, specifically by Radulovic et al. [17]. The central point behind their proposal is that indistinguishability should be guaranteed regarding not only quasi-identifiers but the whole one hop neighborhood of entities of interests. They presented a graph modification algorithm that only deletes edges

In [23], intermediary nodes regroup semantically related values of sensitive attributes via *semantic anatomization*, a process made to regroup attributes with strong semantic similarities. The privacy and utility of the process are then dependant on the creation of relevant or diverse groups. Modifications are expressed as SPARQL update queries.

Delanaux et al. [6] developed a declarative framework for anonymizing RDF graphs by replacing sensitive nodes by blank nodes. They consider a set of utility SPARQL queries and a set of privacy SPARQL queries, and generates operations on a graph so that utility queries can be answered while privacy ones cannot. The generated anonymization operations are in the form of SPARQL update queries of DELETE type (deletion of triples) and DELETE/INSERT (triples update).

While SPARQL is a standard description rather than add-hoc, it is RDF-specific and may exhibit ambiguities [8].

Graph rewriting for database modification. To generalize and abstract consistent updating methods, different works have used formalisms such as tree automata or grammars for XML (see [18] for a survey) or first order logic for graph databases (*e.g.*, [4,10]).

Despite the importance of graphs in databases and ontology representations, the use of formal graph rewriting techniques to model database evolutions is seldom studied. Formal graph rewriting techniques are usually based on *category theory*, an abstract way to deal with different algebraic mathematical structures and the relationships between them. Algebraic approaches of graph rewriting allow a formal yet visual specification of rule-based systems characterizing both the effect of transformations and the contexts in which they may be applied.

Few approaches relying on graph rewriting to formalize ontology evolutions have already been proposed [5,20,16]. They usually focus on formalization but do not provide an implementation. Recently, Chabin et al. [3,2] proposed SETUP, a tool for the management of RDF/S updates. In SETUP, graph rewriting rules formalize atomic updates and guarantee the preservation of RDF/S intrinsic constraints.

3. Background and setting

We consider databases to be modeled as attributed oriented multigraphs. In such models, it is customary for nodes and edges to have properties (among a finite set) and attributes (as words on a signature). In [2], RDF/S databases are modeled as a typed graph with 4 node types and 6 edge types. These types are inherent to RDF and thus the model can not be applied natively to arbitrary graph databases (*e.g.*, neo4j).

We argue that considering a single node type and a single edge type having a single attribute (named *att* and *prop*, respectively) is in fact at least as expressive. Indeed, typing and additional properties can be encoded via special kinds of relation. We believe this model to be able to capture most –if not all– graph database representations.

In what follows, we present a running example of such databases and target privacy-preserving scenarios for its exploitation. Then, we introduce the formalism used to model their transformations.

3.1. Running example and target scenarios

As a running example, we consider a graph database that contains information on travels, both professional and personal. An example of such a database is provided in Fig. 1.

It has nodes for relevant entities, people and travels, whose attributes are identifiers. It also has nodes for literals, *i.e.*, information that would have been stored as a raw value instead of a link to another node in formats like RDF, *e.g.*, last name, first name, and address for people, date and destination for travels. We do not differentiate nodes representing entities or literals.

Its edges describe both relations between entities, *e.g.*, "this person participated in this travel," represented by an edge of attribute ``attends'', but also relations between entities and their information, *e.g.*, "this person's name is in this literal," represented by an edge of attribute ``name''. Typing falls within this second case *e.g.*, "this node is a person" or "this literal is a city," represented by an edge of attribute ``type''.

In the example of Fig. 1, id105 (named Miller) attended travel id207 to Paris for professional reasons.

We give two motivating examples consisting in the application of two privacy mechanisms in this database, whose implementation will be shown to be possible in Section 5. In

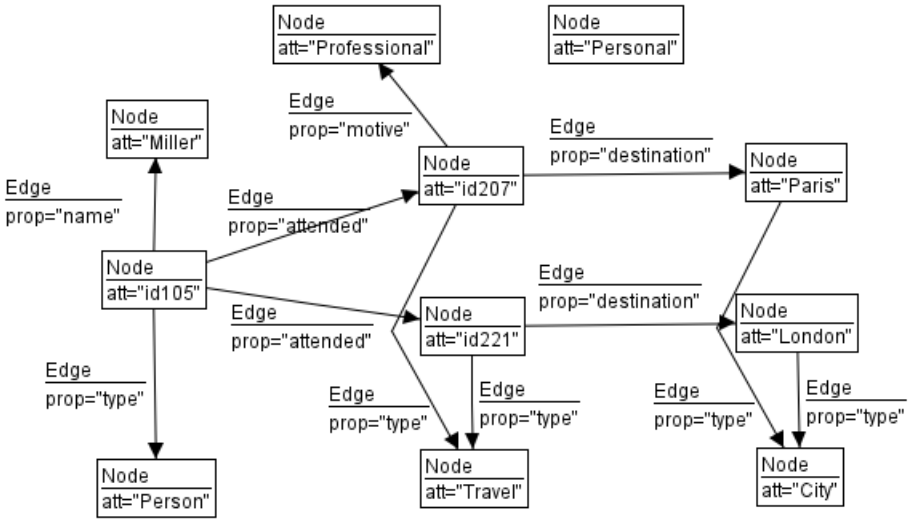


Fig. 1. Running example: instance of a database

these examples, we will note that the described procedures should only apply to specific nodes and relations.

- We want to provide plausible deniability with regard to the relation "destination" between personal travels and cities. To do so, we want to randomize this relation for personal trips exclusively, to preserve privacy with a bias towards correct answers to preserve utility. This corresponds to guaranteeing local differential privacy (LDP [14,7]) on trips with a "motive" edge leading to "personal". More precisely, we want to modify the database such that querying it to output the destination of personal trips would be locally differentially private.
- For professional trips, we want to obfuscate the relation "destination" between travels and cities, as to hide precise dates and frequency of collaboration between the database's company and its collaborators, for instance. This can be done, for instance, by grouping trips in certain cities together (*e.g.*, "Paris," "Bordeaux," "Toulouse" all grouped in the more nebulous group "France") and rerouting the "destination" edges towards those groups rather than a precise value. This would mean that we want to apply anatomization [17,23] where the "destination" attributes of travels with attribute "motive" set to "professional" is considered sensitive.

3.2. Graph Rewriting Rules

We adopt the Single Push Out (SPO) formalism ([15]) to specify rewriting rules as well as one of its extensions to specify additional application conditions and restrict their applicability, *Negative Application Conditions* (NACs) [11], and *Positive Application Conditions* (PACs).

The SPO approach is a simple yet expressive way of fully formalizing graph transformations. Furthermore, it offers an easy-to-understand yet formal graphical view of the

graph transformation. In what follows, we adopt the graphical representation provided by AGG's graphical interface.

The SPO approach is an algebraic approach based on category theory. A rule is defined by two graphs – the Left and Right Hand Side of the rule, denoted by L and R – and a partial morphism m from L to R (*i.e.*, an edge-preserving morphism m from a subgraph of L to R).

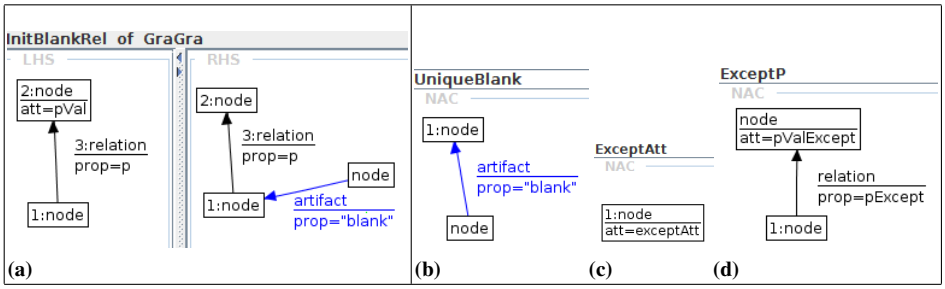


Fig. 2. Identifying nodes having some relation with some value to be replaced by a blank: (a) SPO core and its NACs (b) uniqueness of the blank replacing a node, (c) EXCEPT a specific node, (c) EXCEPT nodes with some relation having some value

Fig. 2a formalizes the SPO core of the first step of the *blank* operator. It identifies the nodes to be replaced and creates a blank for each. Its L is composed of a relation `p` from some node (1) to a node (2) with an attribute `pVal`. R has the same pattern plus a new node that is the source of an artifact edge labeled ``` ``blank'' ``` whose target is node (1). Note that the attributes of two nodes of R are not represented. In general, this can stem from three possibilities: (i) it does not matter, *e.g.*, an unattributed node in L or NAC will match any node; (ii) it can be inferred, *e.g.*, a node in R or NAC has the same attributes as the node it is matched with in R. This is the case for node (2) in the example; (iii) a node in R is created without an attribute value. This is the case for the node representing a blank.

By convention, an attribute value within quotation mark (*e.g.*, ``` ``blank'' ```) is a fixed constant, while *a value noted without quotation mark (e.g., p) is a variable that is either a wildcard matching any value or whose is given as input.*

The partial morphism from L to R is specified in the figure by tagging graph elements - nodes or edges - in its domain and range with a numerical value. An element with value i in L is part of the domain of m and its image by m is the graph element in R with the same value i. For instance, in Fig. 2a, the notation *1:* for the nodes with an unspecified attribute in L and R indicates that they are mapped through m. In the following, we refer to such nodes as `1:node`.

A graph rewriting rule $r = (\mathsf{L}, \mathsf{R}, m)$ is applicable to a graph G iff there exists a total morphism $\tilde{m} : \mathsf{L} \to G$. The result of the application of r to G w.r.t. $\tilde{m}$ is the object of the push-out of the diagram composed by L, R, G, m, and $\tilde{m}$. Informally, the application of r to G with regard to $\tilde{m}$ consists of modifying G by (1) removing the image by $\tilde{m}$ of all elements of L that are not in the domain of m (*i.e.*, removing $\tilde{m}(\mathsf{L} \backslash Dom(m))$); (2) removing

all dangling edges (*i.e.*, deleting all edges that were incident to a node suppressed in step (1)); (3) adding an isomorphic copy of all elements of R that are not mapped through m.

In Fig. 2a, the rule is applicable to any non-artifact edge with the possibility to specify the attribute of the edge (`p`) and/or of its target (`pVal`) as input of the rule. Indeed, such attributes are either wildcard that matches any value or their value may be given as input. For example, with the input (``rdf:type'', ``city''), the rule only maps `1:node` of *type city*. The application of the rule consists of adding an unattributed (blank) node and a ``blank'' artifact edge from the new node to `1:node`, representing the fact that the latter will be replaced by the former.

NACs are well-studied extensions that restrict rule application by forbidding certain patterns in the graph. EXCEPT clauses will mostlty be encoded through NACs. A NAC for a rule r is defined as a constraint graph which is a super-graph of its left-hand side. An SPO rule $r = (L, R, m)$ with NACs is applicable to a graph iff: (i) there exists a total morphism $\tilde{m} : \mathsf{L} \to G$ (this is the classical SPO application condition); (ii) for all NAC N associated with r, there exists *no* total morphism $\bar{m} : N \to G$ whose restriction to L is $\tilde{m}$.

By convention, since NACs are super-graphs of L, unnecessary parts of L are not depicted when illustrating a NAC. Graph elements common to L and NAC are identified by a numerical value, similar to elements mapped by m.

Figures 2b, 2c, and 2d represent NACs associated to the SPO core of Fig. 2a. The first specifies that `1:node` must not be the target of a ``blank'' artifact edge, ensuring that we will not create several blank nodes linking to the same existing node. The second maps `1:node` with a node whose attribute `exceptAtt` is given as input. This forbids the application of the rule to any node with such attribute, akin to an EXCEPT clause excluding a particular node. The third NACs forbids the application of the rule to any `1:node` either source of a relation `pExcept` and/or with value `pValExcept`. Indeed, like variables appearing in L those in NACs can be either wildcards or their value may be given as input. *pExcept* given as ``capital city of'' the rule exclude all `1:node` that are a *capital city*. With `pValExcept` given as ``New York' the rule will exclude all `1:node` that are linked to the a node with the attribute ``New York' (regardless of the relation). Note that here, at least one of the two should be given a input value, or the NAC would match L and the rule would never be applicable. For instance, with

A rewriting procedure –or rule sequence– as we consider it here is a succession of steps. Each step is the application of a rewriting rule as long as the rule applies or a specified number of times. We consider that *when a rule is applicable w.r.t. several morphisms, it chooses one uniformly at random*.

4. Our Language

Our language's goal is to support the specification of privacy procedures and mechanisms that will apply operators (*e.g.*, randomization) on some particular sets of nodes and edges.

In its general form, we would expect each of our instructions to be the combination of an operator, a set of sources (or subject), a set of destinations (or objects), and potentially additional operator- or procedure-specific requirements (*e.g.*, new targets for randomization, or lists of identifiers and sensitive attributes). To this end, we aim to provide a small

number of simple operators, as well as a a way to descrinbe the sets of nodes and edges on which they will apply.

This section introduces the language we build and its design choices: procedures composed of elementary operators applied to simple cases, and the special operator `JoinSet` that allow it to preserve its expressiveness.

4.1. Design principle

Simple operators Our main design choice is to keep the basis of our language as simple as possible. We propose a set of eight operators, each described by a single graph rewriting rule, with an easy-to-match pattern. They allow to create and delete nodes, copy, modify, cut or merge edges, randomize the sources or destination of a relation. Al these operators work on very basic patterns, and the main role of the eighth operator, `JoinSet`, is to reduce complex application cases to the simple cases handled by our operators.

Node and Edge identification To limit the use of an operator to specific nodes, our language needs a way to define and manipulate sets of node and edges.

As discussed in the example of 3, these sets can be defined as types, can exclude types, or can require the presence or absence of a certain property at a certain value. It is possible to specify these kinds of restrictions directly in graph rewriting rules, using L, PACs, and NACs. However, this would have the important drawback of necessitating numerous versions of the same rule and/or complicating the pattern matching phase. We therefore believe a pre- and post-processing procedure to construct temporary add-hoc sets to be necessary.

To identify *nodes*, our language considers either the label of the node (*e.g.*, the node.s labeled "id207"), or the existence of labels with a given edge towards nodes of a certain label (*e.g.*, the node.s with edges "type" heading to nodes labeled "City"). As a shorthand, we sometimes say that the nodes with a label S and an edge labeled p towards a node labeled O as "matching (S, p, O)". Such a triplet (S, p, O) can be called a **set**, and identified with a single letter $\mathcal{X}$.

We note that we can specify two special value in these sets:

- $\star$ corresponds to any label we want
- `null` corresponds to no expectation, *i.e.*, not only any label but the existence is not necessary either

For instance, $(\text{City}, null, null)$ would match any node whose label is "City", whether they have an outgoing edge or not, which includes one node in our example. $(\text{City}, *, *)$ would match any node whose label is "City" with any outgoing edge at all, which includes no node of our example. Note that `null` differs from $\star$ only when both p and O are equal to `null`. Indeed, $(\text{City}, *, null)$ would not constrain O but impose the existence of some p, and thus of some O.

To identify *edges*, our language considers their labels and identifiers of their source or target (*e.g.*, all edges labeled "destination" from a node matching $(*, \text{"type"}, \text{"Travel"})$ to a node matching $(*, \text{"type"}, \text{"City"})$).

Pre-processing using `JoinSet` To reduce a wide variety of operators' scopes and targets to the basic case, we only define operators on very simple cases, *i.e.*, by considering simple node sets, and when applicable the edges that link them. For this to not hinder the

expressiveness of our language, we need a way to create or populate sets that correspond to nodes matching intricate conditions. For this, we use the following instruction:
`JoinSet` (x, X) `Where` $\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ `Except` $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_m\}$

This primitive matches nodes that match every $\mathcal{X}_i$ in the `Where` section and match none of the $\mathcal{Y}_i$ in the `Except` section. When it does, it create from it an edge labeled x towards a node labeled X.

Graph rewriting primitives: We present here the few primitives, formalized and implemented directly using AGG, that create or delete nodes, copy, divide, or merge edges, and randomize specific relations[1].

- `NewNode`(X) creates a new node labeled X
- `DeleteNode`$(\mathcal{X})$ matches all nodes matching $\mathcal{X}$ and deletes them.
- `JoinSet` (x, X) `Where` $\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ `Except` $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_m\}$ (explained above)
- `EdgeCopy`$(\mathcal{S}, p, \mathcal{O}, p')$ matches couples of source nodes matching $\mathcal{S}$ and destination nodes matching $\mathcal{O}$, where there is an edge from the source to the destination labeled p.
  When a match is found, creates an edge labeled p' from the source to the destination.
- `EdgeReverse`$(\mathcal{S}, p, \mathcal{O}, p')$ matches couples of source nodes matching $\mathcal{S}$ and destination nodes matching $\mathcal{O}$, where there is an edge from the source to the destination labeled p.
  When a match is found, creates an edge labeled p' from the destination to the source.
- `EdgeCut`$(\mathcal{S}, p, \mathcal{O}, p_I, M, p_O)$ matches pairs of source nodes matching $\mathcal{S}$ and destination nodes matching $\mathcal{O}$, where there is an edge from the source to the destination labeled p.
  When a match is found, creates a new intermediary node of label M. Then, it creates an edge labeled p_I from the source to the intermediary, an edge labeled p_O from the intermediary to the destination, and finally it deletes the edge p from the source to the destination.
- `EdgeChord`$(\mathcal{S}, p_I, \mathcal{M}, p_O, \mathcal{O}, p)$ is the converse of `EdgeCut`, and matches triplets of source nodes matching $\mathcal{S}$, intermediary nodes matching $\mathcal{M}$ and destination nodes matching $\mathcal{O}$ where there is an edge from the source to the intermediary labeled p_I and there is an edge from the intermediary to the destination labeled p_O.
  When a match is found, creates an edge labeled p from the source to the destination.
- `RandomTarget`$(\mathcal{S}, p, \mathcal{O}, \mathcal{T})$ matches all edges labeled p between a source matching $\mathcal{S}$ and a destination matching $\mathcal{O}$, and it reroutes this edge by picking a new target uniformly among nodes matching $\mathcal{T}$.

Procedures: A procedure is a sequence of instructions, executed in order. They can be as simple as intermediary operators, or complex enough to describe graph manipulations guaranteeing certain types of privacy. For instance, a procedure `DeleteEdge`$(\mathcal{S}, p, \mathcal{O})$ meant to delete all edges labeled p between a source matching $\mathcal{S}$ and a destination matching $\mathcal{O}$ would be:

1: `EdgeCut`$(\mathcal{S}, p, \mathcal{O}, p_I, "ToBeDeleted", p_O)$
2: `DeleteNode`$("ToBeDeleted", null, null)$

[1] Rules specifications are available at `https://github.com/ceichler/granon/blob/master/anonOperator.ggx`

This procedure first cuts every occurrence of p between $\mathcal{S}$ and $\mathcal{O}$ in two with a new node labeled "ToBeDeleted" (line 1). Then, it immediately deletes all those "ToBeDeleted" nodes (line 2), thus erasing the original edges.

5. Privacy Procedures

This section introduces procedures that provide privacy guarantees: Local Differential Privacy [7] and anatomization [17]. These procedures are expressed as combinations of our primitives.

5.1. Local Differential Privacy

The first proposed procedure support randomization to achieve local differential privacy (LDP) as defined and used in [14,7]. Assuming a query outputing the value.s of a property for a particular node or a set of nodes, we wish to modify the graph to make the query satisfy LDP.

Definition 1 (Local DP (Duchi *et al.*) [7]). *Let χ be a set of possible values and Y the set of noisy values. A mechanism $\mathcal{M}$ is ε-locally differentially private (ε-LDP) if for all $x, x' \in \chi^2$ and for all $y \in Y$ we have*

$$Pr[\mathcal{M}(x) = y] \leq e^{\varepsilon} \times Pr[\mathcal{M}(x') = y]$$

LDP-mechanisms outputing a value in Y achieve optimal utility for a given ε by giving an answer randomly drawn from a staircase distribution over Y, with the most probable value being the *real* value -whose probability depends solely on $|Y|$ and ε- and all other values being equiprobable [13].

To achieve ε-LDP for a set R of relations, our random operator should therefore transform each $(s,t) \in R$ into a relation $(s,t') \in R'$ under the following specification:

$$P(t') = \begin{cases} 0 & t' \notin T \\ \frac{K}{|T|-1+K} & t' = t \\ \frac{1}{|T|-1+K} & t' \in T \wedge t' \neq t \end{cases}$$

with K an integer approximation of e^{ε}, $K = \lfloor e^{\varepsilon} \rfloor$.

To do so, we pick a new target at random, but, to obtain a staircase distribution from a uniform distribution –used to choose the morphism with regard to which the transformation rule is applied–, we skew the odds by creating $K-1$ dummies. Picking a dummy as a target should ultimately result as giving the true answer to recreate a staircase distribution. The procedure, defined in Alg. 1, and detailed thereafter, has the following arguments:

- $\mathcal{S} = (X_S, s, S)$ to match the sources of the relation to randomize
- p to match the edges of the relation to randomize
- $\mathcal{O} = (X_O, o, O)$ to match the destinations of the relation to randomize
- K the factor by which correct answers are more likely than other values

Algorithm 1 Procedure LDP(S,p,O,K)

1: **for** i from 1 to $K-1$ **do**
2: `NewNode`$(Dummy)$
3: **end for**
4: `JoinSet`(o,O)`Where`$\{(Dummy,*,*)\}$ `Except`$\{\}$
5: `EdgeCut`$(\mathcal{S},p,\mathcal{O},p_I,"Intermediary",p_O)$
6: `EdgeCopy`$((*,null,null),p_O,(*,null,null),p_N)$
7: `RandomTarget`$((*,null,null),p_N,(X_O,o,O),(*,o,O))$
8: `EdgeChord`$(\mathcal{S},p_I,(*,p_N,Dummy),p_O,\mathcal{O},p)$
9: `EdgeChord`$(\mathcal{S},p_I,(*,null,null),p_N,\mathcal{O},p)$
10: `DeleteNode`$(Dummy,*,*)$
11: `DeleteNode`$(Intermediary,*,*)$

Recurring example: We illustrate the steps of this procedure in a recurring example presented in Figures 3 to 8. We present a $\ln(4)$-LDP-providing randomization of persons attending travels, *i.e.*, the procedure LDP((type,Person),attended,(type,Travel),3).

In this example, nodes and edges in black are definitive data that started or are meant to remain in the graph. Nodes and edges in blue are temporary nodes that are only used as intermediary elements in the rewriting process (e.g. the Dummy nodes). Nodes and edges in bold/thick have recently been created (e.g. the "attended" edge in Figure 6). Nodes and edges that are dashed with reduced opacity have recently been deleted (e.g. the "Dummy" nodes in Figure 8).

Initialization: We start by creating a bias for the correct value, by creating $K-1$ dummy nodes thanks to the `NewNode` operator This rule is to be repeated as many times as required to obtain ε-LDP, then they are matched to $(*,o,O)$ with `JoinSet` (lines 1 and 2). Creating two dummies makes the truth three times likelier, and suits $\varepsilon \geq \ln(3)$. If we create 0 dummies, then we are 0-LDP as the edges' targets will be uniformly randomized.

This step is illustrated in Figure 3 with the creation of 3 dummy nodes.

Edge Cut: We want to use `RandomTarget` to reroute edges p towards a new target matching $(*,o,O)$ at random. However, picking a dummy as a new target means we want to keep the original target. This means that instead of rerouting the edges p directly, we would like to create "forks" that lead both to the original and new targets. Since hyper-edges are not objects of our graphs, we use `EdgeCut` and `EdgeCopy` to emulate this behavior (lines 3 and 4):

- First, we cut the edges p from $\mathcal{S}$ to $\mathcal{O}$ into p_I and p_O edges coming from a middle node labeled "Intermediary"
- Then, we create a copy of p_O edges labeled p_N

This is pictured in Figure 4.

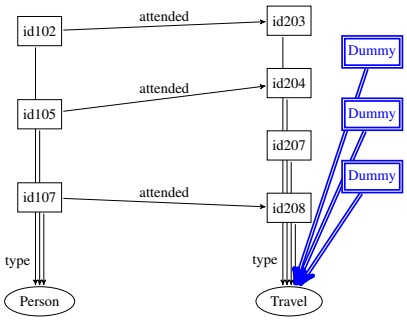


Fig. 3. Create dummy travel nodes

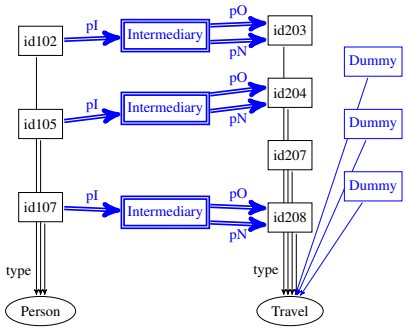


Fig. 4. Split "attended" edges into pI, pO, pN

Randomizing: We then use `RandomTarget` (line 5) toto redirect the edges p_N towards any target matching $(*, o, O) = \mathcal{O}$, with uniform probability. We have one chance to pick the original node, and $K - 1$ chances to pick a dummy. This means we are K times more likely to pick one of these nodes than any other real node matching $\mathcal{O}$. This is pictured in Figure 5, where node matching *(*,type,Travel)* are depicted in red.

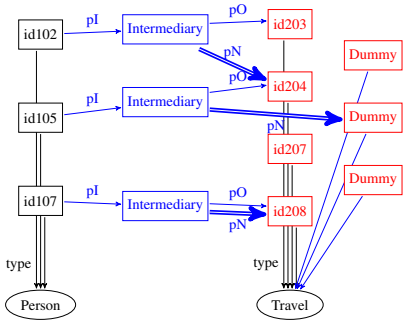


Fig. 5. Randomize the target of pN edges

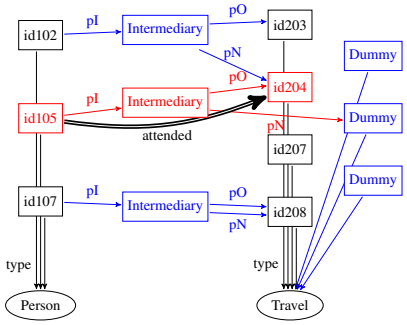


Fig. 6. Chord rerouting: pN targets a Dummy

Rerouting: We start by dealing with the case where p_N was rerouted to a dummy (line 6). In that case, the intermediary node matches $(*, p_N, Dummy)$, and we want to keep the old target of p, which means the current target of p_O. We say that if there is a chord from a node matching $\mathcal{S}$ to an intermediary node matching $(*, p_N, Dummy)$ to a destination node matching $\mathcal{O}$ with edges p_I, p_O, then we create the edge p from the source to the old destination. This is pictured in Figure 6, the matched chord being depicted in red.
We then deal with the case where p_N was rerouted to a real target (line 7). We say that if there is a chord from a node matching $\mathcal{S}$ to any intermediary node to a destination node matching $(*, o, O)$ with edges p_I, p_N, then we create the edge p from the source to the new destination. This is pictured in Figure 7, the matched chords being depicted in red.
We note that this also creates unnecessary edges from sources to dummies. This is not a problem, as dummies will be deleted shortly, as pictured in Figure 8.

Termination: Since dummies and intermediary nodes are artifacts we created rather than real nodes of the graph, we end the procedure by using `DeleteNode` to delete all nodes of label "Dummy" and "Intermediary" (lines 8 and 9). This also deletes all edges p_I, p_O, p_N linked to intermediary nodes, as well as edges p unduly targeting dummies generated during rerouting.

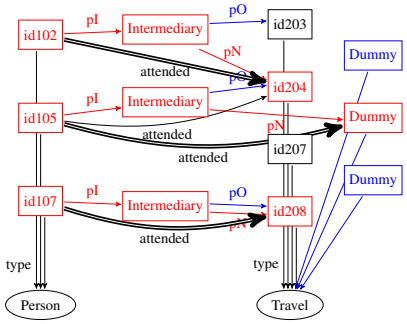


Fig. 7. Chord rerouting: every path pI, pN generates a new "attended" edge

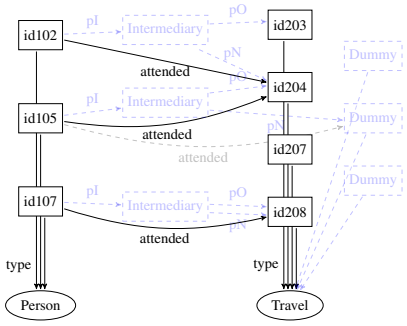


Fig. 8. Cleanup

5.2. Anonymization through Anatomization

In this section, we show that sorting sensitive attributes in groups to prevent inferences can be naturally expressed in our elementary operators.

We consider that some attributes of our graphs are sensitive, as described in [17,23]. These papers describe an anonymization process that involves the deletion of explicit identifiers (*e.g.* first and last name), and the separation of nodes with quasi-identifiers (*e.g.* date of birth, zip code) to a set of sensitive attributes (*e.g.* religion, sexual orientation). To cut links between quasi-identifiers and the sensitive attributes, the possible values of sensitive attributes are put into groups through a process called semantic anatomization [23]. Their approach is qualified as semantic as it concentrates on semantic-aware grouping of sensitive attributes. Furthermore, it retains the correlation between entity quasi-identifiers and semantically related sensitive values.

While the creation of these groups itself is a potentially involved and complex process that goes beyond the scope of graph rewriting, the redirection of links from identifiers to sensitive attributes is possible if the groups are provided.

We consider that our graph contains explicit identifiers (edge labels $e_1, \ldots, e_n$), quasi-identifiers (edge labels $q_1, \ldots, q_m$), sensitive attributes (edge labels $p_1, \ldots, p_k$), and group nodes that aggregate attribute values. A value X is part of a group if nodes of label X have an edge of label "inGroup" pointed at this group's node. The anonymization is made as follows:

Recurring example: The algorithm is detailed thereafter and illustrated through a toy example where the destinations of travels are considered sensitive $(p_i) = (destinatiation)$, the identifiers are names $(e_i) = (name)$, and the quasi-identifiers are types $(q_i) = (types)$.

More specifically, we illustrate the steps of sensitive attributes redirection in a recurring example presented in Figures 9 to 14.

Erasing explicit identifiers: Line 1 deletes the edges towards sensitive attributes. The values are preserved, but the links are cut for anonymization. In the example, this would lead to the deletion of every edge whose attribute is "name".

Flagging quasi identifiers: Lines 2 and 3 create a node QI and relate every node with at least one quasi identifiers to QI through an edge $hasQI$. In the example, all nodes with a *type* are the source of an edge $hasQI$ whose target îs QI.

Algorithm 2 Procedure Anat$((e_i)_{1\leq i\leq n}, (q_i)_{1\leq i\leq m}, (p_i)_{1\leq i\leq k}, inGroup)$

1: For all e_i do `DeleteEdge`$((*, null, null), e_i, (*, null, null))$
2: `NewNode`(QI)
3: For all q_j do `JoinSet`$(hasQI, QI)$ `Where`$\{(*, q_j, *)\}$ `Except`$\{\}$
4: For all p_i do `EdgeChord`$((*, hasQI, QI), p_i, (*, null, null), inGroup, (*, null, null), p'_i)$

5: For all p_i do `EdgeCut`$((*, hasQI, QI), p_i, (*, null, null), p^i_I, "Intermediary", p^i_O)$
6: For all p_i do
`EdgeChord`$(("Intermediary", null, null), p^i_O, (*, null, null), inGroup, (*, null, null), p''_i)$

7: For all p_i do `EdgeReverse`$(("Intermediary", null, null), p''_i, (*, null, null), p''_i)$
8: For all p_i do
`EdgeChord`$((*, null, null), p''_i, ("Intermediary", null, null), p^i_O, (*, null, null), hasOne)$

9: For all p_i do `EdgeCopy`$((*, null, null), p'_i, (*, null, null), p_i)$
10: For all p_i do `EdgeCut`$((*, hasQI, QI), p'_i, (*, null, null), p^i_I, "Intermediary", p^i_O)$
11: `DeleteNode`$("Intermediary", null, null)$
12: `DeleteNode`$(QI, null, null)$

Redirecting sensitive attributes: Line 4 matches cases where a node with quasi-identifiers has an edge p_i pointing towards the value of a sensitive attribute, which itself has an edge $inGroup$ designating a group of the semantic anatomization. These matches are depicted in red on Fig. 9.
When such a match is detected, an edge p'_i is directly traced from the initial node to the group. The label used is p'_i instead of p_i to avoid undue deletions of redirected edges in line 5.
Should we want to preserve those sensitive but ungrouped edges, we would only split edges pointing towards grouped values by replacing line 5 with:

For all p_i
do `EdgeCut`$((*, hasQI, QI), p_i, (*, inGroup, null), p_{I,i}, "Intermediary", p_{O,i})$

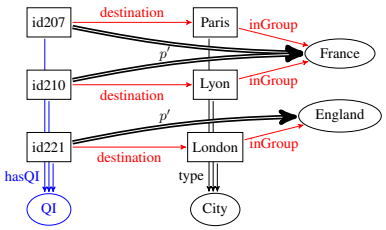


Fig. 9. Short-circuit sensitive values

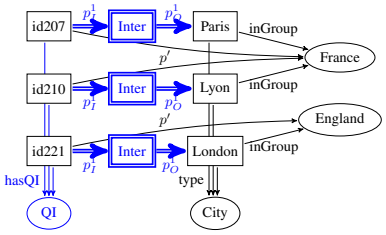


Fig. 10. Split sensitive edges into p_{I_1}, p_{O_1}

Keep track of sensitive attribute values: Lines 5 through 8 aim to keep track of the cardinalities of each sensitive value in the anatomization groups, represented by the edges p_i we delete.
Line 5 creates an intermediary node between the source and the value of its sensitive attribute, as depicted in Fig. 10. Line 6 links those intermediary nodes with the group of their sensitive attributes with an edge labeled p_i'', illustrated in Fig. 11. The matched chords are depicted in red.

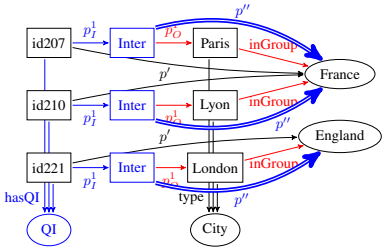


Fig. 11. Link intermediaries with groups

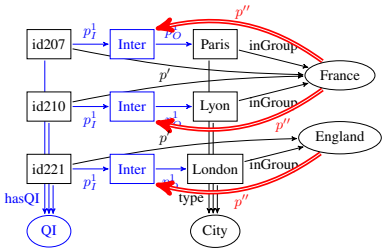


Fig. 12. Reverse from group to intermediaries
This edge is inverted in line 7, then used in line 8 to create a direct link between the group and the sensitive attribute, as shown if Fig. 12 (where the inverted p'' are depicted in red) and 13 (where the p'', p_O^i chords are depicted in red), respectively.

Bringing back p_i: Line 9 creates a copy p_i of each p_i' linking a node with the group its sensitive value is in. Then line 10 cut the p_i' edges to prepare them for deletion.

Cleanup: Finally, lines 11 and 11 erase the nodes we created in previous steps. This notably includes the last trace of edges p_i that could not have been redirected –*i.e.*, those that did not belong to an anatomization group– for which the information is gone for good. This last step is depicted in Fig. 14.

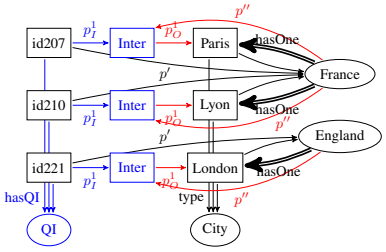


Fig. 13. Chord from groups to values

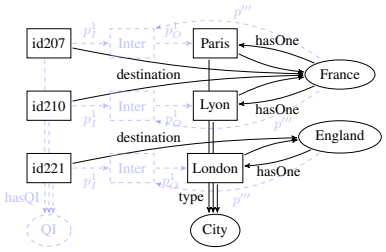


Fig. 14. Cleanup

6. Experimental evaluation

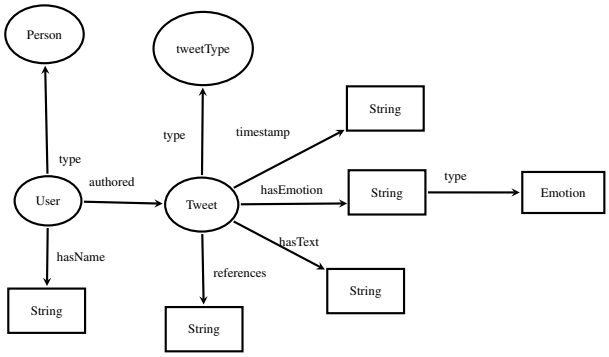


Fig. 15. Schema for Sentiment140

This section investigates the feasibility of the approach and provides a preliminary experimental evaluation by applying two privacy procedures to a real dataset. We implemented

the formal specification of the operators using AGG – The Attributed Graph Grammar System ([22]), one of the most mature development environment supporting the definition and application of typed graph rewriting systems ([19]). Using AGG's Java API, we implemented Granon, a tool in Java[2] to handle their management and on-the-fly modification (*e.g.* the definitions of multiple NACs and PACs in the *JoinSet* operator). Granon also supports the procedures defined herein. Experiments are conducted in a single-thread on an Intel Xeon Gold 5215 2.5GHz with 64 GB RAM and a 20Go JVM heap size.

Dataset(s): The experiments are conducted on a sampling of the Sentiment140 dataset composed of 1.6 million tweets[3], which we have parsed to load as a knowledge graph conforming to the schema shown in Fig. 15. The dataset is composed of *tweets* authored by a named user. Tweets each have a timestamp, a full text, and an emotion (positive, neutral, or negative). They may reference users' name. The users, tweets, and emotions are all typed.
The most important factor regarding runtime is the number of tweets and the size of the graph. Therefore, we apply our experiments on the graphs resulting from the parsing of the t first tweets of the datasets, with $t = 200$, 400, 600, 800, 1000, 1200, and 2000.

Nature of the experiments: To investigate the scalability of the proposal, we apply an instantiation of both our privacy schemes:

1) *Procedure LDP((*,"type", "tweetType"), "hasEmotion", (*, "type", "Emotion"), ln(2))* as described in Sec. 5.1. We arbitrarily consider the emotion of a tweet to be the sensitive value and use $\varepsilon = ln(2)$, the value of $epsilon$ having a negligible influence on the runtime.
2) *Procedure Anat((hasText), (references), (timestamp), inGroup)* described in 5.2. We consider texts to be identifiers, timestamps to be sensitive and references to be quasi-identifiers. Therefore, the procedure will delete all "hasText" edges and generalize the timestamp of any tweet that references someone. After considering the dataset, we construct anatomization groups representing a one minute timewindow: for instance a timestamp with the value "Mon Apr 06 22:20:19 PDT 2009" belongs to the group "Mon Apr 06 22:20 PDT 2009". Groups are constructed while parsing the experimental datasets, resulting in slightly bigger graphs for the same t.

Experimental results: Average and median execution times (in ms) over 50 runs of the procedures are reported in Fig. 16 for the various t. The input graphs' sizes (*i.e.*, their number of vertices #V and edges #E) constructed by parsing the first t tweets of the dataset are reported in Tab. 1. The size of the graph is linear in *t*, which is consistent with the schema. The average and median runtimes for the LDP procedure are overlapping as the distribution has a very low standard deviation. The distribution of runtimes for the Anatomization procedure consistently comports high outliers for every t.

Experimental interpretation: As expected, the LDP procedure is more time consuming than the Anatomization procedure. For $t = 1000$, the median for LDP and Anatomization

[2] available at `https://github.com/ceichler/granon`
[3] `https://www.kaggle.com/kazanova/sentiment140`

Table 1. Size of the graphs resulting from parsing the t first tweets of the dataset

	LDP		Anatomization	
t	#V	#E	#V	#E
100	531	743	538	843
200	1042	1465	1054	1665
300	1558	2201	1576	2501
400	2081	2930	2105	3330
500	2605	3661	2636	4161
600	3113	4386	3150	4986
700	3623	5115	3665	5815
800	4147	5889	4195	6659
900	4648	6576	4704	7476
1000	5170	7310	5232	8310
2000			10393	16531
3000			15463	24625

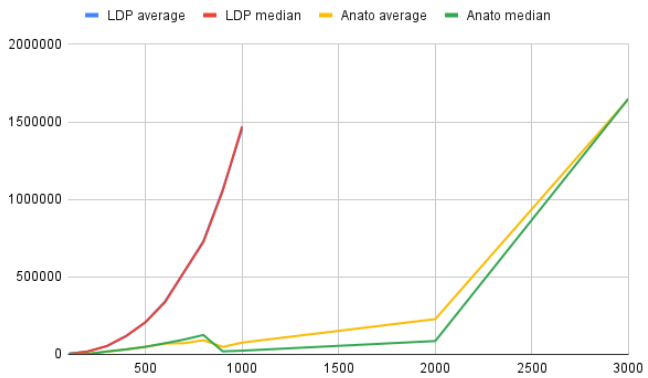


Fig. 16. Experimental results runtime (ms) with various t

are 1,47 * 10^6 ms and 2,4 * 10^4ms, respectively. Furthermore, while the asymptotic complexity of the latter is roughly quadratic for $t \geq 2000$, the former is over-quadratic. The median execution time of the LDP procedure roughly triples from $t = 400$ to $t = 500$ (1,2 * 10^5 ms to 3,3 * 10^5ms) when the number of nodes is multiplied by 1,25. The average execution time of the Anatomization procedure is multiplied by 547 regarding t = 100 and t =3000, with 28,7 times the number of nodes.

This can be explained by the randomization inherent to LDP. Indeed, randomizing the target of a relation requires restarting the matching process from scratch, without benefiting from optimizations (e.g. smart backtracking). Otherwise, the first match would influence all others. Therefore, randomizing the sensitive value of n items requires -in the randomizing step- running n search for graph homomorphisms, which is itself super-linear in the size of the graph.

A 30 minute run would allow to execute the LDP (resp. anatomization) scheme on a graph slightly bigger than #V = 5170, #E = 7310 (resp. #V = 15463, #E = 24625). While such a runtime is very much reasonable when considering the sanitization of a database prior to its release as the process is expected to be run infrequently, the asymptotic complexities discourage the application of the proposed graph-rewriting based techniques to big graphs.

7. Conclusion

This paper is a first effort toward bridging the gap between rigorous graph rewriting and graph sanitization. We propose a basic yet expressive set of atomic operators, formalized as simple Single Push-Out graph rewriting rules, providing a generic expressive rigorous definition that can be parametrized with node and set identifiers. We show how they can be used to express privacy-preserving graph databases publication mechanisms, namely local differential privacy and anatomization. These operators and procedures have been subject to an open-source implementation[4]. This work stands as a proof of concept and a first step towards a graph rewriting based approach to graph database sanitization. A lot of considerations and work remain open on the topic.

Firstly, while the current implementation would reasonably allow the sanitization of small to medium graphs, its scalability remains limited due to high asymptotic complexity. A first effort would be required to reduce multiplicative constants, for example, by expanding operators to reduce the number of transformations, e.g. implementing an operator for edge suppression or edge modification rather than relying on two operators to conduct the operation; 2) or even the asymptotic complexity of our procedures, for example by considering a subgraph for the transformation, as procedures work on local properties.

Another natural outlook would be to expand the tool to encompass other privacy protocols (e.g. [6]). This might require the use of other operators, that we hope share the simplicity and versatility of our current set.

Furthermore, our language and operators are designed to work on simple relations between two sets of nodes. In real world usecases, most queries are joins of several relations (or to speak in graph terms, path queries on more than one edge). For a sanitization mechanism to preserve good qualities on such requests while still providing privacy guarantees, it is likely that we will need operators adapted to the preservation of invariants on composite paths. It is possible, but not yet shown, that some such operators can be built as compositions of one-relation operators.

Ultimately, as a more general goal, we aim to expand our current tool to allow a user to specify semantic and privacy constraints. Such a tool would compile those requests as sequences of our operators and offer to perform the resulting graph transformations.

Acknowledgments. This work is part of the SENDUP project, supported by the French National Research Agency, under grant ANR-18-CE23-0010.

[4] available at `https://github.com/ceichler/granon`

References

1. Boiret, A., Eichler, C., Nguyen, B.: Privacy operators for semantic graph databases as graph rewriting. In: Chiusano, S., Cerquitelli, T., Wrembel, R., Nørvåg, K., Catania, B., Vargas-Solar, G., Zumpano, E. (eds.) New Trends in Database and Information Systems. pp. 366–377. Springer International Publishing, Cham (2022)
2. Chabin, J., Eichler, C., Ferrari, M.H., Hiot, N.: Graph rewriting rules for RDF database evolution: optimizing side-effect processing. Int. J. Web Inf. Syst. **17**(6), 622–644 (2021)
3. Chabin, J., Eichler, C., Halfeld-Ferrari, M., Hiot, N.: Graph rewriting rules for rdf database evolution management. In: Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services. p. 134–143. ACM (2020)
4. Chabin, J., Halfeld Ferrari, M., Laurent, D.: Consistent updating of databases with marked nulls. Knowledge and Information Systems (2019)
5. De Leenheer, P., Mens, T.: Using graph transformation to support collaborative ontology evolution. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) Applications of Graph Transformations with Industrial Relevance. pp. 44–58. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
6. Delanaux, R., Bonifati, A., Rousset, M.C., Thion, R.: Query-based linked data anonymization. In: International Semantic Web Conference. pp. 530–546. Springer (2018)
7. Duchi, J.C., Jordan, M.I., Wainwright, M.J.: Local privacy and statistical minimax rates. In: 54th Annual IEEE Symposium on Foundations of Computer Science. pp. 429–438. IEEE Computer Society (2013)
8. Duval, D., Echahed, R., Prost, F.: An algebraic graph transformation approach for RDF and SPARQL. In: Hoffmann, B., Minas, M. (eds.) Proceedings of the Eleventh International Workshop on Graph Computation Models, GCM@STAF 2020, Online-Workshop, 24th June 2020. EPTCS, vol. 330, pp. 55–70 (2020). https://doi.org/10.4204/EPTCS.330.4, https://doi.org/10.4204/EPTCS.330.4
9. Dwork, C.: Differential privacy. In: Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II. pp. 1–12 (2006)
10. Flouris, G., Konstantinidis, G., Antoniou, G., Christophides, V.: Formal foundations for RDF/S KB evolution. Knowl. Inf. Syst. **35**(1), 153–191 (2013)
11. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundam. Inf. **26**(3,4), 287–313 (Dec 1996)
12. Heitmann, B., Hermsen, F., Decker, S.: k-rdf-neighbourhood anonymity: Combining structural and attribute-based anonymisation for linked data. PrivOn@ ISWC **1951** (2017)
13. Kairouz, P., Oh, S., Viswanath, P.: Extremal mechanisms for local differential privacy. Journal of Machine Learning Research **17**(17) (2016), http://jmlr.org/papers/v17/15-135.html
14. Kasiviswanathan, S.P., Lee, H.K., Nissim, K., Raskhodnikova, S., Smith, A.D.: What can we learn privately? SIAM J. Comput. **40**(3), 793–826 (2011)
15. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theoretical Computer Science **109**(1–2), 181 –224 (1993)
16. Mahfoudh, M., Forestier, G., Thiry, L., Hassenforder, M.: Algebraic graph transformations for formalizing ontology changes and evolving ontologies. Knowledge-Based Systems **73**, 212 – 226 (2015)
17. Radulovic, F., García Castro, R., Gómez-Pérez, A.: Towards the anonymisation of rdf data (2015)
18. Schwentick, T.: Automata for XML - A survey. J. Comput. Syst. Sci. **73**(3), 289–315 (2007)
19. Segura, S., Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Automated Merging of Feature Models Using Graph Transformations, pp. 489–505. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

20. Shaban-Nejad, A., Haarslev, V.: Managing changes in distributed biomedical ontologies using hierarchical distributed graph transformation. Intern. Journal of Data Mining and Bioinformatics **11**(1), 53–83 (2015)
21. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **10**(5), 557–570 (2002)
22. Taentzer, G.: Agg: A graph transformation environment for modeling and validation of software. In: AGTIVE (2003)
23. Thouvenot, M., Curé, O., Calvez, P.: Knowledge graph anonymization using semantic anatomization. In: 2020 IEEE International Conference on Big Data (Big Data). pp. 4065–4074. IEEE (2020)
24. Wu, X., Ying, X., Liu, K., Chen, L.: A Survey of Privacy-Preservation of Graphs and Social Networks, pp. 421–453. Springer US (2010)
25. Zheleva, E., Getoor, L.: Privacy in social networks: A survey. In: Social network data analytics, pp. 277–306. Springer (2011)
26. Zhou, B., Pei, J., Luk, W.: A brief survey on anonymization techniques for privacy preserving publishing of social network data. SIGKDD Explorations **10**(2), 12–22 (2008). https://doi.org/10.1145/1540276.1540279, http://doi.acm.org/10.1145/1540276.1540279

Adrien Boiret, Associate Professor at INSA-CVL, LIFO Lab, Systems and Data Security research team since 2022. His current focus is in data privacy for graph databases, especially as they contain semantic information. His previous work was on formal languages for semistructured data.

Cédric Eichler obtained his Ph.D. from Toulouse III in 2015. Since 2016 he is associate professor at INSA Centre Val de Loire where he developed his interest for both technical and legal aspects of privacy. He was Data Protection Officer of INSA Centre Val de Loire from 2018 to 2022 and head of the "Security of Data and Systems" research team from 2017 to 2022. His scientific contributions lie at the intersection of graphs and privacy: from private graph computations to the sanitization of graph databases with underlying semantic.

Benjamin Nguyen is a Professor at INSA Centre Val de Loire since 2014. He received his Ph.D. from Université de Paris-Sud in 2003. His research interests focus on privacy preserving data management techniques, with currently a special focus on anonymization and reidentification techniques. He was head of the Laboratoire d'Informatique Fondamentale d'Orléans (LIFO, EA 4022) from 2016 to 2022, and member of the executive board of INSA Centre Val de Loire from 2015 to 2019.

Sara Taki, Research Engineer at INSA Centre Val de Loire, LIFO Lab, Systems and Data Security research team. She earned her Ph.D. in Computer Science from INSA Centre Val de Loire in 2023. Her research focuses on data privacy within graph databases, in particular differential privacy over RDF graphs. She holds an M.Sc degree in Information Systems and Data Intelligence from Lebanese University.

Received: April 20, 2023; Accepted: May 05, 2024.