SHACTOR: Improving the Quality of Large-Scale Knowledge Graphs with Validating Shapes

Kashif Rabbani
Aalborg University
Denmark
kashifrabbani@cs.aau.dk

Matteo Lissandrini
Aalborg University
Denmark
matteo@cs.aau.dk

Katja Hose
Aalborg University, Denmark
TU Wien, Austria
khose@cs.aau.dk

ABSTRACT

We demonstrate SHACTOR, a system for extracting and analyzing validating shapes from very large Knowledge Graphs (KGs). Shapes represent a specific form of data patterns, akin to schemas for entities. Standard shape extraction approaches are likely to produce thousands of shapes, and some of those represent spurious constraints extracted due to the presence of erroneous data in the KG. Given a KG having tens of millions of triples and thousands of classes, SHACTOR parses the KG using our efficient and scalable shapes extraction algorithm and outputs SHACL shapes constraints. The extracted shapes are further annotated with statistical information regarding their support in the graph, which allows to identify both erroneous and missing triples in the KG. Hence, SHACTOR can be used to extract, analyze, and clean shape constraints from very large KGs. Furthermore, it enables the user to also find and correct errors by automatically generating SPARQL queries over the graph to retrieve nodes and facts that are the source of the spurious shapes and to intervene by amending the data.

CCS CONCEPTS

• **Information systems** → **Semantic web description languages**; *Data cleaning*; • **Computing methodologies** → Knowledge representation and reasoning; Ontology engineering.

KEYWORDS

Knowledge Graphs, SHACL, Shapes Extraction, Quality Assessment

ACM Reference Format:
Kashif Rabbani, Matteo Lissandrini, and Katja Hose. 2023. SHACTOR: Improving the Quality of Large-Scale Knowledge Graphs with Validating Shapes. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3555041.3589723

1 INTRODUCTION

Knowledge Graphs (KGs) are in widespread use both within companies and on the Web [5, 9], thanks to their ability to represent a wide range of information in different domains. DBpedia (dbpedia.org) and WikiData (wikidata.org) are examples of large public KGs, where data is stored using the Resource Description Framework (RDF) [1], which by design allows modeling data without a schema definition (unlike, for instance, the relational model). In RDF, data is modeled as nodes representing entities and data values, while edges represent relationships and attributes (see the example in Figure 1a).

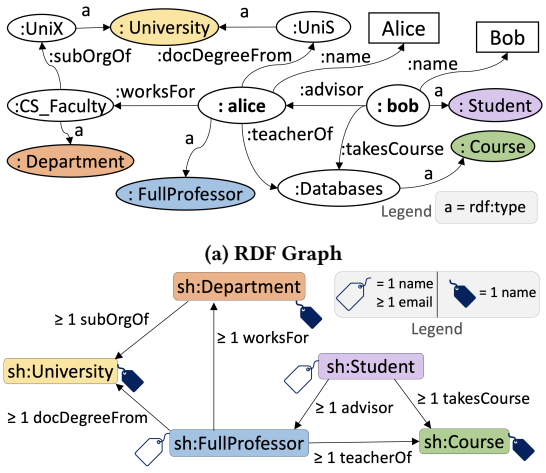

Figure 1: An example RDF Graph and Validating Shapes

Nonetheless, as more and more data is accrued within KGs, practical applications impose further demands regarding quality assessment and validation. To provide a way to validate the contents of a KG, shape constraint languages, namely SHACL [4] and ShEx [6], have been proposed as ways to define and enforce constraints using so-called *validating shapes*. Validating shapes allow to define a partial schema for the entities end relationships contained in a KG and overcome some of the limitations of the RDF Schema specification while being easier to use than OWL ontologies. Shapes can be used to express that an entity of type Student needs to have a name, an advisor, and should be enrolled in some courses; and that these attributes should be instances of type string, FullProfessor, and Course, respectively (see Figure 1b for a simplified depiction of some validating shapes describing the relationships between some entity types). Thanks to their simplicity and expressivity, shape constraint languages have attracted increasing interest, and SHACL became a W3C standard in 2017 [4].

Validating shapes have attracted substantial attention in the past few years, and recently, we conducted a survey [7] to analyze the extraction and adoption of validating shapes in industry and academia. Data scientists are faced with the challenge to *craft a set of validating shapes for already existing KGs* that they are working with and then use those to clean such datasets, i.e., a *post-hoc* validation.

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06…$15.00
https://doi.org/10.1145/3555041.3589723

Hence, we face the need to develop semi-automatic methods to help users generate shapes for large existing KGs. These are usually the shape extraction approaches [2, 3]. Due to well-known issues in data quality and the heterogeneity of existing KGs, spuriousness poses important challenges to automatic shape extraction methods. For example, in DBpedia, a few entities representing musical bands are wrongly assigned to dbo:City class. As a consequence, when shapes are extracted from its instance data using approaches that do not analyze the support of the shapes, the resulting node shape for dbo:City specifies that cities are allowed to have dbo:genre and dbo:formerBandMember properties. We call these *spurious shapes*.

Existing shape extraction approaches produce many spurious shapes when dealing with erroneous triples or incomplete data [8]. Thus, extraction approaches will generate tens of thousands of shape constraints due to the effect of *spuriousness*. In these situations, it becomes *unmanageable for domain experts to manually analyze and clean* the resulting validating shapes. Finally, *existing methods are not scalable*, i.e., unable to extract shapes from very large KGs (especially on commodity machines). Therefore, we proposed a solution called *Quality Shapes Extraction (QSE)* [8] to tackle both the limitations of *scalability* and *spuriousness* in existing shapes extraction approaches. QSE extracts validating shapes from very large graphs on a commodity machine (it takes only 3 minutes on DBPedia with just 16 GB of RAM) and also provides information about the reliability of the extracted shape constraints by computing their *confidence* and *support*. Hence, QSE identifies those shapes that are the most informative and distinguishes those that are indeed affected by incomplete or incorrect data.

Contributions. In this work, we show how the shapes extracted with QSE in combination with the information about their confidence and support enable a wide range of data profiling and cleaning functionalities, beyond simple validation. We thus propose SHACTOR (for SHapes extrACTOR), a tool that data scientists can use *to speed up the end-to-end KG cleaning process* by (1) automatically extracting shapes, (2) *helping to evaluate their quality*, (3) providing *important structural profiling information*, and (4) allowing *to find errors and missing data* in the given KG as well as correct such issues by *automatically generating and executing SPARQL queries*. SHACTOR uses QSE [8] to filter shapes by various confidence and support thresholds to identify reliable shapes. These shapes provide essential information on the structure and content of the KG as well as on possible data quality issues. SHACTOR then highlights the spurious shapes and automatically generates SPARQL queries to extract the erroneous or incomplete data generating such shapes. The system allows for removing the erroneous triples interactively and then correcting the generated shapes. Thus, the tool will lead to a valid set of shapes, which can be used to maintain the quality of the given KG in the future.

Demo Video and Source Code. The demonstration video and source code of SHACTOR is available on our website and GitHub[1].

2 SHACTOR

SHACTOR provides a graphical user interface for interacting with our QSE algorithm [8] and provides a wide range of new functionalities. Given a KG in RDF, SHACTOR uses QSE to extract a full set

[1]https://relweb.cs.aau.dk/qse/shactor/, https://github.com/dkw-aau/demo-shactor

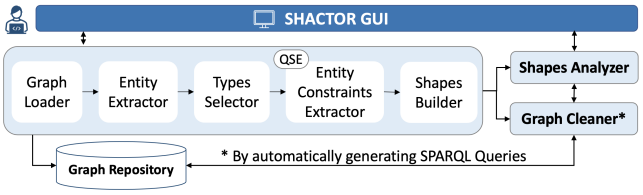

Figure 2: SHACTOR Architecture

of validating shapes and then allows the user to explore and analyze this output. Hence, SHACTOR consists of three main phases, (1) shapes extraction with support and confidence, (2) shapes analysis, and (3) KG cleaning. Figure 2 shows SHACTOR's architecture.

Background and Problem. An RDF knowledge graph models entities and their relationships in the form of <subject, predicate, object> triples. Given pairwise disjoint sets of IRIs $\mathcal{I}$, blank nodes $\mathcal{B}$, and literals $\mathcal{L}$, an RDF Graph $\mathcal{G}$:$\langle \#, \; \rangle$ is a graph with a finite set of nodes $\# \subset (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ and a finite set of edges $\subset \{\langle \mathcal{B}, ?, > \rangle \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})\}$. See a sample $\mathcal{G}$ in Figure 1a, where oval and rectangular shapes represent IRIs and literal nodes, respectively. The standard query language for RDF data is SPARQL. A SPARQL query $\&$ consists of a set of triple patterns along the conditions that have to be met in order for data in $\mathcal{G}$ to contribute to the result.

The SHACL shapes of $\mathcal{G}$ are defined as set of node shapes $\mathcal{S}$:$\{\langle B, g_B, \; _B \rangle, """\}$, where B is the shape *name*, $g_B \in C$ is the *target class*, and $_B$ is a set of property shapes of the form q_B:$\langle g_p, T_p, C_p \rangle$, where $g_p \in \mathcal{P}$ is called the target property, $T_p \subset \mathcal{I}$ contains either an IRI defining a *literal type*, e.g., xsd:string, or a set of IRIs – called *class type constraint*, and C_p is a pair $(=, <) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\}).=\leq <$ – called min and max *cardinality constraints*. For example, the node shape for :Student in $\mathcal{G}$ (Figure 1a) is $\{B$:studentNodeShape, g_B : Student, $_B$:$\{q$:name, q:takesCourse, q:advisor$\}\}$ where property shape q:name has constraints $\langle$:name, xsd:string, $(1, \infty)\rangle$, i.e., every node of type Student should have at least one name of type string (see Figure 1b).

In QSE, we introduced the notion of *support* and *confidence* for shape constraints to study the reliability of extracted shapes and tackle the issue of spuriousness. These concepts are inspired by the well-known theory developed for the task of frequent pattern mining. The *support* l of a constraint measures how many entities are conforming to a specific constraint q_B:$\langle g_p, T_p, C_p \rangle \in \; _B$ of a shape $\langle B, g_B, \; _B \rangle \in \mathcal{S}$ appearing in the data graph $\mathcal{G}$. Similarly, the *confidence* γ of a constraint q_B measures the ratio between how many entities conform to q_B and the total number of entities that are instances of the target class of the shape B.

Given the need to improve the quality of the data within an existing KG, SHACTOR *addresses the problem of helping users to produce high-quality validating shapes and use them to analyze and correct the data quality issues present in the graph.* Hence, to extract shapes from a large existing graph $\mathcal{G}$ while tackling the effects of spuriousness and helping the user to *analyze* $\mathcal{S}$ and *clean* $\mathcal{G}$, the SHACTOR system takes a knowledge graph $\mathcal{G}$, a threshold l for support, and γ for confidence and produces all shapes after distinguishing for which subset of node shapes $\langle B, g_B, \; _B \rangle \in \mathcal{S}$ it holds that $\text{supp}(B) > l$ and for which property shapes q_B:$\langle g_p, T_p, C_p \rangle \in \; _B$, $\text{supp}(q_B) > l$ and $\text{conf}(q_B) > \gamma$. Moreover, for each $_B \in \mathcal{S}$, SHACTOR generates a SPARQL query $\&$ to fetch either triples that conform to some $q_B \in \; _B$ or triples that fail to conform to it, allowing

to inspect specific data quality issues. Thus, SHACTOR is a system that *takes full advantage of existing standards* in terms of data formats, query languages, and validation constraints in the realm of KGs, while also ensuring ease of use and scalability.

Shapes Extraction and Analysis. SHACTOR parses $\mathcal{G}$ (available as SPARQL endpoint or from file) to extract entities and their constraints and computes support and confidence for each constraint. To focus on the specific subset of the graph, it also allows selecting a custom subset of types in the KG, where the user can select one or more classes to focus only on entities of the specific class, e.g., :Student or :FullProfessor class in Figure 1a. In the second step, it asks to input thresholds for support l and confidence γ. Given this information, it will use the QSE algorithm to parse $\mathcal{G}$ and produce all shapes that satisfy the support and confidence thresholds while also producing shape constraints that fail to meet these conditions (See for instance **1** in Figure 3). The data scientists can further dynamically filter the produced shapes by providing more restrictive values for l and γ. They can further fine-tune the pruning thresholds by using statistical information like the frequency of each class. Note that while a spurious shape does not usually represent a valid constraint to be enforced, it usually signals the presence of erroneous data in the graph. Hence, on the one hand, SHACTOR provides information w.r.t. the contents and structure of the KG, helping in identifying a set of reliable validating shapes to be enforced, while, on the other hand, by looking at the shapes with low support and confidence, it also helps in understanding which portions of the data may be particularly problematic. *To the best of our knowledge, SHACTOR is the first tool that considers both support and confidence to identify errors in a KG.*

Therefore, to further inspect the shapes produced in this way, SHACTOR implements several useful features. First, it displays interactive charts that show which percentage of shapes are above and below each threshold (see **2** in Figure 3). Furthermore, in the list of node shapes, i.e., for each node shape B in $\mathcal{S}$, it shows the support l_{B} of B along with the quality of the data behind each of them in terms of the number of property shapes specific for that node shape (i.e., $|\ _{\mathit{B}}|$ of B) that are above the two thresholds l and γ (see **3** in Figure 3). Node shapes with many property shapes with low support and confidence signal the presence of noisy or incomplete data for entities of that type.

Moreover, for each property shape $q_{\mathit{B}} \in\ _{\mathit{B}}$ of a given node shape $\mathit{B} \in \mathcal{S}$, it shows which predicate and type constraints it involves (the values of $\langle g_{\mathsf{p}} \cdot \mathsf{T}_{\mathsf{p}} \cdot \mathsf{C}_{\mathsf{p}} \rangle$) along with its specific support l_{q_s} and quality indicator of each property shape q_{B} computed by visually showing the confidence of the shape and highlighting those property shapes whose confidence is below the user-provided threshold (see **4** in Figure 3). These quality indicators and highlights based on support and confidence of each node shape and its property shapes help the user find the spurious shape constraints in $\mathcal{S}$.

Improving Data Quality. The result of validating a KG using validating shapes is usually a validation report that lists all entities and triples that violate the given set of constraints expressed by the shapes. Nonetheless, as we have seen, when using an automatic shape extraction tool, the data scientist needs to decide which subset of shapes to enforce among those extracted by the tool. SHACTOR helps data scientists by automatically generating queries to retrieve the entities and triples that caused a given shape to be extracted from the data. In a sense, it allows to inspect the *provenance* of a shape. This helps the user both in deciding which shapes to be used for validation as well as to identify either erroneous triples or missing information when inspecting spurious shapes.

Specifically, for a given node shape $\mathit{B} \in \mathcal{S}$, SHACTOR can build SPARQL queries for each property shape $q_{\mathit{B}} \in\ _{\mathit{B}}$ using the property path $g_?$, class type constraint T_{p} of q_{B}, and target class g_{B}. Hence, given the node shape B with property shapes $_{\mathit{B}}$, SHACTOR automatically builds a SPARQL query to retrieve all the triples having property path $g_?$ and class type g_{B} for a given $q_{\mathit{B}} \in\ _{\mathit{B}}$, e.g., to inspect which entities of type dbo:City in DBpedia have the genre attribute (see **5** in Figure 3). SHACTOR can then directly execute the query and return the corresponding list of triples (see **6** in Figure 3). Further, SHACTOR provides the option to generate the queries to delete selected triples, thus allowing to automatically delete erroneous triples on the spot, as well as to generate INSERT queries to amend the missing information in the graph.

3 DEMONSTRATION SCENARIO

SHACTOR demonstrates the power and versatility of a tool that effectively exploits the information carried by shapes extracted and annotated with statistical data. We show how data scientists can use SHACTOR to speed up the process of KG cleaning by automatically extracting shapes and evaluating the quality of extracted shapes along with their provenance. First, the demonstration guides the participants through different extraction phases (not shown in the figure). We use a full snapshot of DBpedia from 2021, which contains 52 M triples, 15 M literals, 5 M typed entities, 1.3 K properties, and 427 classes. Moreover, participants will also be able to analyze shapes from a full snapshot of WikiData (having more than 1.9 billion triples) that we have extracted in advance. SHACTOR can extract shapes on a commodity machine also for WikiData [8], but while for DBpedia, it takes only a few minutes, for WikiData, it takes too long for a live demonstration of the extraction.

Configuration. At first, we present various options to provide a KG as input, e.g., the user can upload a KG file or point to a SPARQL endpoint. *The audience is also welcome to bring their own KG to be analyzed.* Then, the user selects the target sub-graph (from the list of extracted classes); after that, SHACTOR starts the shapes extraction step and the user is directed to the main analysis interface (Figure 3). SHACTOR also supports uploading already extracted shapes.

Shapes Analysis. The user provides support and confidence as pruning thresholds to analyze the extracted shapes. Hence, SHACTOR applies the input pruning thresholds and display an overview in the form of pie charts (see **1** and **2** in Figure 3). These charts show the portions of node and property shapes that are below or above the provided pruning thresholds. Furthermore, they help the user decide the optimal values for the pruning thresholds. In our DBpedia example, we see the quality indicators for two node shapes (shown in **3** of Figure 3). Then, we show an example node shape called :CityShape having target class dbo:City and explore its property shapes (shown in **4** of Figure 3). The list of shape constraints can be sorted by increasing or decreasing support so that the user can select the desired set of node or property shapes. The selected