LEMMING – Example-based Mimicking of Knowledge Graphs

Michael Röder
DICE group
Department of Computer Science
Paderborn University, Germany;
Institute for Applied Informatics
Leipzig, Germany

Pham Thuy Sy Nguyen
and Felix Conrads
and Ana Alexandra Morim da Silva
Department of Computer Science
Paderborn University, Germany

Axel-Cyrille Ngonga Ngomo
DICE group
Department of Computer Science
Paderborn University, Germany;
Institute for Applied Informatics
Leipzig, Germany

***Abstract*—The size of knowledge graphs used in real applications grows constantly. Predicting the performance of storage solutions for knowledge graphs w.r.t. their query performance is hence of central performance for the practical use of said storage solutions. We address this challenge by learning graph invariants of a given graph. We then use these invariants to fuel a stochastic generation model that is able to generate graphs of arbitrary sizes similar to the input graph. We evaluate our graph generator, dubbed LEMMING, by comparing the performance of storage solutions on synthetic and real versions of three datasets of up to 3.4×10^6 triples. Our results suggest that the performance of storage solutions on synthetic data generated by LEMMING reflects their performance on real data in most of the cases. The source code of LEMMING is available at https://github.com/dice-group/Lemming.**

I. INTRODUCTION

The size of the knowledge graphs (KGs) used in real applications grows constantly. For example, the Google Knowledge Graph grew from 3.5×10^9 facts to 18×10^9 facts in 7 months. The authors of [1] point out that comparable growth rates can be observed in KGs of other large companies. The same phenomenon is also present in open data sets. For example, DBpedia crossed the mark of 23×10^9 triples in 2017[1] while it begun with 0.1×10^9 triples in 2007 [2]. The ranking of corresponding storage solutions w.r.t. their runtime performance (measured in query mixes per hour, short QMpH) has been observed to change with the size of the KGs [3], [4]. For example, the authors of [3] report the performance of four triple stores on different versions of an RDF KG ranging from 10^5 to 10^6 triples. While BlazeGraph Free version 8.5 achieves the second-best performance on the smallest version of the KG, it achieves the worst performance on the subsequent version of the KG, which is merely five times larger. Similar insights can be derived from [4], where Jena TDB version 2.3.0 is ranked first across three triple stores and achieves the best QMpH performance on a 10% fragment of DBpedia version 2016-10 but is outperformed by Virtuoso version 7.0.0 on the full version of the same dataset and even achieves the worst QMpH performance in some high-load settings with 16 concurrent queries.

[1]https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10

Given that the performance of triple stores changes across dataset versions, there is a need to predict the future performance of storage solutions given existing versions of a dataset. Such a prediction can facilitate the deployment of reliable KG infrastructures, the timely acquisition and alteration of software components and the maintenance of quality-of-service requirements, e.g., in terms of minimal QMpH performance. In this work, we hence address the challenge of *predicting the performance and ranking of storage solutions on a (future) version of a knowledge graph of size k (measured in numbers of nodes) given previous versions of the same dataset*. This task differs from that addressed by current RDF generators, which assume a particular dataset or ontology (e.g., universities) and generate data based thereupon [5], [6], [7], [8], [9], [10].

We formalize the problem as follows: Given versions $K_1, \ldots, K_\nu$ of a KG (e.g., WikiData, DBpedia, MusicBrainz), we aim to learn a synthetic dataset generator $\mathbb{K}$ which allows the prediction of the performance and ranking of storage solutions (w.r.t. their performance measured in QMpH and in queries per second, short QpS) on a version of K of size k. We use $\mathbb{K}(k)$ to denote the KG of size k generated by $\mathbb{K}$. To learn $\mathbb{K}$, we use training data in form of $\mathscr{K} = \{K_1, \ldots, K_\nu\}$ to learn *graph-specific invariants*, which we define as functions g whose with a low variance on $\mathscr{K}$ and a high variance on other sets of graphs disjoint from $\mathscr{K}$. We learn these functions using a refinement operator ρ for arithmetic functions. We show that our operator is finite, redundant and complete. Our experiments show that our approach is able to learn generators $\mathbb{K}$ which generate datasets with which the ranking on real datasets can be approximated with a root mean squared error on ranks under 0.15.

II. RELATED WORK

The generation of synthetic graphs that can mimic real-world graphs is an important field of research. Starting from the Erdös-Renyi model [11], several further models have been developed. The Watts-Strogatz model [12] is able to create random graphs with small-world properties. The Barabasi-Albert model [13] is able to create scale-free graphs similar to the link graph of the World Wide Web. However, all these

models and their extensions aim at general graphs and do not take special features of RDF graphs into account.

The Attribute Synthetic Generator [5] mimics social networks and takes different types of edges and features of nodes into account. In a similar way, the Property Graph Model [14] takes node features and link types into account. However, both approaches are not applicable for RDF as they create undirected graphs and take only a limited set of graph features into account.

Statistical analysis for RDF datasets can be found in different publications. LODStats [15], [16] collects statistical data about more than 9 000 RDF datasets gathered from a dataset catalogue. In [17], the authors gather and analyse 3.985 million open RDF documents from 778 different domains regarding their conformity to Linked Data best practices. In [18] and [19], the authors propose a set of metrics to characterize RDF graphs and show that most of the analyzed RDF graphs have a power-law distribution with respect to the in- and out-degree distributions of their nodes.

There are several generators for RDF datasets. In [20], the authors propose an approach to generate synthetic schemas of RDF datasets. This is different to our work since we focus on the instance data and not the schema. The Lehigh University Benchmark [7] generates RDF graphs with a given number of triples describing synthetic universities, their lectures etc. The LDBC [9] generator creates RDF data describing a social network. In a similar way, SP2Bench [8] relies on the publication domain. The Waterloo SPARQL Diversity Test Suite [6] offers a data generator for scalable RDF datasets relying on the WatDiv schema. PoDiGG [10] is an RDF generator for an artificial transport network based on a given population density. While all these generators create RDF graphs, they are bound to a certain domain or ontology.

Some approaches support the generation of RDF datasets independently of the dataset's domain. Grr [21] is a generator that relies on commands written in a domain specific language describing the single steps that are necessary to generate the dataset. In contrast, gMark [22] offers a more comfortable generator for an RDF dataset and a set of queries that can be used to benchmark the dataset. However, gMark needs a large amount of statistical information about the dataset including in and out degree distributions. Similarly, LinkGen [23] relies on a given ontology and a set of parameters including distribution parameters. Apart from that, LinkGen has never been evaluated with respect to the quality of the generated graphs. In comparison, the generator proposed in this paper relies solely on the given RDF graphs without additional ontological data and gathers all statistical values that are needed for the generation process by itself. In addition, LEMMING is the first graph generation algorithm able to mimic real-world datasets by determining necessary statistics and characteristic expressions that give invariant values for the given dataset.

III. APPROACH

Our approach begins by learning graph-specific invariants g for $\mathscr{K}$ using a refinement operator ρ. Thereafter, an initial graph of size k is generated, which is further refined to meet a value within a range acceptable for g. Finally, the graph is finalised by adding literals and exporting it to RDF.

A. Preliminaries

1) Knowledge Graph: Let $\mathcal{R}$ be the set of all RDF resources, $\mathcal{B}$ the set of all RDF blank nodes, $\mathcal{P} \subseteq \mathcal{R}$ be the set of all properties and $\mathcal{L}$ the set of all RDF literals. Following [24], an RDF KG K is a set of RDF triples $(s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$. In the following, we assume that K is fully materialized.

2) Labeled Directed Multigraphs: LEMMING represents KGs as labeled directed graphs during the graph generation. Let $G = (V, E, \alpha)$ be the graph representation of K:

- $V = \{v : (v, p, o) \in K \vee (s, p, v) \in K\}$.
- $\alpha : V \to 2^{\mathcal{C}}$, where $\mathcal{C}$ is the set of all instances of `rdfs:Class` in K. This function maps each node $v \in V$ to the set of all classes (including `rdfs:Literal`) to which it belongs.
- $E = \{(u, p, v) : (u, p, v) \in K\}$.

We use G_i to denote the graph representation of K_i. When necessary, we use V_{G_i}, E_{G_i} and α_{G_i} to denote the set of vertices resp. edges or the mapping function of G_i. The graph representations of the versions of $\mathscr{K}$ form $\mathscr{G} = \{G_1, \ldots, G_\nu\}$.

B. Graph Analysis

First, the given set of graphs $\mathscr{G}$ is analysed. The density of each $G_i \in \mathscr{G}$, denoted, δ_{G_i} is determined as follows:

$$\delta_{G_i} = \frac{|E_{G_i}|}{|V_{G_i}|}. \tag{1}$$

Let $T \in 2^{\mathcal{C}}$ be a set of classes. We also determine the distribution over sets of classes $T \in 2^{\mathcal{C}}$ by calculating the probability that a vertex is an instance of exactly all classes in T. This probability is defined as

$$P_{G_i}(T) = \frac{|\{v_j | v_j \in V \wedge T = \alpha(v_j)\}|}{|V|}. \tag{2}$$

In a similar way, the probability that an edge has p as property is calculated using

$$P_{G_i}(p) = \frac{|\{e_j | e_j \in E \wedge \exists s, o \in V : e_j = (s, p, o)\}|}{|E|}. \tag{3}$$

For pairs of class sets (T_t, T_h), the probability that the vertices on the tail and the head of an edge are instances of the classes in T_t and T_h, respectively, is

$$P_{G_i}((T_t, T_h)|p) = \frac{|\{e_j | e_j \in E \wedge e_j = (s, p, o) \wedge T_t \subseteq \alpha(s) \wedge T_h \subseteq \alpha(o)\}|}{|\{e_j | e_j \in E \wedge \exists x, y \in V : e_j = (x, p, y)\}|}. \tag{4}$$

We also collect which types of triples occur in the graph, i.e., which combination of property, domain and range occur. These triples are collected in the set

$$\Omega_{G_i} = \{(T_t, p, T_h) | \exists e_j \in E \wedge e_j = (s, p, o) \wedge T_t = \alpha(s) \wedge T_h = \alpha(o)\}. \tag{5}$$

With respect to datatype properties, we collect the average number of outgoing edges $\delta_{T_j p_d G_i}$ with a datatype property p_d the instances of a class set T have. This is defined as follows:

$$\delta_{T_j p_d G_i} = \frac{|\{e_l | e_l \in E \wedge e_l = (s, p_d, o) \wedge T_j \subseteq \alpha(s)\}|}{|\{v_a | v_a \in V \wedge T_j \subseteq \alpha(v_a)\}|}. \quad (6)$$

After analysing the single graphs, the analysis results are summarised as follows.

$$\delta_{\mathscr{G}} = \frac{1}{|\mathscr{G}|} \sum_{G_i \in \mathscr{G}} \delta_{G_i} \quad (7)$$

$$P_{\mathscr{G}}(T) = \frac{1}{|\mathscr{G}|} \sum_{G_i \in \mathscr{G}} P_{G_i}(T) \quad (8)$$

$$P_{\mathscr{G}}(p) = \frac{1}{|\mathscr{G}|} \sum_{G_i \in \mathscr{G}} P_{G_i}(p) \quad (9)$$

$$P_{\mathscr{G}}((T_t, T_h)|p) = \frac{1}{|\mathscr{G}|} \sum_{G_i \in \mathscr{G}} P_{G_i}((T_t, T_h)|p) \quad (10)$$

$$\Omega_{\mathscr{G}} = \bigcup_{G_i \in \mathscr{G}} \Omega_{G_i} \quad (11)$$

$$\delta_{T_j p_d \mathscr{G}} = \frac{1}{|\mathscr{G}|} \sum_{G_i \in \mathscr{G}} \delta_{T_j p_d G_i} \quad (12)$$

In addition to that, we gather the degrees of the vertices that are instances of all classes over all graphs in $\mathscr{G}$. The degrees are used to determine the degree distribution $d_{T_j \mathscr{G}}$ for T_j. This allows the usage of different types of distributions for different T_j. For each datatype property, we gather data about the literal values the edges of this property have as object. This data is used to create a literal value distribution $\phi_{p_d \mathscr{G}}$ for each p_d.

C. Learning Graph Invariants

Our approach to learning graph invariants is based on a refinement operator ρ, which uses a specificity function as heuristic to measure the quality of arithmetic expression. In the following, we begin by presenting ρ and prove that it is finite, redundant and complete. We then present how we compute the specificity of expressions. Finally, we combine the refinement operator and the specificity function to learn graph invariants.

1) Operator: Let $\mathbb{A}(F)$ be the space of all arithmetic expressions over a finite set F of predefined real-valued functions over the set of all RDF graphs. We denote the i$^{\text{th}}$ element of F with f_i.

Example 1. We can imagine F to be the set of functions which return the minimal (m) and maximal (M) degree of the nodes in a graph.

Every arithmetic expression $\lambda \in \mathbb{A}(F)$ can be naturally represented as a tree. We say that an expression λ_1 is subsumed by an expression λ_2 (denoted $\lambda_1 \sqsubseteq \lambda_2$) iff λ_1's tree representation is a subtree λ_2's tree representation.

Example 2. $\lambda_1 = (M+m)$ is subsumed by $\lambda_2 = (M+m)/m$.

The subsumption relation defines a partial ordering over $\mathbb{A}(F)$. We now define the operator $\rho : \mathbb{A}(F) \rightarrow 2^{\mathbb{A}(F)}$ as follows:

$$\rho(\lambda) = \begin{cases} F & \text{if } \lambda \text{ is the empty expression } \epsilon, \\ \bigcup\limits_{f_i \in F} \{\lambda + f_i, \lambda - f_i, \lambda \times f_i, \lambda / f_i\} & \text{else.} \end{cases} \quad (13)$$

Example 3. Let $F = \{M, m\}$. Then $\rho(m) = \{m + m, m - m, m \times m, m/m, m + M, m - M, m \times M, m/M\}$.

We call two arithmetic expressions λ_1 and λ_2 in $\mathbb{A}(F)$ equivalent iff they return the same value for all input graphs. Based on this definition of equivalence, we can show that ρ is a finite, redundant and complete refinement operator over $(\mathbb{A}(F), \sqsubseteq)$:[2]

ρ is a refinement operator. By virtue of the construction of ρ, it is evident that $\forall \lambda \in \mathbb{A}(F)\ \forall \lambda' \in \rho(\lambda) : \lambda \sqsubseteq \lambda'$. Given that $\forall \lambda \in \mathbb{A}(F) : \epsilon \sqsubseteq \lambda$ because the tree representation of ϵ is the empty tree, we can conclude that $\forall \lambda' \in \rho(\lambda) : \lambda \sqsubseteq \lambda'$. By virtue of the definition of refinement operators [25], we can conclude that ρ is a *refinement operator*.

ρ is finite. A refinement operator τ is called *finite* if $|\tau(\lambda)| < \infty$. The *finiteness* of ρ is given by $|\rho(\lambda)| = 4|F| < \infty$ for all non-empty expressions[3] and $|\rho(\lambda)| = |F| < \infty$ for the empty expression.

ρ is redundant. We call a refinement operator τ *redundant* if at least two different sequences of application of τ can lead to the same expression. ρ is *redundant* because there are two refinement paths from ϵ to the equivalent expressions $f_1 + f_2$ and $f_2 + f_1$, i.e., $\epsilon \rightarrow f_1 \rightarrow f_1 + f_2$ and $\epsilon \rightarrow f_2 \rightarrow f_2 + f_1$.

ρ is complete. A refinement operator τ is called *complete* if it can generate an expression λ equivalent to any $\lambda' \in \mathbb{A}(F)$. The proof of ρ's *completeness* is more involved and is omitted for the sake of space. The idea behind the proof is that the completeness of ρ is a direct consequence of the set of arithmetic operators being closed w.r.t. the inversion of operators. Hence, every tree representation of an arithmetic expression can be converted into an equivalent right-linear tree, which is the set of trees generated by ρ. Hence, ρ can generate an expression $\lambda \in \mathbb{A}(F)$ equivalent to any arithmetic expression $\lambda' \in \mathbb{A}(F)$.

Example 4. Consider the expression $f_1 \times f_2 + f_3 \times f_4$. While this expression cannot be generated by ρ, the equivalent expression $(((f_1 \times f_2)/f_4) + f_3) \times f_4$ can.

2) Specificity: We can compute how characteristic an expression λ is for $\mathscr{G}$ by measuring the invariance of its values over all graphs in $\mathscr{G}$ and by comparing it with negative example graphs. We begin by using a set of graph generators $\mathfrak{G}$ for generating a set of negative examples $\mathscr{G}$' made up of $|\mathfrak{G}| \times \nu$ graphs $G'_1, \ldots, G'_{|\mathfrak{G}| \times \nu}$. $\mathfrak{G}$ can comprise any off-the-shelf graph generator. During the generation, we ensure that for each generator in $\mathfrak{G}$, $\forall i \in [1, \nu] : |V_{G_i}| = |V_{G'_i}|$. The set of negative examples is used to contrast the positive examples found in $\mathscr{G}$ during the learning of the graph-specific invariants. First, we

[2]For the sake of space, we refer the interested reader to [25] for more details on refinement operators.

[3]Note that we have exactly 4 arithmetic operators.

use the following variance-inspired measure to compute how close λ is to being an invariant of $\mathscr{G}$:

$$h(\lambda, \mathscr{G}) = 1 - \frac{\sum_{i=1}^{\nu} \sum_{j=1}^{\nu} (\lambda(G_i) - \lambda(G_j))^2}{\gamma^2 \nu(\nu - 1)}, \quad (14)$$

where $\gamma = \max\{\lambda(G_1), \ldots, \lambda(G_\nu)\}$. For invariants, $h(\lambda, \mathscr{G}) = 1$. The lower bound of the expression is 0.

h treats expressions of all lengths the same. For the sake of computational efficiency, we would want h to prefer shorter invariants over longer ones. To achieve this goal, we extend h by defining h' as follows:

$$h'(\lambda, \mathscr{G}) = h(\lambda, \mathscr{G}) - \eta|\lambda|, \quad (15)$$

where $|\lambda|$ is the number of arithmetic operators in λ and $\eta \in [0, 1]$ is a small constant.[4]

While h' captures how close λ is to being a short invariant on $\mathscr{G}$, it fails to capture how specific this expression is for $\mathscr{G}$. For example, while the expression $f_1 - f_1$ is a trivial invariant for $\mathscr{G}$, it is also an invariant for any non-empty set of graphs. We alleviate this problem by using the following function:

$$g(\lambda, \mathscr{G}, \mathscr{G}') = \frac{2h'(\lambda, \mathscr{G})(1 - h'(\lambda, \mathscr{G}')}{h'(\lambda, \mathscr{G}) + (1 - h'(\lambda, \mathscr{G}))}. \quad (16)$$

$g(\lambda, \mathscr{G}, \mathscr{G}')$ is the harmonic mean of $h'(\lambda, \mathscr{G})$ and $1 - h'(\lambda, \mathscr{G}')$ and is a measure of the specificity of λ as an invariant for $\mathscr{G}$. For $\eta = 0$, $g(\lambda, \mathscr{G}, \mathscr{G}') = 1$ if λ is an invariant of $\mathscr{G}$ (i.e., $h'(\lambda, \mathscr{G}) = 1$) and not an invariant for $\mathscr{G}'$ (i.e., $h'(\lambda, \mathscr{G}) = 0$).

3) Learning Approach: We can now learn invariants for $\mathscr{G}$ as follows. We begin by generating $\mathscr{G}'$ using an off-the-shelf graph generator based on the BA model [26]. We chose this model because it is often representative of real-world graphs. As suggested by previous on negative sampling (see, e.g., [27]), the choice of the models should not affect our results and is hence not further analysed in this work. We initialise the set of candidate expressions Λ with $\{\epsilon\}$. The set S of seen expressions is initialised with $\emptyset$. We then iterate the following three steps a predefined number of times:[5]

1) Selection: $\lambda_{\max} = \underset{\lambda \in \Lambda \setminus S}{\operatorname{argmax}}\ g(\lambda, \mathscr{G}, \mathscr{G}')$.[6]
2) Refinement: $\Lambda = \Lambda \cup \rho(\lambda_{\max})$.
3) Update: $S = S \cup \{\lambda_{\max}\}$.

Finally, we select $\underset{\lambda \in \Lambda}{\operatorname{argmax}}\ g(\lambda, \mathscr{G}, \mathscr{G}')$ as our final output.

D. Initial Graph Generation

The initial graph generation aims at creating an initial graph $\mathbb{K}(k) = (V', E', \alpha')$. To generate the graph, the number of edges is computed based on the given number of vertices k and the average density $\delta_{\mathscr{G}}$. After that, the classes and properties are assigned to the vertices and edges based on the class and property distributions, respectively. Finally, the edges are used to connect the vertices. This is done by applying the following three steps for each edge. First, the set of possible classes for the head and the tail of the edge are determined. Second, the classes of the two endpoints (T_t and T_h) of the edge are chosen from these sets. Third, two instances are chosen which will be connected by the edge from the two sets of vertices that are instances of the chosen classes.

1) Class set selection: This first step uses the previously collected constraints $\Omega_{\mathscr{G}}$. Let p_x be a property and $q_t = (p_x, T_h)$ be a function that returns a set of classes whose vertices are potential tails of edges with Property p_x and a head v_h with $\alpha(v_h) = T_h$. Let $q_t = (p_x, T_h)$ be a similar function for potential head classes.

$$q_t(p_x, T_h) = \{T_t | (T_t, p_x, T_h) \in \Omega_{\mathscr{G}}\} \quad (17)$$

$$q_h(T_t, p_x) = \{T_h | (T_t, p_x, T_h) \in \Omega_{\mathscr{G}}\} \quad (18)$$

Both functions can be used as $q_t(p_x, \cdot)$ and $q_h(\cdot, p_x)$ where $\cdot$ donates any set of classes.

2) Endpoint class definition: We propose three different approaches for selecting the set of classes of the two endpoints of a given edge.

The approach *Uniform Class Selection (UCS)* randomly draws T_t from $q_t(p_x, \cdot)$ using a uniform distribution. In the same way, T_h is chosen from $q_h(T_t, p_x)$.

The approach *Biased Class Selection (BCS)* relies on the $P_{\mathscr{G}}((T_t, T_h)|p)$ probabilities of the different class sets to sample the class sets for the tail and head of the edge. For each set of classes $T_i \in q_t(p_x, \cdot)$ the probability $P_{\mathscr{G}}((T_t, \cdot)|p)$ is used. It is determined as follows:

$$P_{\mathscr{G}}((T_i, \cdot)|p) = \sum_{T_j \in q_h(T_i, p_x)} P_{\mathscr{G}}((T_i, T_j)|p). \quad (19)$$

Based on these probabilities, a class set T_t is sampled for the tail of the edge. Based T_t, a set of classes is sampled for the head of the edge. For each possible class set $T_i \in q_h(T_t, p_x)$ the probability $P_{\mathscr{G}}((T_t, T_i)|p)$ is used for that.

While UCS and BCS are sampling T_t before T_h, the *Clustered Class Selection (CCS)* samples both class sets at the same time. For each possible class set pair (T_i, T_j) with $(T_i, p_x, T_j) \in \Omega$, its probability $P_{\mathscr{G}}((T_i, T_j)|p)$ is used.

3) Vertex selection: After the classes of the tail and head vertices of the edge are chosen, the two single vertices with these classes have to be chosen. Let α'^{-1} be the inverse function of α', i.e., a function that returns for a given set of classes T the set of vertices that are instances of these classes: $\alpha'^{-1}(T_i) = \{v_j | v_j \in V', T_i \subseteq \alpha'(v_j)\}$. For sampling two vertices, the *Uniform Instance Selection (UIS)* assigns a uniform probability to all vertices of the sets $\alpha'^{-1}(T_t)$ and $\alpha'^{-1}(T_h)$, respectively.

In contrast, the *Biased Instance Selection (BIS)* approach uses the $d_{T\mathscr{G}}$ distributions to assign degree weights to the single vertices. For each vertex $v_j \in V'$, a degree weight w_j is sampled from $d_{\alpha'(v_j)\mathscr{G}}$. Based on these weights, a probability $P(v_j|T_i)$ is assigned to each vertex to be chosen

[4] We set $\eta = 0.1$ in all experiments.

[5] In our experiments, we use 50 iterations.

[6] Given that ρ is redundant, we exploit the commutativity and the associativity of some arithmetic operations to detect and remove duplicate expressions from E in our implementation. We omit details for the sake of space.

when sampling a vertex for a given set of classes T_i. The probability is defined as

$$P(v_j|T_i) = w_j / \sum_{v_l \in \alpha'^{-1}(T_i)} w_l. \quad (20)$$

The chosen vertices are connected by the given edge. However, if both vertices are already connected with an edge that has the same property p_x two new vertices have to be sampled. By combining the three approaches for selecting the tail and head classes for an edge with the two techniques to select the single vertices, six approaches are obtained: UCS-UIS, UCS-BIS, BCS-UIS, BCS-BIS, CCS-UIS and CCS-BIS.

E. Graph Amendment

The initial graph is further amended based on the set of characteristic expressions determined on the set of original graphs. To this end, we define an error score that is used to measure the difference between the values of the invariant expressions for the original graphs $\mathscr{G}$ and the generated graph H. Let Λ_{max} be the set of the best invariant expressions learned on $\mathscr{G}$ as described in Section III-C. Let μ_i be the average value the expression λ_i returns for the original graphs and let σ_i be its standard deviation. Let $\Delta(H, \lambda_i, \mu_i, \sigma_i)$ be the difference function defined as follows:

$$\Delta_i = \frac{(\lambda_i(H) - \mu_i)^2}{\sigma_i^2}. \quad (21)$$

Let $\varepsilon(H)$ be the error of graph H with respect to the characteristic expressions defined as follows:

$$\varepsilon(H) = \sum_{i=1}^{|\Lambda_{max}|} \Delta(H, \lambda_i, \mu_i, \sigma_i). \quad (22)$$

The target of the amendment phase is to optimise for the error score of the graph H by successive modifications. We achieve this goal by using a greedy approach. In each iteration, the algorithm generates two new versions of H by adding or removing a random edge, respectively. Thereafter, the graph with the lower error score is used for the next iteration. The amendment phase ends when a maximum number of iterations is reached or no improvement has been achieved for several iterations. The removal of an edge randomly chooses an edge and removes it. The addition of a new edge starts with choosing a property p_x for the edge following the property distribution. Based on the chosen property, the same steps as during the generation of the initial graph are executed to assign tail and head vertices to the newly generated edge.

F. Graph Completion

The completion phase takes the result graph H of the amendment phase as input and extends it to form the final, complete graph. First, datatype edges are created. For each set of classes T_j and each datatype property p_d, the number of edges with p_d the instances of these classes should have is determined by multiplying the number of instances with the average degree $\delta_{T_j p_d \mathscr{G}}$. Second, for each datatype edge, a literal is generated by sampling a literal value from the previously learned distribution v_d. Third, the datatype edges are connected to a resource node within the graph. This is done by sampling a vertex of the set of instances of T_j.

Finally, the graph is transformed into an RDF graph representation. To this end, each resource vertex of the graph receives a generated URI. With these URIs, the graph can be transformed into an RDF triple representation. After that, the `rdf:type` triples are generated, i.e., for each vertex $v_j \in V'$ and $c \in \alpha'(v_j)$ an RDF triple v_j `rdf:type` c using the URIs of v_j and c, respectively.

IV. Evaluation

We evaluate our graph generation approach based on three different real-world datasets and four different triple stores. The main aim of our evaluation is to measure the performance of the selected triple stores based on our generated datasets and compare it with that achieved by the same triple stores on an unseen version of the dataset.

A. Experimental setup

1) Overview: The three datasets we use, i.e., Semantic Web Dog Food (SWDF), Linked Geo Data (LGD) and the International Chronostratigraphic Chart (ICC), are such that at least three different versions are available. We use the latest version of each dataset as held-out graph and its size as input parameter k for our generation algorithm. We use the six versions of our approach and compare it with a baseline algorithm to generate graphs. Since all approaches are based on sampling mechanisms, we execute each algorithm three times. After that, we evaluate four reference triple stores—Virtuoso, Blazegraph, Fuseki and GraphDB—on the held-out as well as the generated datasets using Iguana [4].[7] Iguana is a generic SPARQL Benchmark execution framework. It can be used to benchmark different triple stores with different datasets in a comparable way. During the benchmarking, we measure the query mixes per hour (QMpH) and queries per second (QpS). QpS is measured for each query while QMpH summarizes the overall performance of a triple store over all queries. The similarity between the measured values is calculated using the Spearman rank correlation (RC) for the QMpH values and the root mean squared error (RMSE) for the QpS values.

2) Datasets: SWDF comprises data about semantic web conferences from 2001 to 2015.[8] The data mainly focuses on persons, events, papers and organisations related to these conferences. Since the dataset is designed to build one version upon the previous version we define it to have 15 versions—one version per year. Each version comprises the previous version and the data of the conferences of the next year. The last version of 2015 is the held-out version. The *LGD* dataset is a subset of the Linked Geo Dataset [28]. We use the `Military` and `Craft` files of the three consecutive

[7]The stores are available at https://github.com/openlink/virtuoso-opensource/releases, https://blazegraph.com/, https://jena.apache.org and https://graphDB.ontotext.com/, respectively.

[8]https://old.datahub.io/dataset/semantic-web-dog-food

TABLE I
FEATURES OF THE TARGET GRAPHS OF THE DIFFERENT DATASETS

	SWDF	LGD	ICC
Triples	445 821	3 387 842	12 742
Resources (k)	45 423	591 649	1 423
Queries	20	43	27

TABLE II
SET OF METRICS F USED FOR THE SEARCH OF INVARIANT EXPRESSIONS.

Metric	Description
#edges	The number of edges.
#vertices	The number of vertices.
avgDegree	Average vertex degree.
maxInDegree	The highest in-degree of a vertex found in the graph.
maxOutDegree	The highest out-degree of a vertex found in the graph.
stdInDegree	Standard deviation of the vertex in-degrees.
stdOutDegree	Standard deviation of the vertex out-degrees.
#eTriangles	Number of unique triangles formed by three edges.
#vTriangles	Number of unique triangles formed by three vertices.

versions of 2013, 2014 and 2015. The latter is used as held-out version. We apply an inference step to get infer missing `rdf:type` triples based on domain and range definitions of properties from the dataset's ontologies. Table I lists the features of the target graphs of the different datasets. The third dataset, *ICC*, represents the chronostratigraphic chart as RDF [29], [30], [31], [32]. This chart defines the geological time intervals including their names, start and end dates as well as their relations to each other. The dataset has been updated several times leading to twelve versions in the years 2004–2018. All versions of all three datasets are preprocessed by materializing all implicit knowledge that can be infered based on the ontology of the datasets. Table I shows the features of the target graphs.

We use LSQ [33] to retrieve real user queries to the datasets from query logs. We use FEASIBLE [34] to generate benchmark queries from the LSQ queries, which can be used to benchmark the triple stores based on the different datasets. Table I shows the number of queries generated for the different datasets. For each dataset, the ontology is retrieved. If a dataset makes use of more than one ontology, the intersection of the ontologies is used. For each query, every URI that is not contained in the respective ontology (i.e., each URI that is neither a class nor a property) is replaced by a template variable [4]. Iguana replaces these variables on the fly with resources from the graph used for benchmarking. This leads to several queries with different resources. It is ensured that only queries with a non-empty result are used for the benchmarking. This allows the usage of queries comprising instance URIs although the target graph as well as the generated graphs have different instance URIs.

3) Configuration: Table II shows the set F, i.e., the set of metrics that are used to learn the invariant expressions of the input graphs. The refinement operator is configured to use 50 iterations for its search with $\eta = 0.1$. As a set of graph generators $\mathfrak{G}$ for negative examples $\mathscr{G}'$ we use generators for star, ring, grid, clique and bipartite graphs. Our algorithm is configured to use a maximum of 50 000 iterations to reduce the error score during the amendment phase. The phase ends earlier if the error score does not improve for 5000 iterations. Further, we configure the CCS approaches to rely on Poisson distributions for $d_{T\mathscr{G}}$. The distribution parameters are learned for each T individually.

Depending on the datatype of literals, we configure the algorithm to use different literal value distribution types. For properties p_d with literals that have a numeric, `Date` or `DateTime` datatype, we determine the minimum and maximum values y_{min} and y_{max}, respectively. After that, we define $\phi_{p_d\mathscr{G}}$ as uniform distribution of the range $[y_{min}, y_{max}]$. All other literals are treated as datatype string. For the generation of such literals, we define a distribution that always returns a new string making all string-based literals unique.

4) Baseline: An analysis of the datasets showed that they do not have a common type of degree distribution, i.e., it is not possible to assign them to a common class of graphs like scale-free or Poisson graphs. However, since it has been shown that the degree distributions of a large number of RDF datasets follow a power-law distribution [18], [19] we decided to use an implementation of the Barabasi-Albert model [26]. This algorithm adds one node after the other to the graph by creating $\delta_{\mathscr{G}}$ new directed edges. The direction of the edge is sampled from a Bernoulli distribution with the probability 0.5 for both cases. The second vertex for each edge is sampled based on the degree of the vertices, i.e., the higher the degree, the higher the probability that a vertex is chosen. After the generation of the graph, the properties and node types are sampled from $P_{\mathscr{G}}(p)$ and $P_{\mathscr{G}}(T)$, respectively.

TABLE III
INVARIANT CHARACTERISTIC EXPRESSIONS PER DATASET.

	ID	Expression
SWDF	λ_1	maxInDegree / ((#vertices × stdDevOutDegree) + maxInDegree)
	λ_2	maxInDegree / (#vertices + maxInDegree − stdDevOutDegree)
	λ_3	maxInDegree / ((#vertices / maxInDegree) + maxInDegree)
	λ_4	maxInDegree / (#edges + maxInDegree − stdDevOutDegree)
	λ_5	maxInDegree / ((#vertices / stdDevOutDegree) + maxInDegree)
LGD	λ_1	(2× #vertices − #edges) / (#edges × avgDegree)
	λ_2	#vertices / (#edges × avgDegree^2)
	λ_3	(#vertices − #edges) / (#edges + maxOutDegree − #vertices)
	λ_4	#vertices / (#edges × (avgDegree − 1.0))
	λ_5	(#vertices − #edges) / (#edges + maxInDegree − #vertices)
ICC	λ_1	maxInDegree / ((#edges − #vertices) + maxInDegree)
	λ_2	maxInDegree / ((#vertices / maxOutDegree) + maxInDegree)
	λ_3	maxInDegree / ((#vertices − #edges) + maxInDegree)
	λ_4	maxInDegree / ((#vertices × stdDevInDegree) + maxInDegree)
	λ_5	maxInDegree / ((#vertices / maxInDegree) + maxInDegree)

B. Results

Tables III and IV summarize the results of the graph generation process. Table III shows the graph invariants per dataset, which clearly differ across datasets. A comparison of the values of the graph invariants for the original graphs, the target graph and the generated graphs are shown in

TABLE IV

AVERAGE RESULTS OF THE DIFFERENT EXPRESSIONS ON THE ORIGINAL GRAPHS, AND DIFFERENCE OF THE AVERAGE VALUES ON THE TARGET GRAPH AND THE GENERATED GRAPHS IN PERCENTAGES OF THE ORIGINAL GRAPHS' AVERAGE. THE LAST TWO LINES OF THE RESULTS ON A DATASET CONTAIN THE AVERAGE ERROR SCORES $\varepsilon(H)$ OF THE GENERATED GRAPHS AND THE AVERAGE RUNTIMES. *,**,***—1, 2 OR ALL 3 RUNS TERMINATED BEFORE REACHING THE MAXIMUM NUMBER OF ITERATIONS, RESPECTIVELY.

	Exp.	Original graphs	Target graph	UCS UIS	UCS BIS	BCS UIS	BCS BIS	CCS UIS	CCS BIS	BL
SWDF	λ_1	0.0926	-16.05%	-61.86%	-63.24%	3.48%	0.00%	0.00%	0.00%	-99.18%
	λ_2	-0.1751	-15.72%	0.79%	0.69%	-0.59%	-0.09%	-0.09%	-0.09%	-99.33%
	λ_3	0.9997	0.03%	0.03%	0.03%	0.03%	0.03%	0.03%	0.03%	-35.10%
	λ_4	0.1294	-11.98%	0.77%	0.69%	-0.26%	0.12%	0.12%	0.12%	-99.09%
	λ_5	0.9965	0.15%	0.28%	0.29%	-0.11%	-0.11%	0.00%	-0.06%	-35.50%
	Error $\varepsilon(H)$			0.3076	0.3195	0.0034	0.0011	**0.0002**	0.0004	1.4299
	Runtime (in h)			3.4	3.5	* 2.9	3.5	*** 0.9	*** 0.8	0.0
LGD	λ_1	-0.1250	-0.06%	0.02%	0.03%	0.02%	0.02%	-0.03%	0.02%	0.03%
	λ_2	0.0160	-11.17%	-1.00%	-1.43%	-0.96%	-0.13%	4.19%	-0.07%	-4.00%
	λ_3	-0.9978	0.12%	-0.04%	-0.07%	0.17%	0.17%	-0.02%	0.21%	0.18%
	λ_4	0.0851	-8.77%	-0.77%	-1.11%	-0.74%	-0.09%	3.26%	-0.04%	-3.11%
	λ_5	-0.7857	0.68%	0.04%	0.00%	0.04%	-0.03%	-0.43%	-0.04%	27.22%
	Error $\varepsilon(H)$			**0.0008**	0.0014	0.0038	0.0035	0.0046	0.0051	6.7174
	Runtime (in h)			** 36.1	* 56.0	48.4	** 49.1	*** 35.3	** 51.7	0.1
ICC	λ_1	0.0797	15.63%	0.04%	0.04%	0.04%	0.04%	0.37%	0.04%	-87.79%
	λ_2	0.9936	-0.03%	-0.11%	0.00%	0.00%	0.00%	0.00%	-0.17%	-14.74%
	λ_3	-0.0948	19.05%	-0.04%	-0.04%	-0.04%	-0.04%	0.35%	-0.04%	-89.54%
	λ_4	0.0168	3.02%	0.00%	-0.05%	-1.18%	-0.05%	-5.85%	-0.23%	-56.29%
	λ_5	0.9975	0.02%	0.04%	0.06%	0.04%	0.04%	0.04%	0.03%	-14.18%
	Error $\varepsilon(H)$			0.0026	0.0020	0.0008	**0.0007**	0.0022	0.0056	124.8374
	Runtime (in s)			***290.7	***153.3	***190.0	***111.3	505.3	***218.0	0.6

Table IV. The difference between the values of the invariants for original graphs and the target graphs are low for most of the graph invariants we learned. These results corroborate the assumption of the existence of graph invariants for RDF datasets.

Table IV also shows the overall error $\varepsilon(H)$ and the runtimes of the different graph generation approaches are given. Note that for all three datasets the baseline leads to the generation of graphs with the highest error score $\varepsilon(H)$. A comparison of the errors ε of our generation approaches (see Table IV) suggests that none is better overall. Still, our results suggest that the different approaches for selecting the tail and head classes for an edge (UCS, BCS and CCS) have a higher influence on the overall error than the technique to select the single vertices (UIS and BIS). With respect to runtime, all three approaches take several hours for the generation of larger graphs. As expected, the runtimes are shorter for the small ICC graph. The majority of the time is used in the amendment phase. Hence, some approaches lead to shorter runtimes if the amendment phase is stopped earlier after 5000 iterations without any improvement.

Table V shows a summary of the triple store evaluation, i.e., the rank correlation of the QMpH values and the RMSE for the QpS values. The QMpH values suggest that the benchmark on SWDF is harder than that on LGD. On both target graphs, Virtuoso shows the best performance with 1,018 and 1,460 QMpH respectively. In contrast, ICC seems to be less hard since all triple stores achieve values up to 434,911 QMpH (GraphDB). The results in Table V suggest that our approaches show a much better performance than the baseline for the hard SWDF dataset. For the LGD dataset, all generators achieve the same ranking of the triple stores. However, the average RMSE value of the baseline is significantly higher.[9] This is caused by much higher runtimes of the benchmark queries on the BL graphs than on the target graph. For the ICC dataset, the prediction of the order of the triple stores seems to be trivial as well. However, because of the higher QpS values achieved by all triple stores, a small difference in the query runtime leads to large differences in the calculated QpS values and, hence, to large RMSE values. Although the UCS-BIS approach achieves the smallest RMSE value, its difference to the baseline as well as several other approaches is not significant.[10] Overall, our results suggest that LEMMING is consistently better than the off-the-shelf approach. In addition, the differences across the benchmarks propound that the difference in the performance of LEMMING and the baseline is positively correlated with the difficulty of the benchmark.

V. CONCLUSION

In this paper, we presented LEMMING, a graph generator for creating graphs that mimic a given, real-world RDF dataset. We proposed the usage of graph invariants and a refinement operator that is able to find these invariants based on a given set of graph metrics. Further, we proposed six different

[9]We use a Wilcoxon signed rank test with $\alpha = 0.1\%$.

[10]We use a Wilcoxon signed rank test with $\alpha = 2\%$.

TABLE V
RANK CORRELATION (RC) OF THE TRIPLE STORE SYSTEMS BASED ON THEIR QMPH ON THE GENERATED GRAPHS COMPARED TO THE RANKING ON THE TARGET GRAPH AND AVERAGE RMSE VALUES OF THE QPS VALUES MEASURED ON THE TARGET GRAPH AND THE GENERATED GRAPHS.

Approach	SWDF		LGD		ICC	
	RC	RMSE	RC	RMSE	RC	RMSE
UCS-UIS	0.87	81.1	1.00	**111.8**	1.00	280.9
UCS-BIS	0.93	82.0	1.00	115.3	0.93	**219.6**
BCS-UIS	0.93	88.2	1.00	115.3	1.00	261.2
BCS-BIS	0.93	**64.1**	1.00	117.6	1.00	242.8
CCS-UIS	1.00	72.7	1.00	117.3	1.00	255.6
CCS-BIS	0.93	95.8	1.00	115.9	0.93	229.0
BL	0.32	170.5	1.00	159.1	0.93	222.6

approaches for the generation of a graph with a given size that abides to the determined graph invariants. Our evaluation showed that LEMMING is able to generate graphs that lead to similar benchmarking results as the real-world graph while a comparable baseline struggled to achieve this for all datasets.

Our future work is to improve the runtime of LEMMING. Thereafter, we plan to use it to generate large graphs to evaluate the scalability of triple stores.

ACKNOWLEDGEMENTS

This work has been supported by the German Federal Ministries for Economic Affairs and Energy (BMWi), and for Education and Research (BMBF) within the projects RAKI (no. 01MD19012D) and DAIKIRI (no. 01IS19085B).

REFERENCES

[1] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: Lessons and challenges," *Queue*, vol. 17, no. 2, pp. 48–75, 2019.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*. Springer, 2007, pp. 722–735.

[3] V. Papakonstantinou, I. Fundulaki, and G. Flouris, "Second version of the versioning benchmark," in *Holistic Benchmarking of Big Linked Data*, 2018.

[4] F. Conrads, J. Lehmann, M. Saleem, M. Morsey, and A.-C. Ngonga Ngomo, "IGUANA: a generic framework for benchmarking the read-write performance of triple stores," in *The Semantic Web ISWC 2017*.

[5] A. M. Ali, H. Alvari, A. Hajibagheri, K. Lakkaraju, and G. Sukthankar, "Synthetic generators for cloning social network data," in *Proceedings of the Fifth ASE International Conference on Big Data/SocialInformatics/PASSAT/BioMedCom*, 2014.

[6] 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.

[7] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Journal of Web Semantics*, vol. 3, no. 2, 2005.

[8] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "Sp2bench: a sparql performance benchmark," in *Proceedings of the ICDE*, 2009.

[9] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz, "The LDBC social network benchmark: interactive workload," in *Proceedings of the ACM SIGMOD*, 2015.

[10] R. Taelman, P. Colpaert, E. Mannens, and R. Verborgh, "Generating public transport data based on population distributions for rdf benchmarking," *Semantic Web Journal*, Jul. 2018.

[11] P. Erdös and A. Rényi, "On random graphs. i," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[12] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'smallworld' networks," *Nature*, vol. 393, pp. 440–442, 1998.

[13] A.-L. Barábasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, 1999.

[14] A. V. Sathanur, S. Choudhury, C. Joslyn, and S. Purohit, "When labels fall short: Property graph simulation via blending of network structure and vertex attributes," in *Proceedings of the CIKM*, 2017.

[15] S. Auer, J. Demter, M. Martin, and J. Lehmann, "Lodstats – an extensible framework for high-performance dataset analytics," in *Knowledge Engineering and Knowledge Management*, 2012, pp. 353–362.

[16] I. Ermilov, J. Lehmann, M. Martin, and S. Auer, "Lodstats: The data web census dataset," in *The Semantic Web – ISWC*, 2016, pp. 38–46.

[17] A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker, "An empirical survey of linked data conformance," *Journal of Web Semantics*, vol. 14, pp. 14–44, 2012.

[18] J. D. Fernández, M. A. Martínez-Prieto, P. de la Fuente Redondo, and C. Gutiérrez, "Characterizing rdf datasets," *Journal of Information Science*, vol. 1, pp. 1–27, 2016.

[19] M. Zloch, M. Acosta, D. Hienert, S. Dietze, and S. Conrad, "A software framework and datasets for the analysis of graph measures on rdf graphs," in *The Semantic Web*. Springer, 2019, pp. 523–539.

[20] Y. Theoharis, Y. Tzitzikas, D. Kotzinos, and V. Christophides, "On graph features of semantic web schemas," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 5, pp. 692–702, 2008.

[21] D. Blum and S. Cohen, "Generating rdf for application testing," in *Proceedings of the ISWC-PD*. CEUR-WS.org, 2010.

[22] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat, "gMark: Schema-Driven Generation of Graphs and Queries," in *Proceedings of the ICDE*, 2017.

[23] A. K. Joshi, P. Hitzler, and G. Dong, "Linkgen: Multipurpose linked data generator," in *The Semantic Web – ISWC 2016*. Cham: Springer International Publishing, 2016, pp. 113–121.

[24] A.-C. Ngonga Ngomo, S. Auer, J. Lehmann, and A. Zaveri, "Introduction to linked data and its lifecycle on the web," in *Reasoning Web International Summer School*. Springer, 2014.

[25] P. R. van der Laag and S.-H. Nienhuys-Cheng, "Completeness and properness of refinement operators in inductive logic programming," *The Journal of Logic Programming*, vol. 34, no. 3, 1998.

[26] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, 2002.

[27] S. Stergiou, Z. Straznickas, R. Wu, and K. Tsioutsiouliklis, "Distributed negative sampling for word embeddings," in *AAAI*, 2017.

[28] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "Linkedgeodata: A core for a web of spatial open data," *Semantic Web Journal*, vol. 3, no. 4, pp. 333–354, 2012.

[29] S. Cox, "Rdf representation of 2016 edition of international chronostratigraphic chart (geologic timescale) v1," CSIRO, Data Collection, 2017.

[30] ——, "Rdf representation of 2017 edition of international chronostratigraphic chart (geologic timescale) v3," CSIRO, Data Collection, 2018.

[31] S. Cox and S. Richard, "Rdf representation of international chronostratigraphic chart (geologic timescale) v2," CSIRO, Data Collection, 2014.

[32] ——, "Rdf representation of 2018 edition of international chronostratigraphic chart (geologic timescale) v1," CSIRO, Data Collection, 2019.

[33] M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A.-C. Ngonga Ngomo, "LSQ: the linked SPARQL queries dataset," in *The Semantic Web ISWC 2015*, 2015.

[34] M. Saleem, Q. Mehmood, and A.-C. Ngonga Ngomo, "Feasible: A Feature-Based SPARQL Benchmark Generation Framework," in *The Semantic Web ISWC 2015*, 2015.