Mining Validating Shapes for Large Knowledge Graphs via Dynamic Reservoir Sampling

Matteo Lissandrini[1,2,*], Kashif Rabbani[2] and Katja Hose[3]

[1]University of Verona, Italy
[2]Aalborg University, Denmark
[3]TU Wien, Austria

Abstract

Knowledge Graphs (KGs) are databases that model knowledge from heterogeneous domains using the graph data model. Shape constraint languages have been adopted in KGs to ensure their data quality. They encode the equivalent of a schema in the Resource Description Framework (RDF). Unfortunately, few KGs are accompanied by a corresponding set of validating shapes. When validating shapes are missing, the solution is to extract them from the graph via mining techniques. Current shape extraction methods are often incomplete, not scalable, and generate spurious shapes. Thus, in this discussion paper, we present our recent contribution: a novel Quality Shapes Extraction (QSE) method for large graphs. QSE computes confidence and support for shape constraints via a novel Dynamic Reservoir Sampling method, enabling the identification of informative and reliable shapes. QSE is the first method (validated on WikiData and DBpedia) to extract a complete set of shapes from large real-world KGs.

Keywords

Knowledge Graphs, Data Mining, Data Quality

1. Introduction

Knowledge Graphs (KGs) are databases of collections of triples using the Resource Description Framework (RDF) [1] and represented in the form $\langle subject, relation, object \rangle$. They are in widespread use both within companies [2, 3, 4] and on the Web [5, 6]. Yet, their heterogeneous nature and the semi-automatic way in which they are built (usually by crawling data from many sources) leads to important issues of data quality [7, 8, 9]. To face this situation, shape constraint languages such as SHACL [10] and ShEx [11] have been designed to offer a way to enforce validation rules. In practice they work as schema languages for KGs, for this reason, they are generally called *validating shapes*. Consider a simple KG representing entities of type Student (see Figure 1), a validating shape would enforce the requirement for these entities for a name, a registration number, and the enrollment in some courses. The shape would also specify that these attributes should be of type string, integer, and Course, respectively. In an ideal scenario, validating shapes should be manually specified by domain experts before data is inserted into the database. Nonetheless, to specify validating shapes for already-existing large-scale KGs, this manual process is often very cumbersome, so data curators need tools that can speed up this process [8]. To address this need, a few tools have been proposed to produce

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy
*Corresponding author.
matteo.lissandrini@univr.it (M. Lissandrini); kashifrabbani@cs.aau.dk (K. Rabbani); katja.hose@tuwien.ac.at (K. Hose)

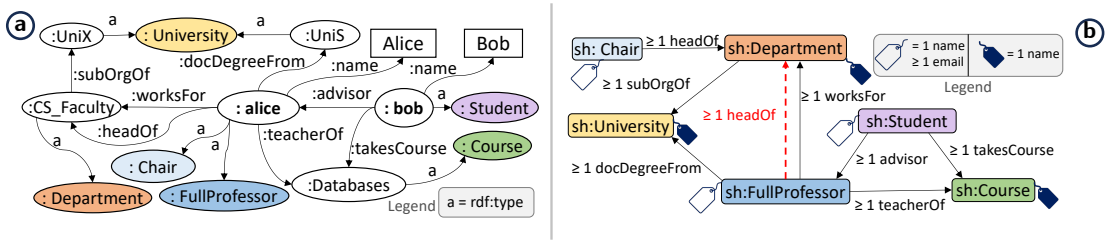

Figure 1: An example RDF Graph (a) and Validating Shapes (b)

automatically [12, 13, 14, 15] or semi-automatically [16, 17, 18] a set of validating shapes for a target KG. In our work [19], we identify 3 important limitations of existing systems: (1) they produce incomplete shapes as they support a very limited family of constraints, e.g., they do not support extraction of cardinality constraints; (2) their extraction process is easily affected by errors and inconsistencies in the KG, e.g., if some departments, by mistake, have attached the property hasAdvisor, a corresponding *spurious shape* is extracted; and above all (3) they do not scale to large KGs, e.g., they take days to process just a subset of WikiData.

Among these issues, we verified that spuriousness poses important challenges to automatic shape extraction methods [19]. For instance, in one of the snapshots of DBpedia [20] we analyzed, some of the entities representing musical bands were wrongly assigned to the class dbo:City because of some faulty automatic text-matching process. As a consequence, when shapes are extracted from this dataset using existing approaches, the resulting node shape for dbo:City specifies that cities are allowed optional properties like dbo:genre and dbo:formerBandMember. Therefore, our position is that the key to solving these challenges passes through *efficient extraction algorithms that take into account the prevalence of shapes.* These algorithms should extract shapes annotated also with the amount of entities and triples that satisfy them (in mining terms we talk of support and confidence [21]). Further, they should be able to do so even on commodity machines. Due to the very restricted nature of validating shapes (i.e., they are not arbitrary graph patterns), we propose a highly optimized mining algorithm (QSE-Exact). Then, to tackle the issue of *scalability* in extremely large KGs, we devised a novel sampling process, called Dynamic Reservoir Sampling, which allows us to propose QSE-Approximate. *As a result, QSE can filter out shapes affected by spurious or erroneous data based on robust and easily understandable measures.* QSE allows shape extraction both from KGs available as files as well as SPARQL endpoints. Finally, our efficient approximation algorithm enables shape extraction even on a commodity machine by sampling the KG entities via a dynamic multi-tiered reservoir sampling technique. Our experiments on real-world KGs further prove that our sampling strategy is accurate and efficient.

2. RDF Shapes and the QSE Problem

The standard model for encoding KGs is the Resource Description Framework (RDF [1]), which describes data as a set of $\langle s, p, o \rangle \in \mathcal{G}$ triples stating that a subject s is in a relationship with an object o through predicate p. In these triples subjects, predicates, and objects can be IRIs ($\mathcal{I}$) or (only for objects) literal values. Moreover, we distinguish two special subsets of the IRIs $\mathcal{I}$:

predicates $\mathcal{P}$ and classes $\mathcal{C}$. The set of predicates $\mathcal{P} \subset \mathcal{I}$ is the subset of IRIs that appear in the predicate position p in any $\langle s, p, o \rangle \in \mathcal{G}$. Among predicates $\mathcal{P}$, we identify the type predicate $\mathbf{a} \in \mathcal{P}$, which corresponds to IRI rdf:type [22] or wdt:P31 WikiData [6], as the predicate that connects all entities that are instances of a class to the node representing the class itself, i.e., their type. Thus, all the IRIs that are classes in $\mathcal{G}$ form the subset $\mathcal{C}: \{c \in \mathcal{I} | \exists s \in \mathcal{I}$ s.t. $\langle s, \mathbf{a}, c \rangle \in \mathcal{G}\}$.

Given a KG $\mathcal{G}$, a set of *validating shapes* represents integrity constraints in the form of a shape schema $\mathcal{S}$ over $\mathcal{G}$. Since the shape schema describes shapes associated with node types and their connections to other attributes and other node types, we can also visualize the shape schema $\mathcal{S}$ as a particular type of graph (see Figure 1b). Therefore, in the following, we refer to two concepts: the *data graph* $\mathcal{G}$ and the *shape graph* derived from $\mathcal{S}$. The *data graph* is the RDF graph $\mathcal{G}$ to be validated, while the *shape graph* consists of constraints in the form of the shape schema $\mathcal{S}$ against which entities of the data graph are validated. These constraints are defined using node and property shapes. In the following, we adopt the previously defined syntax [23] to refer to the set $\mathcal{S}$ according to the SHACL *core constraint components* [24]. Finally, without loss of generality, we focus on the current standard for SHACL shapes in the following, but our methods can be applied to ShEx via converters [25].

When validating a graph $\mathcal{G}$ against a shape schema $\mathcal{S}$ having a node shape $\langle s, \tau_s, \Phi_s \rangle \in \mathcal{S}$, where s is the shape for the *target class* $\tau_s \in \mathcal{C}$ and Φ_s defines which properties each instance of τ_s can or should be associated with, we verify that each entity $e \in \mathcal{G}$ that is an instance of τ_s satisfies all the constraints Φ_s. Note that we use the term entity and node interchangeably throughout the paper.

Shapes Extraction. Here we study the case where $\mathcal{G}$ is given, and we want to extract the set of validating shapes $\mathcal{S}$ that validates every class in $\mathcal{C}$ from $\mathcal{G}$. This is the *shapes extraction* problem. In this case, existing automatic approaches [8] assume the graph to be correct, then iterate over all entities in it, and extract for each entity e all necessary shapes that validate e. The union of all such shapes is assumed to be the final schema $\mathcal{S}$. This is useful when we want to validate new data that will be added *in the future* to the KG so that it will conform to the data already in the graph. Unfortunately, this approach will produce spurious shapes. For instance, in Figure 1, since :alice has both type Full Professor and Chair, when parsing the triple (:alice, :headOf, :CS_Faculty), the property shape headOf (the red dotted arrow in Figure 1.b) is assigned to both node shapes, instead of assigning it to the Chair node shape only.

Shapes Support and Confidence. To contrast the effect of spuriousness, we want to exploit statistics on how often properties are applied to entities of a given type. Therefore, we introduce the notion of *support* and *confidence* for shape constraints to study the reliability of extracted shapes. These concepts are inspired by the well-known theory developed for the task of frequent patterns mining [26]. In our approach, a property shape corresponds to a node- and edge-labeled graph pattern. Thus, given the shape $s: \langle s, \tau_s, \Phi_s \rangle \in \mathcal{S}$ its support is the number of entities that are of type τ_s, while the support of a property shape $\phi_s: \langle \tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p} \rangle \in \Phi_s$ is the cardinality of entities conforming to it. Finally, the confidence of a constraint ϕ_s measures the ratio between how many entities conform to ϕ_s and the total number of entities that are instances of the target class of the shape s (for brevity, we leave out the full computation and formalization [19]).

In practice, as it happens in the case of frequent pattern mining [26], when extracting validating shapes, the support ($\mathrm{supp}(\phi_s)$) provides insights on how frequently a constraint is matched in the graph, i.e., the number of entities e satisfying a constraint ϕ_s. While similar

to the task of itemset mining [27], the confidence ($\mathrm{conf}(\phi_s)$) can tell us how strong is the association between a node type and a specific constraint, i.e., the proportion of entities e satisfying a constraint ϕ_s among all the entities that are instances of the node type τ_s of $s \in \mathcal{S}$. For instance, the confidence for property shape headOf (Figure 1.b) in our snapshot of LUBM is 10% for the Full Professor node shape and 100% for Chair, which indicates a strong association of the headOf property shape to latter and a weak association to the former.

Given the need to extract shapes from a large existing graph $\mathcal{G}$, we formally define the problem of extracting high-quality shapes from KGs as follows:

Problem 1 (Quality Shapes Extraction). *Given an RDF graph $\mathcal{G}$, a threshold ω for support, and ε for confidence, the problem of quality shapes extraction over $\mathcal{G}$ is to find the set of shapes $\mathcal{S}$ such that for all node shapes $\langle s, \tau_s, \Phi_s \rangle \in \mathcal{S}$ it holds that $\mathrm{supp}(s) > \omega$ and for all property shapes $\phi_s : \langle \tau_p, T_p, C_p \rangle \in \Phi_s$, $\mathrm{supp}(\phi_s) > \omega$ and $\mathrm{conf}(\phi_s) > \varepsilon$.*

3. QSE-Exact: Exact Shape Extraction algorithm

Extracting shapes $\mathcal{S}$ from an RDF graph $\mathcal{G}$ requires processing its triples and analyzing the types of nodes involved. At a high level, we need to know for each entity all its types, these will become node shapes, and then for each entity type, identify property shapes, which requires, in turn, knowing the types of the objects as well. Furthermore, we need to keep frequency counts to know how often a specific property connects nodes of two given types compared to how many entities exist of those types. Therefore, the extraction of shapes $\mathcal{S}$ from an RDF graph $\mathcal{G}$ involves a computation over all of its triples. During this process, the system surveys which types are associated with all distinct subjects and objects within these triples. Further, for each entity type, property shapes must be determined, this involves examining the predicates associated to pairs of subject and object types. This process thus maintains frequency counts to establish the prevalence of a specific property connecting nodes of two particular types relative to the total number of entities of those types.

In our solution [19], this is done in four steps (see Figure 2): (1) entity extraction, (2) entity constraints extraction, (3) support and Confidence computation, and (4) shapes extraction. These steps apply similarly to both cases when the graph is stored as a complete dump on a single file or stored within a triplestore [19]. Here, for simplicity, we focus on the file-based approach. The endpoint version of the algorithm requires instead the execution of multiple SPARQL queries to extract equivalent information.

QSE-Exact (file-based). One of the most common ways to store an RDF graph $\mathcal{G}$ on a file $\mathbf{F}$ is to represent it as a sequence of triples. Therefore, QSE reads $\mathbf{F}$ line by line and processes it as a stream of $\langle s, p, o \rangle$ triples. In the entity extraction phase, the algorithm parses each $\langle s, p, o \rangle$ triple containing a type declaration (e.g., rdf:type or wdt:P31 – this can be configured) and for each entity, it stores the set of its entity types and the global count of their frequencies, i.e., the number of instances for each class in maps Ψ_{ETD} (Entity-to-Data) and Ψ_{CEC} (Class-to-Entity-Count), respectively. For example, Figure 2 (phase 1) presents two example entities :bob and :alice (from the example graph of Figure 1) having entity types :Student, :FullProfessor, and :Chair, respectively. Figure 2 also presents the structure of the Entity-to-Data Ψ_{ETD} dictionary map to help understand the captured entities and their information. In the second phase, i.e., entity

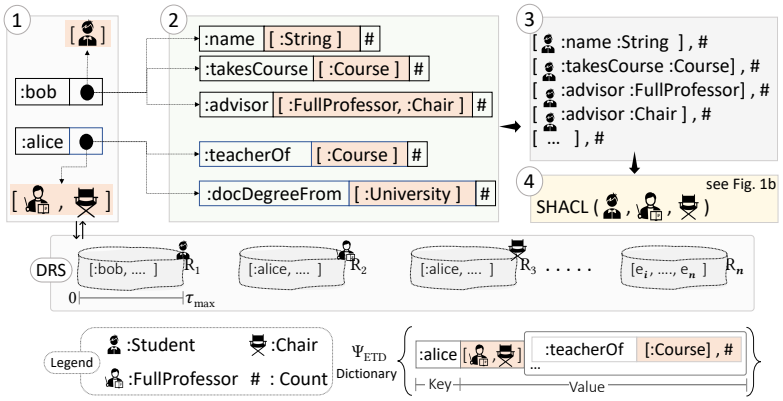

Figure 2: Overview of the four phases of QSE: ① entity extraction, ② entity constraints extraction, ③ support and confidence computation, and ④ shapes extraction. QSE-Approximate uses Dynamic Reservoir Sampling (DRS) in ①.

constraints extraction, the algorithm performs a second pass over $\mathbf{F}$ to collect the constraints and the meta-data required to compute support and confidence of each candidate property shape. Specifically, it parses all triples except triples containing type declarations (which can be skipped now) to obtain for each predicate the subject and object types from the map Ψ_{ETD} that was populated in the previous step. The type of a literal object is inferred from the value, and for a non-literal object is obtained from Ψ_{ETD}. For example, Ψ_{ETD} records that the types of :alice are :FullProfessor and :Chair. Then, the Entity-to-Property-Data map Ψ_{ETPD} is updated to add the candidate property constraints associated with each subject entity. Figure 2 (phase 2) shows the meta-data captured for the properties of :bob and :alice.

In the third phase, i.e., for support and confidence computation, the constraints' information stored in maps (Ψ_{ETD}, Ψ_{CEC}) is used to compute support and confidence for specific constraints. The algorithm iterates over the map Ψ_{ETD} to get the inner map Ψ_{ETPD} mapping entities to candidate property shapes ϕ_s:$\langle \tau_{\text{p}}, \mathsf{T}_{\text{p}}, \mathsf{C}_{\text{p}} \rangle \in \Phi_s$, and retrieves the type of each entity using types information stored in Ψ_{ETD} to build triplets of the form $\langle \tau_e, \tau_p, \tau_{p_o} \rangle$ and compute their support and confidence. Figure 2 (phase 3) highlights some of these triplets for τ_e =:Student. The value of support and confidence for each distinct triplet is incremented in each iteration and stored in Ψ_{Supp} and Ψ_{Conf} maps. Additionally, a map Ψ_{PTT} (Property to Types) is populated with distinct properties' frequencies and their object types to establish the corresponding min/max cardinality constraints.

Finally, in the shapes extraction phase, the algorithm iterates over the values of the Ψ_{CTP} map and defines the *shape name* of s, the *shape's target definition* τ_s, and the set of *shape constraints* ϕ_s for each candidate class. The set of property shapes P for a given *Node Shape* are then extracted from the map $\textsc{Map}\langle\textsc{Property}, \textsc{Set}\rangle$. An example shapes graph for our running example is shown in Figure 1. The C_{p} constraint can possibly have three types of values: sh:Literal, sh:IRI, and sh:BlankNode. In the case of literal types, the literal object types such as xsd:string, xsd:integer, or xsd:date are used. However, in the case of non-literal object types, the constraint sh:class is used to declare the type of object to define the type of value for the candidate property. It is possible to have more than one value for the sh:class and sh:datatype constraints of a candidate property shape, e.g., to state that a property can accept both integers

and floats as values, in such cases, we use sh:or constraint to encapsulate multiple values. A more detailed explanation of each phase is available in the extended version of the paper[1].

Cardinality Constraints. QSE supports assigning cardinality constraints (sh:minCount and sh:maxCount) to C_p to each property shape constraint ϕ_s:$\langle \tau_p, T_p, C_p \rangle$. Following the open-world assumption, all shape constraints are initially assigned a minimum cardinality of 0, making them optional. However, in some cases, certain properties must be mandatory (min count: 1), while others should appear exactly once per entity (i.e., should be assigned both a min and a max count equal to 1). Trivially one can assign minimum cardinality 1 to property shapes having confidence 100%, i.e., for those cases in which all entities have that property. In case of incomplete KGs, QSE allows users to provide a different confidence threshold value for adding the min cardinality constraints. To achieve this, we extend the fourth phase and add a min cardinality constraint for property shapes based on the min-confidence provided by the user. QSE also keeps track of properties having maximum cardinality equal to 1 in a second phase and assigns sh:maxCount=1 to those property shapes in the fourth phase of shapes extraction.

Our analysis [19] shows that QSE-Exact requires only two passed over the file containing the triples. On the other hand, QSE-Exact keeps type and property information for each entity in memory while extracting shapes. As a result, its memory requirements are prohibitively large when dealing with very large KGs. Therefore, we propose QSE-Approximate to enable shape extraction from very large KGs with reduced memory requirements.

4. QSE-Approximate

QSE-Approximate *solves the scalability issue in shapes extraction approaches by employing a sampling technique.* Thanks to this technique, we are able to drastically reduce the memory requirements of QSE-Exact. Thus, QSE-Approximate is based on a multi-tiered dynamic reservoir-sampling (DRS) algorithm. Reservoir sampling is a technique for selecting a random sample of items from a stream of data. Given a sample size fixed in advance it assumes each item has an equal probability of being chosen. In our algorithm we maintain as many reservoirs as types in the graph. Yet, entities have a very skewed distribution, so fixing the size of each reservoir in advance leads to a huge memory waste, as most reservoir remain almost empty. Our method, instead, dynamically resizes each reservoir as new triples are parsed. Moreover, the replacement of nodes in the reservoir is performed based on the number of node types across reservoirs. The resulting algorithm replaces the first phase of QSE. After sampling, the information about the sampled entities is used in the same way as before in the remaining phases of our exact algorithm. Hence, we maintain in memory information only for a representative sample of entities, forming an induced sampled graph, enough to detect all shapes.

QSE-Approximate receives as input a graph file $\mathbf{F}$, sampling percentage ($\text{Sampling}_{\%}$), and maximum size of the reservoir per class (τ_{max}). After initialization, triples t of $\mathbf{F}$ are parsed and filtered based on whether they contain a type declaration. From these, we extract the entities to populate the Entity-to-Data map Ψ_{ETD}, while non-type triples are parsed to keep count of distinct properties in the Property-Count map Ψ_{PC}. For instance, :alice is an entity of type :FullProfessor and :Chair in Ψ_{ETD} shown in Figure 2. QSE-Approximate maintains a reservoir for

[1] https://relweb.cs.aau.dk/qse/

each distinct entity type e_t, e.g., maintaining a distinct reservoir of entities of type :Student (R_1), :FullProfessor (R_2), and :Chair (R_3) shown in Figure 2, using a map of sampled entities per class (Ψ_{SEPC}). The reservoir capacity map (Ψ_{RCPC}) stores the current max capacities for the reservoir for each e_t. If e_t does not exist in Ψ_{SEPC} and Ψ_{RCPC}, i.e., if it has not a reservoir, one is created. Then, e is inserted in the reservoir for e_t, e.g., :alice is inserted into both reservoirs R_2 and R_3 shown in Figure 2. If the reservoir has reached its current capacity limit, we may have to replace an entity in the reservoir with the current one. Hence, neighbor-based dynamic reservoir sampling is performed, i.e., a random number r is generated between zero and the current number of type declarations read from $\mathbf{F}$. If r falls within the reservoir size, then a node in the reservoir is replaced with e. To select which node to replace, we identify as $\widehat{n}$ the target node at index r, and with $\overleftarrow{n}$ and $\vec{n}$ its neighbors at indexes $r-1$ and $r+1$, respectively. Among these, the node having minimum scope (i.e., the minimum number of types that are known at this point in time) is selected to be replaced by the current e. Additionally, the algorithm keeps track of actual Class-to-Entity-Count in Ψ_{CEC}, i.e., the exact count of how many entities of each type we have seen. Once the reservoir for e_t is updated, we compute the proportion of entities sampled so far with type e_t over the total number of entities of that type seen up to now. Given the current and target sampling ratio ($\text{Sampling}_{\%}$) provided as input, the algorithm evaluates whether to resize the reservoir for e_t, if it has not already reached the limit τ_{max}.

While performing shapes pruning using counts over sampled entities, QSE-Approximate requires to estimate actual support $\overline{\omega}_\phi$ and confidence $\overline{\varepsilon}_\phi$ of a property shape ϕ from the current values ω and ε computed from the sampled data. To estimate the effective support for a property shape ϕ we employ the formula $\overline{\omega}_\phi{=}\omega_\phi/min(|P_r^*|/|P|, |T_r|/|T|)$, where ω_ϕ is the support computed for ϕ in the current sample, P represents all triples in $\mathcal{G}$ having property τ_p, P_r^* represents triples having property τ_p across all entities in all reservoirs, T represents all entities of type e_t in $\mathcal{G}$, and T_r represents all entities of type e_t in the reservoir. Similarly, the confidence $\overline{\varepsilon}_\phi$ of a property shape is estimated by dividing by $|T_r|$ the estimated support.

Space Analysis. QSE-Approximate's space complexity depends on the values of target $\text{Sampling}_{\%}$, the maximum reservoir size τ_{max}, and the number of entity types |T| in $\mathcal{G}$. In the worst case, it requires $O(2{\cdot}|T|{\cdot}\tau_{max})$, therefore while $\mathcal{G}$ can contain hundreds of millions of entities, we can still easily estimate how many distinct types are in the graph and select τ_{max} to fit the available memory.

5. Evaluation and Discussion

We selected a synthetic dataset, LUBM-500 [28], and three real-world datasets: DBpedia [20] downloaded on 01.10.2020; YAGO-4 [5], for which we use the subset containing instances from the English Wikipedia, downloaded on 01.12.2020; and WikiData [6], in two variants, i.e., a dump from 2015 [29] (Wdt15), used in the original evaluation of SheXer [13], and the truthy dump from September 2021 (Wdt21) filtered by removing non-English strings. Among these, Wdt21 is the largest and contains almost 2B triples, with 82K node types, and 9K property types. We have implemented QSE algorithms in JAVA-11. All experiments are performed on a single machine with 16 cores and 256 GB RAM. Yet, we also test the algorithm performance in a resource constrained environment. The full experimental setup of QSE is available online [30].

Table 1
Running Time (T) in minutes (m) and hours (h) along with Memory (M) consumption in GB.

		DBpedia		LUBM		YAGO-4		Wdt15		Wdt21	
		T	M	T	M	T	M	T	M	T	M
F	SheXer	26 m	18	58 m	33	1.9 h	24	3.2 h	59	-	Out_M
	QSE-Exact	3 m	16	8 m	16	23 m	16	16 m	50	2.5 h	235
	QSE-Approx	**1 m**	10	**2 m**	10	**13 m**	10	**13 m**	16	**1.3 h**	32
Q	SheXer	9 h	65	15 h	140	Out_T	-	13 h	180	Out_T	-
	QSE-Exact	34 m	16	47 m	16	2.4 h	16	1.2 h	16	Out_T	-
	QSE-Approx	**16 m**	6	**3 m**	7	**39 m**	16	**49 m**	16	**5.7 h**	64

QSE-Exact. We initially considered SheXer [13], ShapeDesigner [16], and SHACLGEN [12] as state-of-the-art approaches [8] to compare against QSE. Yet, from our initial experiments, we verified their current implementations cannot handle large KGs with more than a few million triples and do not manage to extract shapes of KGs having more than some hundreds of classes. Therefore, in the following, we focus our comparison on SheXer. Table 1 shows the running time and memory consumption to extract shapes for all datasets using File (**F**) and Query-based (**Q**) variants of SheXer, QSE-Exact, and QSE-Approximate. Among the *file-based* approaches, QSE-Exact is 1 order of magnitude faster than SheXer for all datasets. Further, it consumes up to 50% less memory than SheXer. We note that SheXer goes out of memory (Out_M) for Wdt21.

When looking at the shapes produced after pruning via support and confidence [19], as expected, the results show that the higher we set the threshold for support and confidence, the higher the percentage of PSc and PS to be pruned. Precisely, DBpedia contains $11K$ Property Shapes (PS), with $38K$ non-literal and $5K$ literal Property Shape constraints (PSc), when QSE performs pruning with confidence >25% and support ≥ 1, it prunes out 99% PSc and PS. Similarly for Wdt21, QSE prunes 85% non-literal and 97% literal constraints, and 66% PS for confidence >25% and support $\geq$1. A manual inspection showed us that there was a very high correlation between high support shapes and shapes that should be valid in the KG.

QSE-Approximate. Table 1 shows how QSE-Approximate reduces the memory requirements of the exact approach by allowing users to specify the *sampling percentage* ($Sampling_{\%}$, S% for short) and maximum limit of the *reservoir size* (τ_{max}), i.e., the maximum number of entities to be sampled per class, to reduce the number of entities to keep in memory. Here (with $\tau_{max} = 1000$ and S%=100%), to extract shapes from Wdt21, QSE-Approximate required almost half the time with 1 order of magnitude less memory than QSE-Exact, while SheXer could not complete the computation. Similarly, among *query-based* approaches, QSE-Approximate proved to be the only approach to extract shapes from the Wdt21 endpoint in 5.7 hours with 64 GB memory consumption. In contrast, QSE-Exact and SheXer timed out (24 hours).

Practical Implications of QSE. We tested the practical utility of QSE by evaluating the correctness of extracted shapes and their effect when used to validate the KG [19]. The results of this analysis showed that QSE extracts shapes with 100% precision in terms of correct shapes constraints that should be part of the final set of shapes (qualified as quality shapes) by removing spurious shape constraints. Further, we used these 10 shapes, extracted by QSE, to validate DBpedia using a SHACL validator and found 20,916 missing triples and 155 erroneous triples. The detailed results of this analysis are contained in the extended version[1]. Overall, this experiment shows that by using our technique the user is provided with a refined set of valid shapes that can effectively identify errors in the KG.

References

[1] W. Consortium, RDF 1.1, https://w3.org/RDF/, 2014. Accessed 6th May, 2024.

[2] S. Schmid, C. Henson, T. Tran, Using knowledge graphs to search an enterprise data lake, in: The Semantic Web: ESWC 2019 Satellite Events - ESWC, volume 11762 of *Lecture Notes in Computer Science*, Springer, Portorož, Slovenia, 2019, pp. 262–266. URL: https://doi.org/10.1007/978-3-030-32327-1_46.

[3] J. Sequeda, O. Lassila, Designing and building enterprise knowledge graphs, Synthesis Lectures on Data, Semantics, and Knowledge 11 (2021) 1–165.

[4] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, J. Taylor, Industry-scale knowledge graphs: lessons and challenges, Communications of the ACM 62 (2019) 36–43.

[5] T. P. Tanon, G. Weikum, F. M. Suchanek, YAGO 4: A reason-able knowledge base, in: The Semantic Web - 17th International Conference, ESWC, volume 12123 of *Lecture Notes in Computer Science*, Springer, Heraklion, Crete, Greece, 2020, pp. 583–596.

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

[7] WESO, RDFShape, http://rdfshape.weso.es, 2023. Accessed 6th May, 2024.

[8] K. Rabbani, M. Lissandrini, K. Hose, Shacl and shex in the wild: A community survey on validating shapes generation and adoption, in: Proceedings of the ACM Web Conference 2022, ACM, Online, Lyon, France, 2022, pp. 260–263. URL: https://www2022.thewebconf.org/PaperFiles/65.pdf.

[9] E. Prud'hommeaux, J. E. L. Gayo, H. R. Solbrig, Shape expressions: an RDF validation and transformation language, in: Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS 2014, Leipzig, Germany, September 4-5, 2014, ACM, Leipzig, Germany, 2014, pp. 32–40. URL: https://doi.org/10.1145/2660517.2660523. doi:10.1145/2660517.2660523.

[10] H. Knublauch, D. Kontokostas, Shapes constraint language (shacl), W3C Candidate Recommendation 11 (2017).

[11] E. Prud'hommeaux, J. E. L. Gayo, H. R. Solbrig, Shape expressions: an RDF validation and transformation language, in: Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS, ACM, Leipzig, Germany, 2014, pp. 32–40.

[12] A. Keely, SHACLGEN, https://pypi.org/project/shaclgen/, 2023. Accessed 6th May, 2024.

[13] D. Fernandez-Álvarez, J. E. Labra-Gayo, D. Gayo-Avello, Automatic extraction of shapes using shexer, Knowledge-Based Systems 238 (2022) 107975.

[14] A. Cimmino, A. Fernández-Izquierdo, R. García-Castro, Astrea: Automatic generation of SHACL shapes from ontologies, in: ESWC, volume 12123 of *Lecture Notes in Computer Science*, Springer, Heraklion, Crete, Greece, 2020, p. 497.

[15] N. Mihindukulasooriya, M. R. A. Rashid, G. Rizzo, R. García-Castro, Ó. Corcho, M. Torchiano, RDF shape induction using knowledge base profiling, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC, ACM, Pau, France, 2018, pp. 1952–1959.

[16] I. Boneva, J. Dusart, D. Fernández-Álvarez, J. E. L. Gayo, Shape designer for shex and SHACL constraints, in: Proceedings of the ISWC 2019 Satellite Tracks, volume 2456 of *CEUR Workshop Proceedings*, CEUR-WS.org, Auckland, New Zealand, 2019, pp. 269–272.

[17] T. Quadrant, TopBraid, https://www.topquadrant.com/products/topbraid-composer/, 2023. Accessed 6th May, 2024.
[18] H. J. Pandit, D. O'Sullivan, D. Lewis, Using ontology design patterns to define SHACL shapes, in: Proceedings of the 9th Workshop on Ontology Design and Patterns, volume 2195 of *CEUR Workshop Proceedings*, CEUR-WS.org, Monterey, USA, 2018, pp. 67–71.
[19] K. Rabbani, M. Lissandrini, K. Hose, Extraction of validating shapes from very large knowledge graphs, Proc. VLDB Endow. 16 (2023) 1023–1032. URL: https://doi.org/10.14778/3579075.3579078. doi:10.14778/3579075.3579078.
[20] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. G. Ives, Dbpedia: A nucleus for a web of open data, in: The Semantic Web, 6th International Semantic Web Conference, volume 4825 of *Lecture Notes in Computer Science*, Springer, Busan, Korea, 2007, pp. 722–735.
[21] G. Preti, M. Lissandrini, D. Mottin, Y. Velegrakis, Mining patterns in graphs with multiple weights, Distributed and Parallel Databases (2019). URL: https://doi.org/10.1007/s10619-019-07259-w. doi:10.1007/s10619-019-07259-w.
[22] W3C, W3C: RDF Type, http://www.w3.org/1999/02/22-rdf-syntax-ns#type, 2023. Accessed 6th May, 2024.
[23] O. Savkovic, E. Kharlamov, S. Lamparter, Validation of SHACL constraints over kgs with OWL 2 QL ontologies via rewriting, in: The Semantic Web - 16th International Conference, ESWC 2019, Portorož, volume 11503 of *Lecture Notes in Computer Science*, Springer, Slovenia, 2019, pp. 314–329.
[24] W3C, SHACL- core constraint components, https://www.w3.org/TR/shacl/#core-components, 2023. Accessed 6th May, 2024.
[25] WesoShaclConvert, SHACL to ShEx converter, https://rdfshape.weso.es/shaclConvert, 2023. Accessed 6th May, 2024.
[26] J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: current status and future directions, Data mining and knowledge discovery 15 (2007) 55–86.
[27] C. Borgelt, Frequent item set mining, Wiley interdisciplinary reviews: data mining and knowledge discovery 2 (2012) 437–456.
[28] Y. Guo, Z. Pan, J. Heflin, Lubm: A benchmark for owl knowledge base systems, Journal of Web Semantics 3 (2005) 158–182.
[29] WikiData, WikiData-2015, https://archive.org/details/wikidata-json-20150518, 2023. Accessed 6th May, 2024.
[30] K. Rabbani, Quality Shape Extraction - resources and source code, https://github.com/dkw-aau/qse, 2023. Accessed 6th May, 2024.