# Extracting shapes from large RDF data collections

Daniel Fernández-Álvarez[1,*], Yasunori Yamamoto[2], Jose Emilio Labra-Gayo[1] and Andra Waagmeester[3]

[1] *WESO Research Group, University of Oviedo, Oviedo, Spain*
[2] *Database Center for Life Science, ROIS-DS, Kashiwa, Japan*
[3] *Micelio, Antwerp, Belgium*

**Abstract**

There is an increasing number of projects based on RDF graphs. Shape languages, such as SHACL and ShEx, have been proposed to support the evolution of such projects on two main aspects: description and validation of RDF content. However, producing shapes for an existing knowledge graph is an arduous and time-consuming task when dealing with large data sources.

Automatic shape extractors are software elements that allow us to tackle such issue. They can produce RDF shapes by exploring existing RDF content. However, these tools usually suffer from scalability issues related to memory availability in those scenarios where they could be more useful: large data sources. To deal with these situations, some extractors implement sampling strategies. They extract shapes from a representative part of the input data rather than using the whole dataset. However, such mechanisms may lose some features which are not frequent among the input data.

We propose an alternative approach based on splitting the original input into parts, running the extraction process over each part, and consolidating the obtained result in a single schema. We demonstrate through experimentation that our approach can outperform sampling w.r.t. of quality of the obtained results. The software used for these experiments is publicly available.

**Keywords**

RDF, RDF Shapes, Shape Extraction, Large Graphs

## 1. Introduction

In the last decade, various declarative languages have been developed and discussed to define RDF data patterns or shapes. These languages serve multiple purposes, from automatic validation of RDF data sources  [1] for documenting the presented data or expressing expectations from consumed data [2]. One such language dedicated to constraint validation is the Shapes Constraint Language (SHACL) [3], which is a W3C recommendation. Shape Expressions (ShEx) [4] is a similar declarative language which is the technology chosen by some significant projects such as Wikidata [5]. Although these languages are not entirely equivalent, they are both based on the concept of *data shape*. A data shape is a structure that describes the kind and amount

*SWAT4HCLS 2024: Semantic Web Applications and Tools for Health Care and Life Sciences, February 26–29, 2024, Leiden, The Netherlands*

[*] Corresponding author.

fernandezalvdaniel@uniovi.es (D. Fernández-Álvarez); yy@dbcls.rois.ac.jp (Y. Yamamoto); labra@uniovi.es (J. E. Labra-Gayo); andra@micel.io (A. Waagmeester)
0000-0002-8666-7660 (D. Fernández-Álvarez); 0000-0002-6943-6887 (Y. Yamamoto); 0000-0001-8907-5348 (J. E. Labra-Gayo); 0000-0001-9773-4008 (A. Waagmeester)

CEUR Workshop Proceedings (CEUR-WS.org)

of relations that one should expect to find for a specific type of node in a specific RDF graph. Shapes are organized in schemes that describe a set of structures in a certain data source.

Writing shape schemes is not a trivial task. It requires 1) to be skilled in a shape language, 2) to know the data well enough to be able to describe its structure, and 3) time, both for writing and maintaining it in case the graph structure evolves. The complexity of this process scales up with factors such as the number of concepts to describe, their internal complexity (variety of relations for each concept), the size of the data source, or the frequency of events able to trigger a schema evolution.

To tackle this issue, several approaches to perform automatic shape extraction have been proposed [6]. Such approaches can generate shape schemes by inferring or mining data from existing RDF content. Shape extractors reduce the creation and maintenance times to obtain up-to-date shape schemes, both when they are used to produce final shapes [7], or shapes that act as a draft so domain experts can refine them a posteriori [2].

Shape extractors can be especially useful when dealing with large data sources, as it is usually much harder to produce/maintain a shape schema up-to-date in this type of scenario. However, most of the existing approaches have scalability problems related to memory usage in such contexts.

Some existing tools are able to extract shapes from large data sources by using sampling-based strategies [8, 9]. However, these kinds of techniques could lead to knowledge loss when a certain feature of the target data is not frequent and cannot be observed among the sample chosen.

In this paper, we propose an alternative approach to extract shapes from large graphs. Our proposal is based on slicing the target input, performing independent shape extractions over on each slice, and then merging the obtained results in a single schema. With this, we tackle the described scalability issues while still using all the input's knowledge.

We have developed a software prototype implementing our proposal and used it to test its potential. We extracted shapes from a subset of UniProt [10] using an approach with no memory restrictions, whose produced schema was used as a baseline. Then, we extracted shapes with our proposal and a sampling-based one using different settings. In our experiments, our prototype produced schemas more similar to the baseline ones while using the same amount of memory as the sampling technique. Our prototype works with ShEx, but both the proposal and the experimentation could be trivially extended to SHACL.

In this document, we describe and discuss our proposal and prototype. Both the software and data used in our experiments are publicly available, so the obtained results are fully reproducible.

## 2. Background

Several tools following different approaches have been proposed for the automatic extraction of RDF shapes from existing RDF content. We can distinguish two main different types of approaches: T-BOX-based and A-BOX-based.

On the one hand, there are techniques based on mapping T-BOX concepts into shapes, i.e., ontology definitions involving classes, properties, and their domain, range, and expected cardinality [11, 12, 13]. However, when the data in a graph is built using contextual constraints concerning the use of properties/classes that are not defined in the ontologies of those properties/classes, then the shapes produced by this type of approach may not describe the actual topologies in a graph as precisely as possible.

On the other hand, there are techniques based on mining the neighborhood of some nodes used as examples, assuming that such a mining process will reveal generalizable structures [8, 9, 12, 14, 15, 16, 17, 18, 19]. These approaches can capture contextual restrictions for the use of a certain property/class, but they may not be able to 1) detect ontology constraint violations, or 2) add features to a shape that cannot be clearly distinguished among the example data but are in an ontology [20]. The extraction process that we propose in this paper is another example of an A-BOX-based proposal.

Both T-BOX-based and A-BOX-based approaches can suffer from scalability issues related to lack of main memory when handling large inputs, but this type of issue is more frequent with A-BOX-based tools. When we explore the data contained in big well-known graphs, such as UniProt or Wikidata, we can see that the number of abstract concepts is orders of magnitude smaller than the number of entities that are instances of those concepts.

A limitation of many A-BOX-based extractors is the size of the input graph, as they need to load such content in an in-memory data structure. However, there are some other approaches, such as sheXer [9] and QSE [9], which are based on iterative parsing strategies that do not require allocating in main memory the whole target graph, but only the information that is relevant for the extraction process. Still, the data structures required for such information can lead to scalability issues.

Both sheXer and QSE implement extra mechanisms to be able to perform shape extractions in those scenarios. Those mechanisms are based on the same core idea: sampling. Rather than using the whole set of available instances/examples to extract a shape, the tool selects a subset of them that should be representative enough. However, as with any sampling-based approach, there is a chance of missing some information when the sample is not adequate for detecting infrequent and yet meaningful features.

## 3. Proposal description

The core idea of our proposal consists of slicing the input content into several parts, running a shape extraction process over each part, and, finally, merging the obtained results in a single schema. To test this proposal, we have developed a software prototype built on top of existing solutions for automatic shape extraction. In this section, we will explain each stage of our proposed workflow and describe our implemented prototype.

### 3.1. Input slicing

In this stage of the process, we need to split the input into smaller parts or slices. For this paper's experimentation, we slice the input by using chunks of N consecutive triples. For that, we used the Java ConvRDF library[1] to transform different RDF syntaxes to n-triples and then used bash commands to split the resulting file into chunks of a certain number of lines.

[1] https://github.com/dbcls/ConvRDF

**Figure 1:** Graph example with different turtle serializations

```
:Jimmy a :Person ;
   :age 21    ;
   :name "Jimmy" .

:Sarah a :Person ;
   :age 22    ;
   :name "Sarah" .

:Laura a :Person ;
   :age 23    ;
   :name "Laura" .
```

```
:Jimmy a :Person .
:Sarah a :Person .
:Laura a :Person .

:Jimmy :age 21 .
:Sarah :age 22 .
:Laura :age 23 .

:Jimmy :name "Jimmy" .
:Sarah :name "Sarah".
:Laura :name "Laura".
```

**Figure 2:** Sub-graphs obtained after splitting the content of Figure 1 in three parts

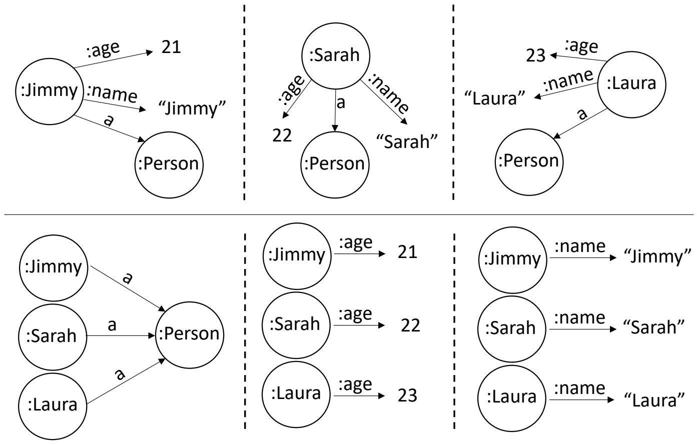


**Figure 3:** Shape that could be obtained with a sub-graph of the Figure’s 1’s upper part

```
:Person_shape {
    a [ :Person ] ;
    :age xsd:int ;
    :name xsd:string  }
```

```
:Person-shape {
    a [ :Person ] ; }
```

However, this slicing approach may cause some knowledge loss. Let us consider a graph such as the one described on the left side of Figure 1. When we split such file into groups of three consecutive triples, we obtain the three sub-graphs shown in the upper part of Figure 2. Each sub-graph contains a complete example of a `:Person` instance. These three inputs will allow a shape extractor to produce the shape shown on the left side of Figure 3.

Let us suppose now that we perform the same slicing approach with the triples on the right side of Figure 1, which describe the same information with a different serialization. Then, we would obtain the three sub-graphs shown at the bottom of Figure 2. An A-BOX-based extractor aiming to produce a shape for the class `:Person` will look for `:Person` instances and check their neighborhood. The only sub-graph of those three containing instances of `:Person` is the first one, so that would be the only useful input for the extractor. However, the `:Person` instances of this sub-graph are not connected to any other information but their type. Then, the shape obtained with this sub-graph will be the one shown on the right side of Figure 3.

Our current prototype can be affected by this kind of information loss when the knowledge of an entity is not contained in a single slice. However, turtle serializations are frequently organized such that the triples with a common subject are placed next to each other. For this reason, our experimental results have not been heavily affected by this issue.

### 3.2. Shape extraction

We have used sheXer to carry the shape extraction part in our proposed workflow. sheXer is an A-BOX-based approach for extracting shapes from large RDF input sources implemented in Python[2]. sheXer meets our requirements for this proposal, as it 1) can be configured to use sampling with very large inputs, and 2) can handle quite large inputs even without sampling. This allows us to perform the experiments described in section 4.

sheXer uses an iterative parsing strategy that avoids placing the whole target content in the main memory. However, it keeps an in-memory structure in which general features of the entities used as examples to extract shapes are annotated. This may cause a memory overflow when the number of example entities is too large. The library prevents such issues by letting the user configure a limit for the number of entities that could be considered to extract a shape.

sheXer allows for configuring many relevant aspects of the extraction process, but reviewing them all falls out of the scope of this paper. A thorough revision of this tool can be read in [9].

### 3.3. Shape consolidation

The last stage of our proposed workflow consists of merging the obtained schemes into a single one. We reduced the problem of merging N schemes to the problem of merging two. We merge two schemes $s_1$ and $s_2$ of the original set $S$ and get a resulting schema $r$. $s_1$ and $s_2$ are added to a set of explored schemes $E$. $r$ is then merged with another schema $s_3 \in S / s_3 \notin E$. $r$ now contains the merged information of $s_1$, $s_2$, and $s_3$, and $s_3$ is added to $E$. This process goes on until $\nexists s_n \in S / s_n \notin E$. At this point, $r$ contains the merged information of every $s_i \in S$.

Given a couple of schemes $s_1$ and $s_2$, merging them into a schema $r$ takes the following steps:

1. We create a set $T_r$ that will contain the shapes of the resulting schema $r$. Initially, $T_r$ contains those shapes that are only in $s_1$ or in $s_2$, but not in both schemes. Formally, $T_r \leftarrow \{t_i / (t_i \in f_t(s_1)) \wedge (\nexists t_j \in f_t(s_2) / f_l(t_j) = f_l(t_i))\} \cup \{t_i / (t_i \in f_t(s_2)) \wedge (\nexists t_j \in f_t(s_1) / f_l(t_j) = f_l(t_i))\}$, where $f_t(s)$ is a function that returns the set of shapes in the schema $s$, and $f_l(t)$ is a function that returns the URI of the shape $t$.
2. For each pair of shapes $t_1 \in f_t(s_1)$ and $t_2 \in f_t(s_2)$ such that $f_l(t_1) = f_l(t_2)$, we add to $T_r$ a new shape that merges the constraints of $t_1$ and $t_2$. The merging process assumes that

[2] https://github.com/DaniFdezAlvarez/shexer. The current features of this library are described in its GitHub repository. The experiments were performed using sheXer 2.2.2.

1) each property will be used in one of the shape's constraints at most (except `rdf:type`, which is treated specially), and 2) the node constraint of a property will not contain logical expressions. These conditions can be enforced with a certain configuration of sheXer. The consolidation process of two shapes $t_1$ and $t_2$ into a single shape $t_r$ consists of the following steps:

a) We create a set $C_t$ containing those constraints which are exactly equal in both shapes, i.e., whose property, node constraint, and cardinality are identical. Formally, $C_t \leftarrow \{c_i/(c_i \in f_c(t_1)) \land (\exists c_j \in f_c(t_2)/c_i = c_j\}$, where $f_c(t)$ is a function that returns the set of constraints of the shape $t$.
b) We find the set $C'_t$ containing constraints of one of the shapes whose property is not used in any constraint of the other shape. Formally, $C'_t \leftarrow T_t \cup \{c_i/(c_i \in f_c(t_1)) \land (\nexists c_j \in f_c(t_2)/f_p(c_j) = f_p(c_i))\} \cup \{c_i/(c_i \in f_c(t_2)) \land (\nexists c_j \in f_c(t_1)/f_p(c_j) = f_p(c_i))\}$, where $f_p(c)$ is a function that returns the property of the constraint $c$. Then, we add to $C_t$ every $c \in C'_t$, but changing their minimal cardinality to zero.
c) We add to $C_t$ a constraint $c_{ij}$ that merges each pair of constraints $c_i \in f_c(t_1)$ and $c_j \in f_c(t_2)$ such that they have the same property but differ in cardinality or node constraint. The node constraint and cardinality of $c_{ij}$ will be selected such that they describe the minimum superset that conforms at a time with $c_i$ and $c_j$.
d) Finally, we create a new shape $t_r$ whose constraints are the ones in the set $C_t$.

When our prototype merges two constraints, it also merges some statistical information generated by sheXer. These statistics indicate the support of a certain constraint among the example data. To mitigate the knowledge loss that may occur after slicing the input, we define a threshold $\theta_n$ in $[0, 1]$ that takes advantage of those statistics. Whenever the ratio of instances supporting a certain constraint is higher than $1 - \theta_n$, we promote that support to 1, assuming that the lack of total support was caused by knowledge loss.

We have implemented the described approach for shape consolidation in Python[3].

## 4. Experiments

To evaluate our proposal, we have extracted shapes from a large RDF dataset using both our approach and a sampling-based one. We compare our proposal with a sampling strategy, which is the current state-of-the-art approach for handling large datasets with affordable hardware. We used sheXer to generate sampling-based schemes.

We designed an experiment involving a real use case with a subset of UniProt containing 83,855,205 triples[4]. We extracted a shape for every class without any memory restriction, which took 32.96 GB of RAM memory. The obtained schema was used as a baseline[5].

We run the consolidation and sampling approaches using different slice sizes or instance limits respectively. As the motivation of both approaches is tackling an issue of memory availability,

[3] https://github.com/shex-consolidator/shex-consolidator. This prototype was implemented to work with sheXer's outputs. This source code may fail when trying to merge schemes that were not originally produced by sheXer.

[4] This content can be downloaded from https://ftp.uniprot.org/pub/databases/uniprot/current_release/rdf/uniprotkb_reviewed_eukaryota_opisthokonta_metazoa_33208_0.rdf.xz

[5] All the shapes extracted during this experimentation are published in the tool's repository: https://github.com/shex-consolidator/shex-consolidator/tree/main/shex_results

| | **Peak of RAM usage (in GB)** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **0.5** | **1** | **2** | **3** | **4** | **5** | **6** |
| **Consolidation: triples per slice** | 1.1M | 2.2M | 4.5M | 7.7M | 9.1M | 11.4M | 13.7M |
| **Sampling: max. instances per shape** | 3.1k | 6.7k | 17k | 27.8k | 39.6k | 53.5k | 66.8k |

**Table 1**
Settings used for the consolidation and sampling approaches.

we think that it is fair to compare these approaches by using settings that produce the same peak of memory usage. With this premise, we run different experiments to compare the sampling and consolidation approaches using the settings listed in Table 1[6].

To compare the shapes generated with the baseline and the different approaches, we check:

- *S_EQ*: Ratio of shapes which are in both schemes and are exactly equal.
- *S_CAR*: Ratio of shapes which are in both schemes and the only difference found between their constraints is related to cardinality.
- *S_NOD*: Ratio of shapes which are in both schemes and whose constraints use the same properties, but at least one of those constraints has different node constraint.
- *S_PRO*: Ratio of shapes which are in both schemes, but one of the schemes has a constraint that is not used in the other shape.
- *S_ERR*: Ratio of shapes that are not in both schemes.

We also perform similar measures at constraint level. For those shapes whose URI exists in both schemes, we pair a constraint of a version of the shape with a constraint of its counterpart that uses the same property (if possible). Considering those paired elements, we check:

- *C_EQ*: Ratio of constraints which are exactly equal in both shapes.
- *C_CAR*: Ratio of constraints that have the same property and node constraint, but different cardinality.
- *C_NOD*: Ratio of constraints that have the same property, but different node constraint.
- *C_ERR*: Ratio of constraints that has a property that is not used in both shapes, i.e., that could not be paired with a similar constraint in the other version of its shape.

In Table 2, we show the comparison between the baseline and every extraction process run for both consolidation and sampling approaches. Note that the columns in this table are primarily organized w.r.t. memory usage. The number of lines per slice or the instance cap used for the consolidation (C) and sampling (S) approaches respectively are shown in Table 1. Accumulated results of each measurement are shown in Figure 4, so one can see the ratio of elements that have, at least, a certain level of similarity with their counterpart element in the baseline.

## 5. Discussion

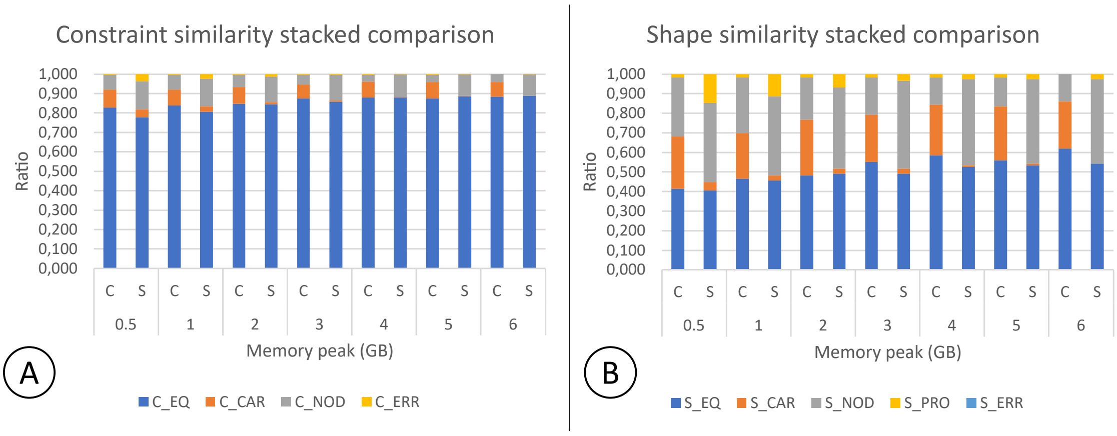
All values for the row *S_ERR* in Table 2 are 0, which means that, in every case, both approaches were able to produce the same shapes for the same classes as the baseline. Also, even with

[6] We empirically determined that setting $\theta_n = 5 \cdot k$, where $k$ is the number of chunks, was adequate to mitigate the slicing knowledge loss. However, more experimentation is required to obtain optimal values of $\theta_n$

| | Peak of memory usage (in GB) | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.5 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | |
| **Concept** | **C** | **S** | **C** | **S** | **C** | **S** | **C** | **S** | **C** | **S** | **C** | **S** | **C** | **S** |
| **S_EQ** | 0.414 | 0.405 | 0.466 | 0.457 | 0.483 | 0.491 | 0.552 | 0.491 | 0.586 | 0.526 | 0.560 | 0.534 | 0.621 | 0.543 |
| **S_CAR** | 0.267 | 0.043 | 0.233 | 0.026 | 0.284 | 0.026 | 0.241 | 0.026 | 0.259 | 0.009 | 0.276 | 0.009 | 0.241 | 0.000 |
| **S_NOD** | 0.302 | 0.405 | 0.284 | 0.405 | 0.216 | 0.414 | 0.190 | 0.448 | 0.138 | 0.440 | 0.147 | 0.431 | 0.138 | 0.431 |
| **S_PRO** | 0.017 | 0.147 | 0.017 | 0.112 | 0.017 | 0.069 | 0.017 | 0.034 | 0.017 | 0.026 | 0.017 | 0.026 | 0.000 | 0.026 |
| **S_ERR** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C_EQ** | 0.829 | 0.779 | 0.839 | 0.807 | 0.847 | 0.845 | 0.875 | 0.859 | 0.880 | 0.880 | 0.875 | 0.886 | 0.884 | 0.888 |
| **C_CAR** | 0.092 | 0.040 | 0.082 | 0.028 | 0.088 | 0.012 | 0.072 | 0.006 | 0.082 | 0.002 | 0.086 | 0.002 | 0.076 | 0.000 |
| **C_NOD** | 0.076 | 0.145 | 0.076 | 0.141 | 0.062 | 0.131 | 0.050 | 0.131 | 0.034 | 0.116 | 0.036 | 0.110 | 0.040 | 0.110 |
| **C_ERR** | 0.004 | 0.036 | 0.004 | 0.024 | 0.004 | 0.012 | 0.004 | 0.004 | 0.004 | 0.002 | 0.004 | 0.002 | 0.000 | 0.002 |

**Table 2**
Comparison of (C)onsolidation and (S)ampling approaches with the baseline

**Figure 4:** Shape (A) and Constraint (B) stacked comparisons



the lowest memory peak, more than 80% of the extracted constraints are exactly equal to the constraints in the baseline with both approaches. This percentage only improves when we increase the memory peak. As shown in Figure 4, the ratio of identical shapes is usually remarkably lower than the ratio of identical constraints. These numbers together indicate that, even if the produced shapes are not exactly equal, frequently they do not have too many internal differences. Note also that the consolidation approach (C) performs better or nearly equal to the sampling approach (S) at almost any memory peak tested.

The ratio of shapes and constraints whose differences with the baseline's counterparts are only related to cardinality is remarkably better with C. Cardinality-related issues in C are mostly caused by situations where the baseline has a constraint whose minimal cardinality is 1, but the extractor produces 0. Cardinality-related issues observed among the shapes obtained with S tend to happen because the extracted constraints are more specific than the ones produced by the baseline. This happens when there is a frequent feature observed among a class' instances, but there are some instances that do not manifest it. If none of those instances are part of the used sample, the obtained cardinality will be wrong.

Most node constraint-related issues occur when this element should be a specific shape label, but the extractor uses the macro *IRI* instead. With both approaches, this situation arises when the sub-graph computed does not contain the type of a certain entity used as object in a triple. However, note that the ratio of elements which at least have the same property as its baseline's counterpart is consistently higher with C.

Finally, note that property-related issues are infrequent and tend to disappear when we use a high enough memory peak. However, note also that, with C, these cases are very rare even for the lowest memory peaks tested.

The results observed in our experiments are promising but could be biased by the type of input. As explained in section 3.1, our prototype could be affected by a type of graph serialization that has not been observed in the chosen use case. On the other hand, we hypothesize that the obtained result could be remarkably improved by implementing a more complex slicing strategy, so we enforce an adequate knowledge organization in the used slices. More experimentation would be required to prove such a hypothesis.

## 6. Conclusions

In this paper, we describe a novel approach for tackling memory-related issues when performing automatic extraction of shapes from existing RDF content. We define a workflow where 1) we slice the input source, 2) perform an independent shape extraction process over each slice, and 3) merge the obtained results. This allows us to use the whole content of the target graph while controlling memory usage. We have developed a prototype implementing that workflow and compared it with a state-of-the-art tool based on sampling. We extracted a shape schema from a UniProt subset with a tool with no memory restrictions. Then, we extracted schemes using the evaluated approaches. In our experiments, our prototype produced shapes more similar to the baseline ones while using the same amount of memory as the sampling-based approach.

Although these results are promising, this is an ongoing work. Experimentation with different types of sources and different slicing strategies should be performed to determine the actual potential of our proposal.

**Acknowledgements** This work has received partial funding from the project ANGLIRU (Applying Knowledge Graphs to Research Data Interoperability and Reusability, MCI-21-PID2020-117912RB-C21) funded by the Spanish Agency for Research. This work was partially supported by the National Bioscience Database Center (NBDC) of the Japan Science and Technology Agency (JST). We also thank the organizers of the DBCLS Biohackathon 2023, which enabled the cross-collaboration that triggered this research.

## References

[1] J. E. Labra-Gayo, H. García-González, D. Fernández-Alvarez, E. Prud'hommeaux, Challenges in rdf validation, Current Trends in Semantic Web Technologies: Theory and Practice (2019) 121–151.

[2] A. Waagmeester, E. L. Willighagen, A. I. Su, M. Kutmon, J. E. L. Gayo, D. Fernández-Álvarez, Q. Groom, P. J. Schaap, L. M. Verhagen, J. J. Koehorst, A protocol for adding knowledge to Wikidata: aligning resources on human coronaviruses, BMC Biol 19 (2021) 12.

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

[4] E. Prud'hommeaux, J. E. Labra Gayo, H. Solbrig, Shape expressions: an rdf validation and transformation language, in: Proceedings of the 10th International Conference on Semantic Systems, 2014, pp. 32–40.

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

[6] K. Rabbani, M. Lissandrini, K. Hose, Shacl and shex in the wild: A community survey on validating shapes generation and adoption, in: A. L. Gentile, L. Pasquale (Eds.), Companion Proceedings of the Web Conference, WWW'22 Companion, ACM, 2022.

[7] F. Cifuentes-Silva, D. Fernández-Álvarez, J. E. Labra-Gayo, National budget as linked open data: New tools for supporting the sustainability of public finances, Sustainability 12 (2020) 4551.

[8] K. Rabbani, M. Lissandrini, K. Hose, Extraction of validating shapes from very large knowledge graphs, Proceedings of the VLDB Endowment 16 (2023) 1023–1032.

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

[10] U. Consortium, Uniprot: a worldwide hub of protein knowledge, Nucleic acids research 47 (2019) D506–D515.

[11] A. Cimmino, A. Fernández-Izquierdo, R. García-Castro, Astrea: automatic generation of shacl shapes from ontologies, in: European Semantic Web Conference, Springer, 2020, pp. 497–513.

[12] A. Keely, Shaclgen, 2023. URL: https://pypi.org/project/shaclgen/.

[13] H. J. Pandit, D. O'Sullivan, D. Lewis, Using ontology design patterns to define shacl shapes., in: WOP@ ISWC, 2018, pp. 67–71.

[14] I. Boneva, J. Dusart, D. F. Alvarez, J. E. L. Gayo, Shape designer for shex and shacl constraints, in: ISWC 2019-18th International Semantic Web Conference, 2019.

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

[16] B. Groz, A. Lemay, S. Staworko, P. Wieczorek, Inference of shape graphs for graph databases, in: 25th International Conference on Database Theory (ICDT 2022), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[17] P. G. Omran, K. Taylor, S. J. R. Méndez, A. Haller, Towards shacl learning from knowledge graphs., in: ISWC (Demos/Industry), 2020, pp. 94–99.

[18] B. Spahiu, A. Maurino, M. Palmonari, Towards improving the quality of knowledge graphs with data-driven ontology patterns and shacl., in: ISWC, 2018, pp. 103–117.

[19] K. Rabbani, M. Lissandrini, K. Hose, Shactor: Improving the quality of large-scale knowledge graphs with validating shapes, in: Companion of the 2023 International Conference on Management of Data, 2023, pp. 151–154.

[20] S. Bechhofer, Owl: Web ontology language, in: Encyclopedia of Database Systems, Springer, 2009, pp. 2008–2009.