Benchmark for Performance Evaluation of SHACL-SPARQL Compliant Open Source Libraries over the ERA Knowledge Graph

Edgar Martínez-Sarmiento

1. Related Work

Schaffenrath et al. in their article performed a SHACL validation performance evaluation over four different graph databases. Their shapes' formulation included a range of SHACL-core constraints; no SPARQL constraint was part of the test set [1].

2. The Benchmark

In this study, a SHACL-SPARQL validation performance evaluation in open-source libraries was proposed. The benchmark focuses on three aspects. Namely, load and validation estimated times, and estimated RAM used for data loading. The experiments were performed on top of the ERA dataset split in three orders of magnitude using seven different open-source libraries. The formulated shapes are a combination of SHACL-core and SHACL-SPARQL constraints that were loaded in each library to validate the dataset's subset triples. Both, the dataset[1] and the shapes[2] are public assets if the reproduction of the benchmark is intended.

3. The Benchmark Dataset

The European Union Agency for Railways (ERA) Knowledge Graph (KG) is the result of an extensive integration effort to produce an interoperable solution for railways datasets around the European Union nations. This dataset

[1]https://data-interop.era.europa.eu/era-vocabulary/era-shapes
[2]https://data-interop.era.europa.eu/endpoint

contains more than 37 million quads from 29 countries' railway infrastructure information following the ERA vocabulary, an specialized vocabulary created for the Register of Infrastructure (RINF) System. Apart from the full dataset, two country-level subsets were prepared: one with around one million triples (ES), and a second with approximately ten million triples (FR). The full set and country subsets were converted to ntriples to import data in the libraries without any inconvenience given that some don't support nquads format. In addition, a toy test set containing only 40 triples was generated to verify the libraries' SHACL-SPARQL support. The full set of SHACL shapes was designed element/system-wise to validate data bearing in mind specific railway components to provide a compatibility report in a given route.

4. The SHACL-SPARQL Shapes

From the collection of shapes generated, the ones concerning the validation of train detection systems were extracted as a subset for the benchmark. 225 triples comprise the combination of SHACL-core and SHACL-SPARQL constraints. Where one is a `sh:NodeShape`, 14 are `sh:PropertyShape`, and 12 are of the `sh:SPARQLConstraint` type. Such a combination provides a balanced collection of shapes for the benchmark. Given that the experiments were designed to run not only on top of countries' subsets but also with the full dataset the number of shapes was reduced due to hardware limitations. An extract of the shapes used in the current study is presented in the Listing 1. The node shape targets all individuals of the class `era:TrainDetectionSystem`. The entire set of SPARQL constraints is of the SELECT kind (`sh:select`), which in this particular case, its purpose is to validate whether a train detection system has a concept within the predefined values. In addition, property shapes were defined to report violations when corrupted datatypes (`sh:datatype`) or REGEX patterns (`sh:pattern`) are present in the data.

Listing 1: SHACL shapes extract serialized in ttl.

```
era-sh:TrainDetectionSystemShape
        a sh:NodeShape ;
        sh:targetClass era:TrainDetectionSystem .
era-sh:TrainDetectionSystemShape sh:sparql era-sh:
    TrainDetectionSystemTypeSKOS .
era-sh:TrainDetectionSystemTypeSKOS
```

```
        a sh:SPARQLConstraint ;
        rdfs:comment "Indication of types of train detection system installed." ;
        era:rinfIndex "1.1.1.3.7.1.1" ;
        sh:severity sh:Violation ;
        sh:message "Indication of types of train detection system installed (1.1.1.3.7.1.1): The train detection system {$this} (label {?tdsLabel}) has a value {?concept} that is not one of the predefined values and cannot be converted into a SKOS concept on this list: http://data.europa.eu/949/concepts/train-detection/TrainDetectionSystems."@en ;
    sh:prefixes era:;
        sh:select """
    PREFIX era: <http://data.europa.eu/949/>
    PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT $this  ?concept ?tdsLabel
    WHERE {
        OPTIONAL {$this rdfs:label ?tdsLabel} .
        $this era:trainDetectionSystemType ?concept .
        era:trainDetectionSystemType era:inSkosConceptScheme ?conceptScheme .
              FILTER NOT EXISTS{
                      ?concept skos:inScheme ?conceptScheme .
              }
        }
""" .
era-sh:TrainDetectionSystemShape sh:property era-sh:maxDistConsecutiveAxles .
era-sh:maxDistConsecutiveAxles
        a sh:PropertyShape ;
        rdfs:comment "Indication of maximum permitted distance between two consecutive axles in case of TSI non-compliance, given millimetres." ;
        era:rinfIndex "1.1.1.3.7.2.2" ;
        sh:path era:maxDistConsecutiveAxles ;
        sh:datatype xsd:integer ;
        sh:pattern "^([1-9]\\d{1,4}|[0-9])$" ;
        sh:severity sh:Violation ;
        sh:message "maxDistConsecutiveAxles (1.1.1.3.7.2.2): Indication of maximum permitted distance between two consecutive axles in case of TSI non-
```

```
compliance, given millimetres. This error may be
due to the train detection system having a value
that is not an integer."@en .
```

5. Performance Evaluation

In this section, an explanation of the selected libraries' usage for the dataset validation is provided. It contains details regarding the difficulties presented and the results each of them yielded.

5.1. Libraries selection

The open-source libraries included in this study were mainly selected from the SHACL Test Suite and Implementation Report [2] bearing in mind only the ones that passed at least one SHACL-SPARQL test. In the case of the non-tested libraries, a brief verification of the feature was performed in order for the library to be included. Five were the reported libraries included from the Test Suite: Corese, dotNetRDF, pySHACL, RDFUnit, and TopBraid. In addition, from the implementations listed in [3] by Labra et. al it was verified that apart from some of the above-mentioned libraries, the rdf4j and Jena libraries also supported SHACL-SPARQL which completed the list of libraries evaluated in this study. The implementations not available as open-source libraries, importing validation functions from other libraries or the non-SHACL-SPARQL compliant were discarded i.e. Netage, Shaclex, SHACL playground, SHACL-Check, Alternative SHACL implementation by Patel-Schneider, and ELI validator.

Written in Java, Corese is a library with a Free Software CeCILL-C license developed by Web-Instrumented Man-Machine Interactions, Communities, and Semantics (Wimmics) a joint research team between Inria Sophia Antipolis - Méditerranée and I3S (CNRS and Université Côte d'Azur) published in HAL open science software and freely available in a GitHub repository[3]. According to its documentation, it is a library compliant with the W3C standards RDF, RDFS, SPARQL1.1 Query & Update, OWL RL, SHACL, with extensions like STTL SPARQL, SPARQL Rule and LDScript.

[3]https://github.com/Wimmics/corese

dotNetRDF is an open-source .NET library for RDF licensed under the MIT License written in C# by the dotNetRDF project team. Its code is fully available in a well-maintained GitHub repository[4]. In addition to reading and writing RDF, it includes implementations of the SPARQL Protocol, SPARQL Graph Store Protocol and SPIN. It also provides some basic inferencing support including RDF-Schema, SKOS and a small subset of OWL reasoning. It provides an API for validating a graph using SHACL and another for working with a SKOS taxonomy.

Belonging to the RDFlib Family, pySHACL is a library fully written in Python by The Commonwealth Scientific and Industrial Research Organisation (CSIRO) Land & Water, Environmental Informatics Group in compliance with the W3C SHACL Recommendation. This open-source module can be configured to expand the data graph through RDFS or OWL2RL inferencing before validation. It is fully available in a Github repository[5], licensed under Apache License, Version 2.0. and published as a research artefact in Zenodo[6].

RDFUnit is an RDF Unit Testing Suite written in Java, developed by the Research Group Agile Knowledge Engineering and Semantic Web (AKSW) and published as Databugger in a conference article [4]. Among its supported schemas such as OWL-CWA, Schema.org, DSP, and RS, it is also capable of validating data against SHACL-Core and SHACL-SPARQL constraints with a very good coverage of the W3C SHACL Recommendation. RDFUnit can be used as a library where the test cases are executed as SPARQL queries using a pattern-based transformation. Its repository[7] is licensed under the Apache-2.0. license.

TopBraid SHACL API is a Java open-source implementation[8] of the W3C SHACL based on Apache Jena and licensed under the Apache-2.0. license. It was developed by the TopQuadrant Inc. company with extended support of SHACL Core and SHACL-SPARQL validation through SHACL Advanced Features.

[4]https://github.com/dotnetrdf/dotnetrdf
[5]https://github.com/RDFLib/pySHACL
[6]https://zenodo.org/records/10958008
[7]https://github.com/AKSW/RDFUnit
[8]https://github.com/TopQuadrant/shacl

rdf4j[9] is a Java framework for RDF data processing developed by the Eclipse Foundation under the BSD-3-Clause license. Its SHACL engine supports a fairly large subset of the W3C SHACL Recommendation features including SHACL-SPARQL, DASH and RSX constraints.

Jena[10] is a free and open-source Java framework licensed under Apache-2.0 for building Semantic Web and Linked Data applications. Its SHACL module implements SHACL Core and SHACL SPARQL Constraints following the W3C SHACL Recommendations. In addition, it provides SHACL Compact Syntax and SPARQL-based targets.

5.2. Evaluation Methodology

By following each library documentation, simple programs were written to run the load and validation tasks in a single virtual machine with the following specifications: 16 Core Intel Xeon Processor, 50GB RAM, and 120GB SSD. The operating system used for the evaluation was Linux Ubuntu 22.04.4 LTS. Time measurements were coded for the above-mentioned tasks as the criterion to assess performance with methods that varied from language to language. In addition, provided that each library has its own way of managing and validating data the time measurement code encapsulated the tasks in the most straightforward way within the scripts. An aggregation of both, the load and validation task times was calculated to give a global overview of the performance of the libraries. Comparing the validation reports is not in the scope of this study; nevertheless, a simple check was done for the toy test set to probe the SHACL-SPARQL support of each library. One requirement was to store data in memory to assess its usage. In order to measure the RAM used to load the dataset the approach was to capture an estimation while tracking the corresponding process' memory consumption. Additionally, a short description of the libraries' coding experience and pitfalls is provided.

5.3. Evaluation Experience

This section contains descriptions of the scripting and evaluation experience for each library considered in the study. Details regarding the possible pitfalls and commonalities are provided. In this study, the default configuration according to each library documentation was used.

[9] https://rdf4j.org/

[10] https://jena.apache.org/

Corese. Despite not being tested in the SHACL Test Suite [2] a simple test executed with the test data demonstrated this library supported SHACL-SPARQL. Loading the ntriples to memory using this library consisted of creating an empty graph and a loader to parse the triples, the load process was triggered by calling the parsing function the loader has. For the validation task, a shapes graph was created and passed together with the data graph to a Shacl object. Finally, the validation process started when running the eval method of such object. No explicit format argument was necessary to load neither the data in ntriples nor the shapes in ttl. The Graph, Shacl and Load objects are full implementations of the W3C recommendations i.e. Corese doesn't need to import external libraries for its use.

dotNetRDF. According to the SHACL Test Suite [2] this library passed all of the test from the core and the SHACL-SPARQL recommendations. The mechanism to load and validate the data resembles that of the previous library. An empty graph is filled with data through a method the file loader class has. This time there is no need to create a new load object because there is an existing one that is part of the library. After loading the shapes into a graph a processor should be initialized before calling the validate method which executes such a task. There was no need to provide the optional format argument to execute the load task. This .NET implementation covers the W3C recommendations from scratch without other RDF implementations as external dependencies.

pySHACL. The coverage in terms of SHACL-SPARQL compliance of this library is fairly high according to the Test Suite [2]. As part of the RDFLib modules family, its graph management follows the RDFLib logic. An empty graph after being created, is the object that is in charge of parsing the data within itself by calling its parse method. After generating the graphs for the data and the shapes, the validate function can be called from the pySHACL module taking both graphs as inputs to trigger the validation task. Parsing didn't require an explicit format to make it work properly.

RDFUnit. The SHACL-SPARQL coverage is partial according to the SHACL Test Suite [2]. What makes this library different from the others is that beyond a validation engine it provides a whole suite for unit testing not only for SHACL but also for other kinds of restrictions as listed in section 5.1. Moreover, the validation process doesn't require loading data in memory due to its capability of validating directly over an SPARQL endpoint. In this study, to comply with the requirement of storing data in memory, it was loaded in a Jena model given that this implementation supports them. The loading process is described in the Jena library paragraph. For the validation process, a test suite object is built pointing the shapes directory within a predefined validator object. The validate method from the validator object takes the graph model as input and executes the validation task. As previously seen, if an in-memory validation is intended this library should rely on the Jena library.

TopBraid. Passing all the tests from the SHACL Test Suite [2], this library was developed reusing Apache Jena classes. This implies that the data-loading process follows equivalent steps to that of the Jena library. A validation util method for model validation takes the Jena models for the data and the shapes as inputs to execute the validation process and at the same time, it generates a Jena report. This implementation is heavily dependent on the Jena library not only for the loading process but also for reporting the results.

rdf4j. It was demonstrated through the validation of the toy test set that this framework supported SHACL-SPARQL at least for the shapes considered in this study (Section 4). The loading and validation processes were different from the other libraries because, in the implementation of rdf4j as a whole framework, the storage system is managed at the repository level for large dataset validation. Thus, RDF data was loaded into a in-memory repository through a connection method that took the input stream and the format as input arguments. Likewise, the shapes were loaded into the same repository but specifying the SHACL shape graph in an additional argument of the add method. A repository commit method runs the validation operation which raises validation exceptions for the non-valid data. To extract the validation report as a model, the exception should be captured and optionally lifted. With the rdf4j framework, SHACL validation is only possible using this sequence.

Jena. The SHACL validation engine the Jena library has, is compliant with the shapes used in this study. The toy test set was successfully passed by this library. Regarding the data loading process, it consists of a single line where a load graph method of the preconceived data manager object points to the data directory and generates a Jena graph stored in memory. In the case of the shapes, after creating the shapes graph, they were parsed by a predefined shapes artefact which created a shapes object of it. The Jena shacl validator was the object that through the validate method executed the validation process and generated the validation report.

5.4. Evaluation Results

Each library performance was evaluated using the four data sets. Load and validation times were measured and RAM usage was estimated for the load task. The results of such experiments are listed in tables 1, 2, 3 and depicted in figures 1, 2, 3. Table 4 and figure 4 provide a global overview of the performance in terms of the aggregated time needed to perform the load and validation tasks together by library. The results were ordered from the best to the worst performing in the validation task.

Table 1: Load estimated time in seconds.

Graph	Test	ES	FR	EU
Triples	40	$\sim 1M$	$\sim 10M$	$\sim 37M$
Jena	0.382	7.961	78.935	610.657
TopBraid	0.67	7.977	75.123	625.79
Rdf4J	0.006	19.271	196.975	322.898
RDFUnit	0.559	8.471	83.536	635.805
dotnetRDF	0.104	24.283	236.224	2541.839
pySHACL	0.009	60.871	556.355	1726.141
Corese	0.219	15.128	144.361	630.132

Table 2: Validation estimated time in seconds.

Graph	Test	ES	FR	EU
Triples	40	$\sim 1M$	$\sim 10M$	$\sim 37M$
Jena	0.052	2.293	32.997	73.672
TopBraid	0.617	2.689	35.55	114.491
Rdf4J	0.077	5.24	60.071	317.121
RDFUnit	0.718	14.067	143.894	623.016
dotnetRDF	0.302	21.427	754.915	4140.534
pySHACL	1.0146	244.592	5333.809	10686.519
Corese	0.384	1947.9	353739.139	-

A noticeable difference can be spotted between the best and worst-performing libraries. In terms of time elapsed while loading data, with the full dataset

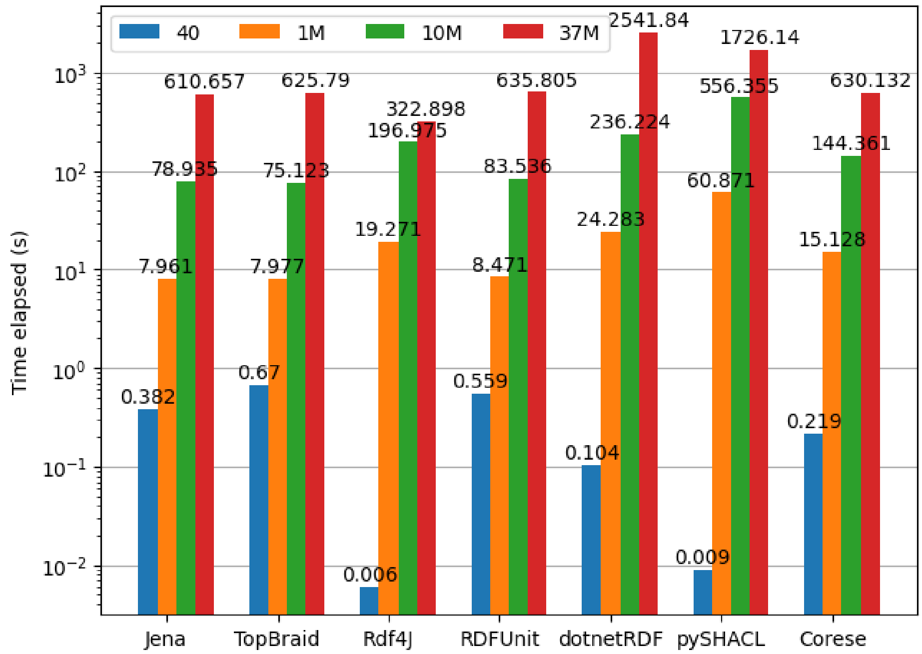

Figure 1: Load estimated time by library.

Table 3: Load estimated RAM used in GB.

Graph	ES	FR	EU
Triples	$\sim 1M$	$\sim 10M$	$\sim 37M$
Jena	1.2	3.2	13.5
TopBraid	1.1	3.1	13.5
Rdf4J	1.4	5.3	9.4
RDFUnit	1	5.7	20.9
dotnetRDF	1.4	13.7	48
pySHACL	2.9	10	46
Corese	0.7	2	3.9

dotnetRDF took almost eight times compared to the fastest one which was rdf4j. For the RAM use comparison with all of the data, the worst performing were those using almost the full capacity of the hardware i.e. dotnetRDF and pySHACL, using around 48GB and 46GB respectively to load a 9GB dataset. In other words, 5 times more memory than the storage on disk required. Surprisingly Corese, the best-performing in RAM usage, used only

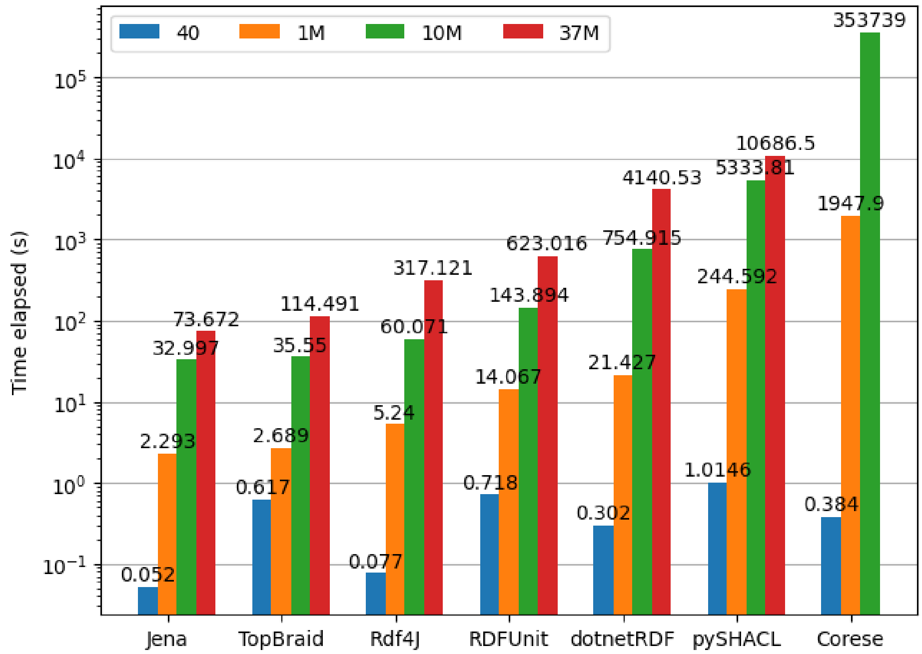

Figure 2: Validation estimated time by library.

Table 4: Cumulative estimated time in seconds.

Graph	Test	ES	FR	EU
Triples	40	$\sim 1M$	$\sim 10M$	$\sim 37M$
Jena	0.434	10.254	111.932	684.329
TopBraid	1.287	10.666	110.673	740.281
Rdf4J	0.083	24.511	257.046	640.019
RDFUnit	1.277	22.538	227.430	1258.821
dotnetRDF	0.406	45.710	991.139	6682.373
pySHACL	1.024	305.463	5890.164	12412.659
Corese	0.603	1963.028	353883.500	-

around half of the memory to store the full dataset but it didn't complete the validation task even after 15 days of running the script (the process was forced to stop at this point). On the other hand, the Jena library was the best of the seven for the validation task. It completed the task in 73.6 seconds.

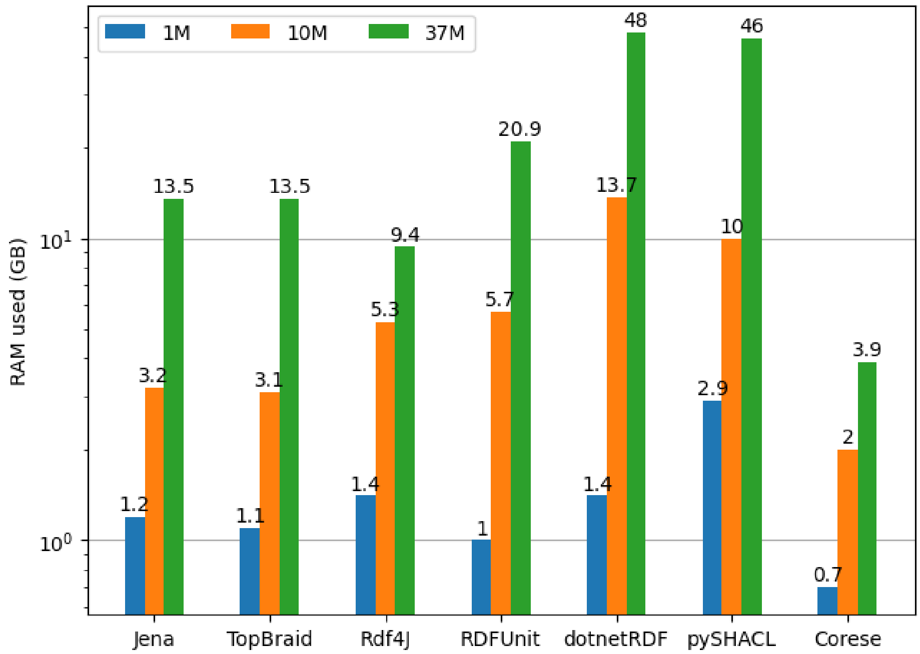

Figure 3: Load estimated RAM used by library.

6. Discussion

For the loading task, it can be inferred that load times in Jena, TopBraid and RDFUnit libraries are close to each other due to the dependency on the same Jena classes. In general terms, the loading task took longer in the case of the dotnetRDF and pySHCAL libraries when dealing with big datasets. Memory consumption was also the highest while using these libraries. What can explain the similitude or difference between libraries' needs of RAM is the *spo*, *osp* and *pso* indexations. For the most common computer specifications nowadays the worst-performing libraries in terms of RAM usage are unfeasible when working with high amounts of data. Concerning the validation task, the results suggest Jena library outperforms the others when working with the full dataset. Apparently, most of the SHACL engines present a logarithmic complexity in their validation algorithms when dealing with medium to large-size data, but in order to confirm such an affirmation more test cases should be executed. The small amount of data case presented a different trend when comparing the libraries because the objective of the toy test dataset was not intended to represent a real validation. The cumulative

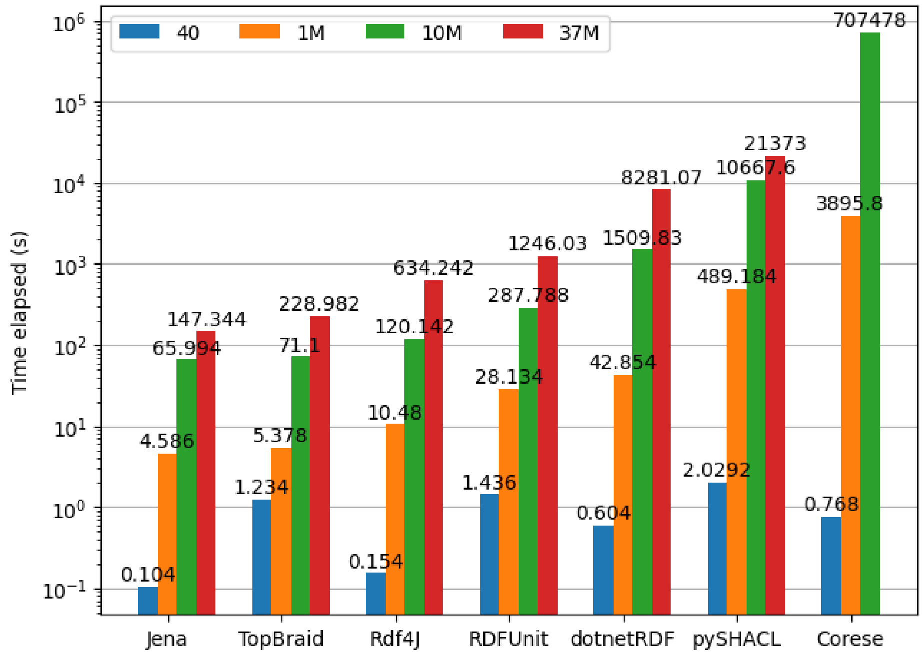

Figure 4: Cumulative estimated time by library.

estimated time depicted that it requires more than two minutes to load and validate a 37 million triples dataset using only a small collection of shapes that has only 12 SPARQL constraints. The total amount of time required is not neglectable, especially for use cases in which systems require low latency in the validation tasks.

7. Conclusions

This work fulfils the need for an open-source state-of-the-art libraries benchmark for SHACL-SPARQL validation performed in a real-world dataset. While not exhaustive in terms of constraints variety, this benchmark also provides good insight into real dataset validation requirements for data quality. Extending the limits of the validation by using more SPARQL shapes in the benchmark would provide a bigger map of the performance of the selected engines. More test cases could provide clarity regarding each library's algorithm complexity. Finally, this study reaffirms the fact that there is still plenty of room for improvement in the performance using Semantic Web technologies validation schemes.

References

[1] R. Schaffenrath, D. Proksch, M. Kopp, I. Albasini, O. Panasiuk, A. Fensel, Benchmark for performance evaluation of shacl implementations in graph databases, Vol. 12173 LNCS, Springer, 2020, pp. 82–96. doi:10.1007/978-3-030-57977-7_6.

[2] J. E. L. Gayo, H. Knublauch, D. Kontokostas, Shacl test suite and implementation report (2024).
URL https://w3c.github.io/data-shapes/data-shapes-test-suite/implementation-report

[3] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, D. Kontokostas, Shacl (2018) 119–194doi:10.1007/978-3-031-79478-0_5.

[4] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, Databugger: A test-driven framework for debugging the web of data, Association for Computing Machinery, Inc, 2014, pp. 115–118. doi:10.1145/2567948.2577017.