What Are the Parameters that Affect the Construction of a Knowledge Graph?

David Chaves-Fraga[1(✉)], Kemele M. Endris[2,3], Enrique Iglesias[4], Oscar Corcho[1], and Maria-Esther Vidal[2,3]

[1] Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain
{dchaves,ocorcho}@fi.upm.es
[2] TIB Leibniz Information Centre for Science and Technology, Hanover, Germany
maria.vidal@tib.eu
[3] L3S Institute, Leibniz University of Hannover, Hannover, Germany
endris@l3s.de
[4] University of Bonn, Bonn, Germany
s6enrigle@uni-bonn.de

Abstract. A large number of datasets are made publicly available on a wide range of formats. Due to interoperability problems, the construction of RDF-based knowledge graphs (KG) using declarative mapping languages has emerged with the aim of integrating heterogeneous sources in a uniform way. Although the scientific community has actively contributed with several engines to solve the problem of knowledge graph construction, the lack of testbeds has prevented reproducible benchmarking of these engines. In this paper, we tackle the problem of evaluating knowledge graph creation, and analyze and empirically study a set of variables and configurations that impact on the behaviour of these engines (e.g. data size, data distribution, mapping complexity). The evaluation has been conducted on RMLMapper and the SDM-RDFizer, two state-of-the-art engines that interpret the RDF Mapping Language (RML) and transform (semi)-structured data into RDF knowledge graphs. The results allow us to discover unknown relations between these engines that cannot be observed in other configurations.

Keywords: Knowledge graph construction · RDFizers · Testbeds

1 Introduction

Following the FAIR principles [19] and Open data initiatives, the size of publicly available data has grown exponentially in the last decade, expecting a faster growth rate in the following years as a result of the advances in the technologies for data generation and ingestion. In order to extract values for existing datasets, several data integration approaches have been proposed in the literature [5]. The Semantic Web community has also proposed various approaches that enable the integration of data presented in diverse formats into a knowledge graph. Knowledge graphs comprise data and the knowledge that describe the main characteristics of the integrated data following a graph-based data model, e.g. RDF

H. Panetto et al. (Eds.): OTM 2019, LNCS 11877, pp. 695–713, 2019.
https://doi.org/10.1007/978-3-030-33246-4_43

[18]. With the aim of transforming structured data in tabular or nested formats like CSV, relational, JSON, and XML, into RDF knowledge graphs, diverse mapping languages have been proposed. Exemplary mapping languages include RDF Mapping Language (RML) [3], R2RDF [16], xR2RML [12], and R2RML [2], as well as tools like KARMA [4], SPARQL-Generate [11], and DIG [9]. Despite these developments, the absence of testbeds has prevented the community from conducting fair evaluations of the existing tools for knowledge graph creation. This testbed deficiency has also impeded for a holistic understanding about the pros and cons of the state of the art, as well as for clear directions to advance the area. Given the expected growth rate of available data, testbeds are demanded in order to devise the next generation of tools able to integrate data at scale.

Our Goals: We study the process of knowledge graph creation and analyze various variables and configurations that can impact on the performance of RDFizers – tools for transforming (semi)-structured data following mapping rules specified in the RDF Mapping Language (RML). The relevant parameters studied in this paper include selectivity of the joins between mapping rules, types of relations, and percentage of duplicates. We also present diverse examples that evidence the heterogeneous behaviour that each RDFizer may exhibit whenever small changes are conducted to the variables and the configurations of a testbed.

Our Approach: We devise a set of parameters involved in a knowledge graph construction process and we empirically show how they can impact on the behaviour of two existing RDFizers: RMLMapper[1] and SDM-RDFizer[2]; these engines are compliant with the RML specification according to a set of defined test-cases[3]. We develop a synthetic data generator for the generation of (semi)-structured data and RML mapping rules, that consider the identified set of parameters. The results of our empirical study provide evidence of the importance of the proposed set of variables and configurations during the evaluation of these tools. The testbeds used to conduct this evaluation are available online[4].

Contributions: Our main contribution includes the definition of various dimensions and set of variables to be considered during the creation of testbeds or to be measured while the evaluation of knowledge graph construction tools. Another contribution represents the empirical evaluation of the effects that the variables and configurations have on the tasks of knowledge graph creation. Furthermore, the results of the experimental study contribute to the understanding of the pros and cons of the studied RDFizers, and the directions that need to be followed in order to devise tools able to scale up to real-world scenarios.

The remainder of this article is structured as follows: Sect. 2 presents several examples where the evaluated tools exhibit unexpected behaviours. Section 3 analyses the variables and configurations that need to be considered in a testbed in order to ensure reproducibility and generality during benchmarking. Section 4

[1] https://github.com/RMLio/rmlmapper-java.
[2] https://github.com/SDM-TIB/SDM-RDFizer.
[3] http://rml.io/implementation-report/.
[4] https://github.com/SDM-TIB/KGC-Param-Eval.

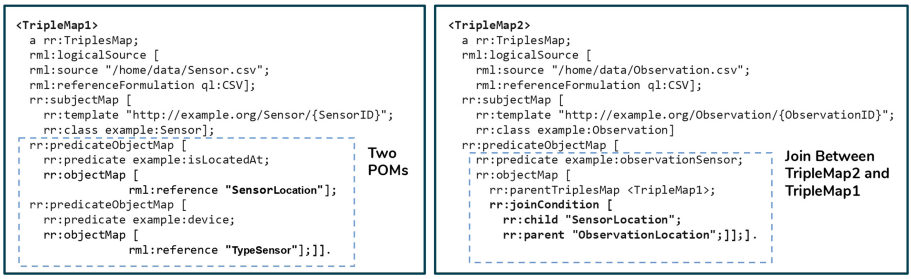


Fig. 1. Motivating example. RML triple maps to transform two CSV files into RDF. TripleMap1 is composed of two predicate-object, i.e., Two POM. TripleMap2 has a join to TripleMap1; Observation.csv (outer relation) is joined to Sensor.csv (inner relation) and the result, SensorID, is used as an object value.

reports on the results of the empirical study where several parameters and configurations are evaluated. Related work is presented in Sect. 5, and finally, Sect. 6 concludes and give insights for future work.

2 Motivating Examples

We motivate our work by analysing different scenarios where the performance of RMLMapper and SDM-RDFizer may be affected by changing the configuration of the testbeds utilised for empirically evaluating these engines. We aim at remarking the importance of taking into account different parameters during the definition of a testbed. We first describe a scenario where naïve parameters (size and format) leads to wrong decisions during the comparing of SDM-RDFizer and RMLMapper. The testbeds include a data source with one thousand rows, different number of predicate-object (POM) in RML triple maps, and diverse configurations of selectivity of triple map joins.

RML expresses mappings to transform sources represented in (semi)-structured format, e.g. CSV or XML, into RDF. Each mapping rule in RML, named RML triple map, is represented in RDF and consists of the following parts [3]:

- A *Logical Source* that refers to a data source from where data is collected.
- A *Subject Map* that defines the subject of the generated RDF triples.
- *Predicate-Object Maps* (POM) that expresses the predicate and the object the RDF triple to be generated; a triple map can comprise several POMs.
- A *Referencing Object Map*, that indicates the reference or join condition to another triple map; the subject URL is the referenced triple map corresponds to the result of the evaluation of the join.

Figure 1 illustrates two RML triple maps. `TripleMap1` is composed of two predicate-object maps, i.e. it is a Two POM mapping rule. `TripleMap2` has a referencing object map that joins the records of file Observation.csv with the records of the file Sensor.csv on the attributes `SensorLocation` and

`ObservationLocation`. The result of executing the join between the two RML triple maps is the identifier of the sensor that collected the observation; this value is used as the object value of the predicate `observationSensor`.

2.1 Impact of Number of Predicates and Objects in Mapping Rules

In this example, we execute a testbed where three different configurations of an RML mapping rule: Two-POM, Five-POM, and Ten-POM, i.e. they correspond to three mapping rules with two, five, and ten Predicate-Object Maps, respectively. Both RDFizers exhibit a similar behaviour while the number of predicate-object maps varies from two to five POMs, as shown in Table 1. However, when more complex mapping rules with more POMs are considered, the behaviour of the SDM-RDFizer and RMLMapper is not impacted equally. Moreover, the results suggest that RMLMapper execution time increases with the number of POMs, while the SDM-RDFizer seems to be slightly affected.

Table 1. Impact of Number of Predicate-Object Maps. Various predicate object maps (POM) specified in the mapping rules. The behaviour of the two RDFizers is similar when the mapping rules are simple (less than 5 POM) but it is different when more complex mappings are running (10 POM).

Engine	Execution time (secs.)	Number of results
	Two POM	
RMLMapper	0.92	2,000
SDM-RDFizer	1.72	2,000
	Five POM	
RMLMapper	1.84	5,000
SDM-RDFizer	1.85	5,000
	Ten POM	
RMLMapper	3.36	10,000
SDM-RDFizer	1.98	10,000

2.2 Impact of Join Selectivity

We consider the join selectivity, i.e. the cardinality of matching values from outer to the inner table (relation), in a referencing object map between two RML mapping rules; Fig. 1 depicts an example of a join between two RML triple maps. The join selectivity varies from **High Selectivity**, **Medium Selectivity**, and **Low Selectivity**, and Table 2 reports on the results of RMLMapper and SDM-RDFizer. First, it can be observed that the RMLMapper execution time increases by around 8 seconds, while the SDM-RDFizer behaviour is not equally affected by the selectivity of the join condition. As can be seen in Table 2, the SDM-RDFizer execution time (in seconds) increases from high to medium selectivity by 0.04 (from 2.16 to 2.20), then decreases from medium to low selectivity by 0.01 (from 2.20 to 2.19). On the other hand, the RMLMapper execution time

increases by 1.83 (from 38.6 to 40.43), and 5.63 (from 40.43 to 46.06) seconds from high to medium, and medium to low selectivity, respectively. As in the previous example, both engines are not equally affected by the complexity of the testbed.

Table 2. Impact of Join Selectivity. Impact of the join selectivity variable over the RDFizers with high, medium and low percentage of selectivity. While RMLMapper engine behaviour increases in terms of execution time when the selectivity decreases, the SDM-RDFizer behaviour is maintained, i.e. this variable affects to the first engine but it does not impact equality to the second one.

Engine	Execution time (secs.)	Number of results
	High Selectivity	
RMLMapper	38.6	2,100
SDM-RDFizer	2.16	2,100
	Medium Selectivity	
RMLMapper	40.43	23,000
SDM-RDFizer	2.20	23,000
	Low Selectivity	
RMLMapper	46.06	30,000
SDM-RDFizer	2.19	30,000

The uncorrelated behaviour of studied RDFizers shows clearly the need to considering diverse variables and configurations during the definition of testbeds, and thus, uncovering characteristics of these engines. In this paper, we analyze the parameters that might affect a knowledge graph construction process and evaluate some of the most problematic ones (e.g. partitioning, relation type) to remark the importance of setting them during testbed design.

3 Relevant Parameters for Testbed Design

In this section, we perform a study of the parameters that have impact on the knowledge graph construction engines. First, we identify the generic groups of parameters involved and the effect they produce in this process. Second, we provide a list of specific variables that influence the construction of knowledge graphs and determine the relationships among them. Finally, we describe each parameter in detail given the reasons why it might affect the performance of the engines. Together with these descriptions, we provide use cases over a set of parameters to illustrate the importance of involving them in a testbed definition.

As in every empirical study, we consider two groups of variables: independent and observed. The independent variables are those features that need to be specified in a benchmark to ensure that the performed evaluation is reproducible. These variables are grouped in five dimensions: mapping, data, platform, source, and output. On the other hand, observed variables correspond to those characteristics that are measured during the evaluation of the testbed and that may be influenced by independent variables. The observed variables are as follows:

- *Execution time:* The variable is in turn comprised of: *(i) Time for the first triple* (elapsed time between the engine starts and the first triple), *(ii) total execution time* required to produce all the triples of the knowledge graph.
- *Completeness:* Number of returned triples in relation to all the RDF triples that should be created according to the data and input mappings.

The relations among independent and observed variables are presented in Table 3. These variables are described in detail in the next section.

3.1 Mapping Dimension

This dimension involves the variables that characterise the mappings in terms of their structure and evaluation. Regarding the structure, there are various aspects

Table 3. Variables and Configurations. Set of variables and configurations that impact on the behaviour of the tools for knowledge graph creation. Independent variables are divided into five groups and the impact on the observed variables is depicted.

	Independent variables	Observed variables	
		Execution time	Completeness
Mapping	mapping order	✓	
	# triplesMap	✓	✓
	# predicateObjectMaps	✓	✓
	# predicates	✓	✓
	# objects	✓	✓
	# joins	✓	✓
	# named graphs	✓	✓
	join selectivity	✓	✓
	relation type	✓	✓
	object TermMap type	✓	
Data	dataset size	✓	
	data frequency distribution	✓	
	type of partitioning	✓	✓
	data format	✓	✓
Platform	cache on/off	✓	
	RAM available	✓	
	# processors	✓	
Source	distribution data transfer	✓	✓
	initial delay	✓	
	access limitation	✓	✓
Output	Serialization	✓	✓
	Duplicates	✓	✓
	Generation type	✓	✓

to be considered: mapping order, the complexity of the mapping in terms of number of predicates, objects, and the join type and selectivity.

Mapping Order. Although the mappings are usually defined using an RDF serialisation, where the order is not relevant, the features of each `rr:tripleMap` (e.g. joins) can affect the execution plan generated by each tool, having, thus, a potential negative impact on the total execution time.

Table 4. Impact of Relation Types. Various relation types in a join specified in the mapping rules. N corresponds to 15 values in the case of 1−N and N−1 relations, N and M has 10 values in the last case. RMLMapper execution time is not affected by 1−N and N−1 relation types while it is affected by N−M relations. SDM-RDFizer performs better in N−1 than 1−N but the time increases in N−M.

Engine	Execution time (secs.)	Number of results
	1−1	
RMLMapper	42.86	25,000
SDM-RDFizer	2.19	25,000
	1−N	
RMLMapper	43.34	22,490
SDM-RDFizer	2.19	22,490
	N−1	
RMLMapper	43.26	22,490
SDM-RDFizer	2.15	22,490
	N−M	
RMLMapper	78.64	25,200
SDM-RDFizer	2.33	25,200

Mapping Complexity. The number of properties defined in a rule mapping, e.g. number of predicates, objects, or named graphs may affect the observed variables because the number of triples to be generated, is related to what is specified in the mappings. Additionally, the `rr:termtype` of the `rr:objectMap` can affect the total execution time because the cost of generating a constant or a template is not the same. Finally, the join selectivity (as illustrated in Sect. 5) and types of relation have also impact on the performance of an RDFizer. In Table 4, we illustrate how the relation type affects the total execution time of the studied RDFizers. In this case, the behaviour of the RMLMapper only occurs when the relation type is N−M. However, the SDM-RDFizer behaviour is impacted during the evaluation of 1−N and N−M joins. Additionally, during the join evaluation, there are many cases when duplicates are generated, then the engines have to eliminate them. Table 5 reports on how the generation of the duplicates –during the join condition evaluation– affects the total execution time. RMLMapper decreases its performance while the percentage of duplicates increases. However, SDM-RDFizer implements optimised data structured that allow for efficiently eliminating duplicates, and seems not to be equally affected by the complexity of these configurations, e.g. number of duplicates.

3.2 Data Dimension

We describe the independent variables related with the original data that are required for the generation of a knowledge graph. Each dataset can be defined in terms of **size** and **total number of sources**. The first characteristic impacts on the number of triples that will be generated, affecting, thus, the total execution time. Additionally, the total number of sources that have to be processed to generate a knowledge graph may also affect the total execution time.

Table 5. Impact of duplicates generation during join evaluation. Various configurations of duplicates generated during the evaluation of a join between two triple maps. While the complexity of the configuration increases (more percentage of duplicates), the RMLmapper decreases its performance. Surprisingly, the SDM-RDFizer seems not to be affected by the complexity of the testbeds, and improves its performance even when the complexity of testbeds increases.

Engine	Execution time (secs.)	Number of results
	Low percentage of duplicates	
RMLMapper	37.94	20,027
SDM-RDFizer	2.01	20,027
	Medium percentage of duplicates	
RMLMapper	39.201	20,105
SDM-RDFizer	1.87	20,105
	High percentage of duplicates	
RMLMapper	40.81	20,263
SDM-RDFizer	1.89	20,263

Partitioning and **distribution** are important variables considered in the generation of a knowledge graph. Partitioning refers to the way that a dataset is fragmented, and distribution is the format (e.g. CSV, JSON) of each partition. A dataset can be presented in only one format or in multiples formats, and this variable affects not only the total execution time but also the completeness of the results. A dataset may be fragmented into disjointed partitions; the partition may be horizontal, vertical or a combination of both. Horizontal partitioning fragments the dataset, so that, they represent different instances of the same resource (equal *TripleMaps* with different sources). Vertical partitioning produces fragments that contain at least one property of the same resources (*TriplesMaps* with *JoinCondition*). The horizontal partitioning may affect the completeness of a knowledge graph while the vertical partitioning has an influence on the execution time. Table 6 compares the behaviour of the RMLMapper and SDM-RDFizer with different configurations. The two engines increase their execution time when the horizontal partitioning is compared with and without including replication. However, RMLMapper decreases its execution time when the vertical partitions with and without replication are compared, while SDM-RDFizer execution time increases. Thus, even SDM-RDFizer is tailored towards efficient duplicate elimination, data partitioning– with and without replication – seems to affect the SDM-RDFizer performance.

Table 6. Impact of Partitioning: Various configurations of vertical and horizontal partitioning with and without duplicates. The two engines perform similar with the two cases of the horizontal partitioning but they have different behaviours in vertical partitioning.

Engine	Execution time (secs.)	Number of results
	Horizontal partitioning without replication	
RMLMapper	1,904.31	310,000
SDM-RDFizer	4.84	310,000
	Vertical partitioning without replication	
RMLMapper	2,067.77	310,000
SDM-RDFizer	4.73	310,000
	Horizontal partitioning with replication	
RMLMapper	2,276.98	310,000
SDM-RDFizer	5.86	310,000
	Vertical partitioning with replication	
RMLMapper	2,024.66	310,000
SDM-RDFizer	4.98	310,000

3.3 Platform Dimension

The platform dimension comprises variables related with the hardware used to create a knowledge graph. We include a set of variables related with the system cache, the available RAM memory for running the tool, and the number of processors of the machine. The **cache** and the **available RAM memory** may affect the total time execution. We recommend that other parameters, like the versions of operating system and processor, should be specified in the evaluation setup. To conclude, during testbed design, the platform and hardware specification requires attention and needs to be defined in detail.

3.4 Source Dimension

In this dimension, we consider different variables related with the original sources defined in the mapping rules. The **distribution data transfer**, which corresponds to the transfer time of a file by a Web service–in case the data is not in a local machine– will definitely influence the total execution time. Additionally, the **initial delay** of each engine to configure the corresponding wrappers for each data format and the **limit access** for example, a database, also strikes out the execution time and the completeness of the results.

3.5 Output Dimension

In this dimension, we consider the variables related with the output of the generation process. The **serialization** impacts on the total execution time; the effect will depend on the size of the output and the number of times the processor

has to access the disk to store the output. **Generation type** represents how an RDFizer generates a knowledge graph. The generation can be continuous, e.g. the SDM-RDFizer stores each RDF triple in a file once it is generated. Contrary, the generation can be in-memory, e.g. RMLMapper stores the output when the knowledge graph is created completely. Finally, the RDFizers usually can have a flag for removing **duplicates**; this operation has to be specified in the setup because it strikes out the completeness and also the total execution time. The efficiency of the RDFizers components that eliminate duplicates, can be captured by observing the variables of this dimension.

Table 7. Datasets. Properties of datasets used in the empirical evaluations.

Dataset	#rows	#columns	#tables
1K	1,000	2	2
10K	10,000	2	2
50K	50,000	2	2

As can be observed in the results reported in this section, the behaviour of the studied engines is not equally affected by the different independent variables. Thus, benchmarks need to include all these variables in order to provide a holistic overview of the performance of the studied engines, and ensure general and reproducible evaluations.

4 Experimental Evaluation

The goal of our experiment is to assess the impact of the discussed variables and configurations during the evaluation of existing knowledge graph creation tools. We aim at answering the following research questions: **(RQ1)** What is the effect of mixing different variables in one testbed? **(RQ2)** What is the impact of considering configurations of different complexity of the same variable in one testbed? **(RQ3)** Do the different variables and configurations influence in the behaviour of existing knowledge graph creation tools? To answer these research questions, we set up the following experimental studies:

Datasets. For this evaluation, we generated three different datasets with 1,000 (1K), 10,000 (10K), and 50,000 (50K) rows, and various number of columns based on the tested parameters; Table 7 shows the properties of the datasets generated for `Relation Type`, `Join Duplicates`, and `Join Selectivity` evaluations. For the *Dataset Size (Näive)* parameter, we generated the same number of rows as in Table 7, but with 30 columns. During the experiments, we only considered the CSV file format to represent the generated tables.

Configurations. We consider different configurations for the above-discussed variables in each dimension. `Dataset Size Configurations:` (1) SDM-RDFizer 1K; (2) SDM-RDFizer 10K; (3) SDM-RDFizer 50K; (4) RMLMapper

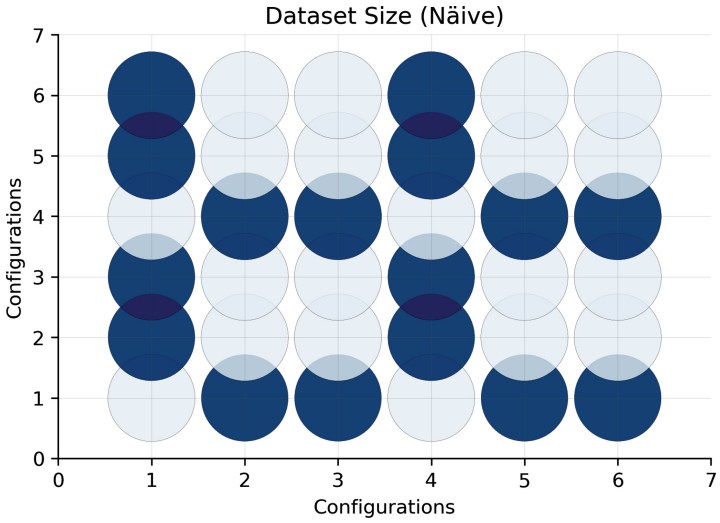


Fig. 2. Comparison of Knowledge Graph Creation Tool on Different Dataset Sizes (Naïve). The first three configurations, i.e. 1, 2, and 3 in x-axis and y-axis, correspond to SDM-RDFizer on datasets 1K, 10K, and 50K, respectively. The last three configurations, i.e. 4, 5, and 6 on x-axis and y-axis, correspond to RMLMapper 1K, 10K, and 50K, respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation. The number of blue bubbles suggests that both systems exhibit similar behaviour. (Color figure online)

1K; (5) RMLMapper 10K; and (6) RMLMapper 50K. In each configuration of this parameter, we only use one data file. `Relation Type configurations:` (1) SDM-RDFizer 1-N; (2) SDM-RDFizer N−1; (3) SDM-RDFizer N−M; (4) SDM-RDFizer Combinations (all relation types); (5) RMLMapper 1−N; (6) RMLMapper N−1; (7) RMLMapper N−M; and (8) RMLMapper Combinations (all relation types). For relation cardinality, we evaluated $N = \{1, 5, 10, 15\}$ and $M = \{1, 3, 5, 10\}$. In addition, we set the percentage of rows that involve in those relation types to 25%, i.e. 25% of the overall rows from outer table have a matching join value to inner table, and 50%, respectively. `Join Duplicate configurations:` (1) SDM-RDFizer Low, (2) SDM-RDFizer High, (3) RMLMapper Low, (4) RMLMapper High. `Low` Join Duplicates refer to datasets with low percentage of duplicates, i.e. from 5% to 20% of data generated could have duplicates due to the join conditions, similarly `High` Join Duplicates refer to higher percentage of duplicates, i.e. from 30% to 50% of data generated could be duplicated. `Join Selectivity Configurations:` (1) SDM-RDFizer High; (2) SDM-RDFizer Low; (3) RMLMapper High; and (4) RMLMapper Low. In this case, the join selectivity `High` represents how many time the join condition matches the values in the inner join file from 5% to 20% of the overall rows, while `Low` means that the join condition matches range from 60% to 100% of the overall number of rows. As previously shown, we hypothesise

that these configurations allow us to uncover patterns in the behaviour of these engines that could not be observed if only naïve variables were studied.

Metrics. We report on the following metrics or observed variables: **(a)** *Execution Time*: Elapsed time between execution of RDFizer and the delivery of the results. **(b)** *Number of Results*: Number of triples generated by the RDFizer.

Implementations. The SDM-RDFizer and the testbeds are implemented in Python 3.6; the SDM-RDFizer is publicly available[5]. Furthermore, Jupiter Notebooks are available to generate the data and plot the results. Additionally, we have created a Docker image to run the testbeds and reproduce the experimental results[6]. The experiments were run in an Intel(R) Xeon(R) equipped with a CPU E5-2603 v3 @ 1.60GHz 20 cores, 100G memory with Ubuntu 16.04LTS.

Testbeds. Results of each configurations are ordered from lower to higher complexity and compared using the Pearson's correlation. A high positive value of correlation between two configurations indicates that the corresponding RDFizers had a similar behavior, i.e. the trends of execution time of the tools are similar; they are represented with blue bubbles in our plots. When a configuration is compared to itself, the Pearson's correlation reaches the highest value (1.0), represented with grey bubbles in our plots. On the other hand, a negative value indicates that there is an inverse correlation between the RDFizers, i.e. they exhibit an opposite behaviour; they are represented with red bubbles.

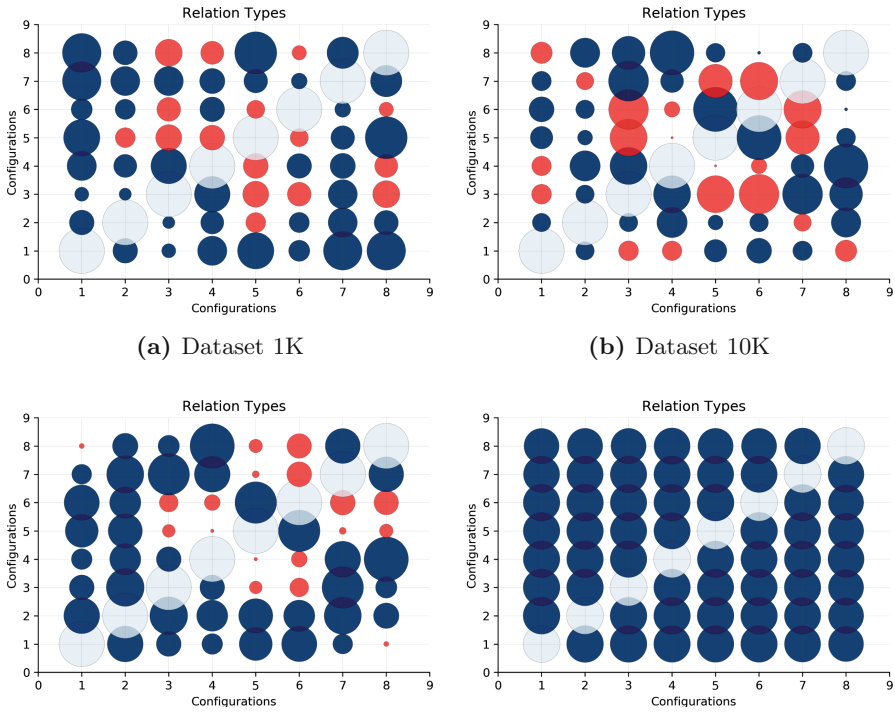
Discussion of the Observed Results

We observe that the behaviour of the engines can be affected when multiple variables are involved in a testbed (e.g. size and relation type) or when different levels of complexity of a variables (e.g. low, high join selectivity). We discuss the obtained results during our evaluation over the different configurations and parameters involved in each experiment:

Dataset Size (Naïve): Figure 2 depicts the comparison of RDFizers when the dataset size is considered. When `configuration 1` is compared to itself, the Pearson's correlation value is 1.0; additionally, it is high and positive when it is compared to `configurations 2, 3, 5, and 6` (large blue bubbles). Using this parameter, the correlation analysis suggests that both RDFizers behave similarly in all configurations. Moreover, this indicates that only considering the data size is not enough to uncovered the properties of the studied engines.

Relation Types: Figure 3 reports on the correlation of different configurations for various join relation types. We can observe in Figs. 3a, b, and c several red bubbles, indicating a negative correlation in the behaviour of the compared configurations and RDFizers. Contrary, Fig. 3d does not depict any red bubble, suggesting thus, that the two RDFizers in all the configurations exhibit the same

[5] https://github.com/SDM-TIB/SDM-RDFizer.
[6] https://github.com/SDM-TIB/KGC-Param-Eval.



(a) Dataset 1K **(b)** Dataset 10K

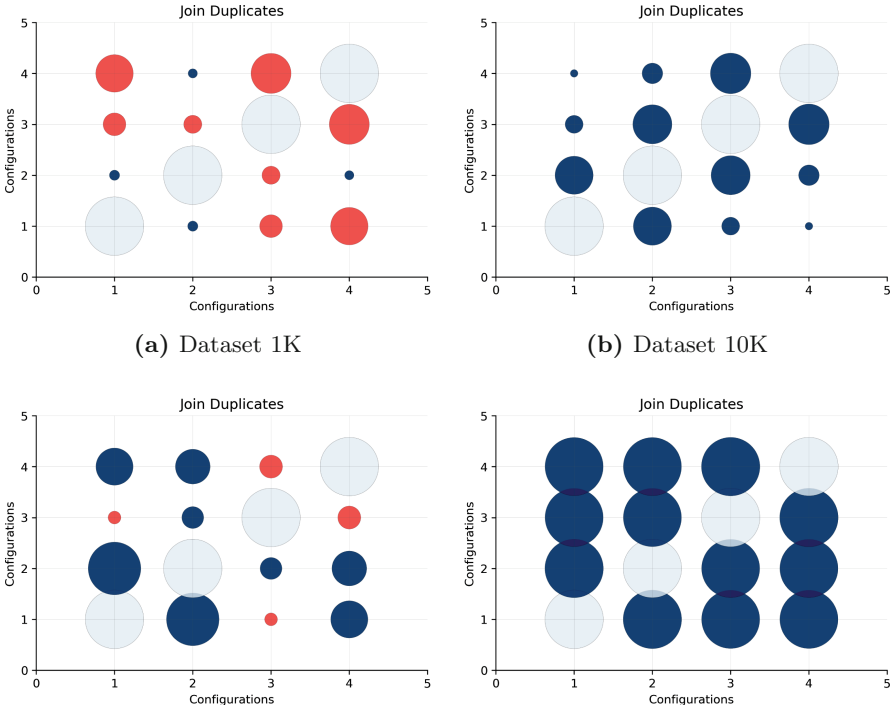
(c) Dataset 50K **(d)** Combination of 1K, 10K, and 50K

Fig. 3. Comparison of Knowledge Graph Creation Tools on Different Types of Relations. The first four (4) configurations, i.e. 1–4 in both x-axis and y-axis, represent results of SDM-RDFizer on *1−N*, *N−1*, *N−M*, and combination of all relations types, respectively. The later configurations, 5–8 both in x-axis and y-axis, shows results of RMLMapper on *1−N*, *N−1*, *N−M*, and combination of all relations types, respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation while red bubbles show a negative correlation. The plots reveal that both type of relations and size of the dataset need to be taken into account to uncover patterns in the behaviour of the engines. (Color figure online)

behaviour. These results clearly illustrate the need of considering different configurations and parameters in order to avoid drawing wrong conclusions about the main characteristics of existing tools.

Join Duplicates: Figure 4 depicts the correlation between different configurations when different setting of duplicates are produced during the execution of joins between triple maps. As can be observed, Figs. 4a, and c include several red bubbles that indicate an opposite behaviour of the RDFizers. Contrary, Figs. 4b, and d suggest that both engines behave similarly.

Join Selectivity: Figure 5 shows the correlation between different configurations for the selectivity of join conditions. Similarly, these testbeds reveal



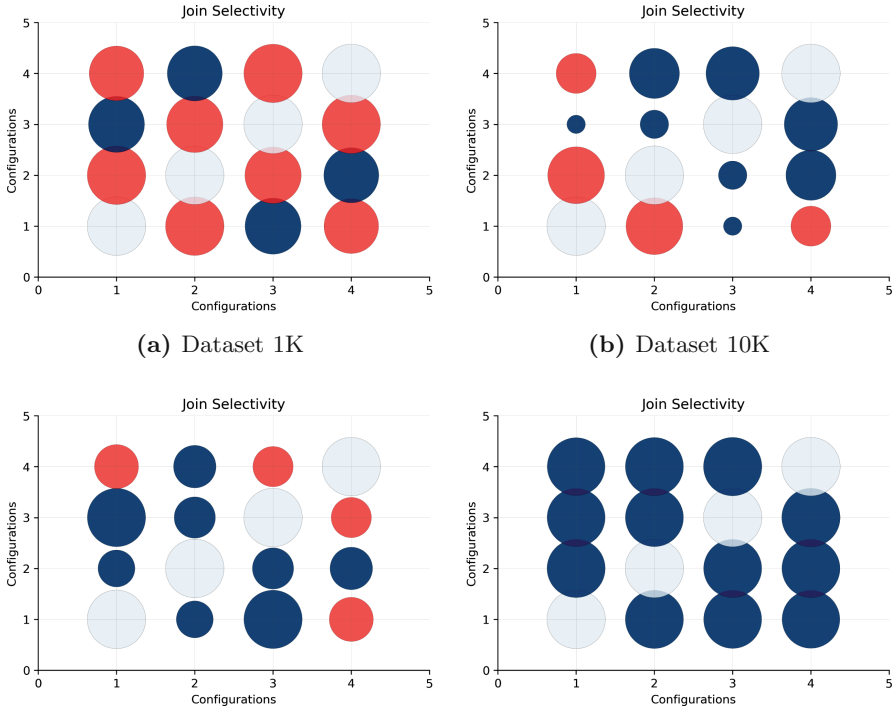
(a) Dataset 1K **(b)** Dataset 10K

(c) Dataset 50K **(d)** Combination of 1K, 10K, and 50K

Fig. 4. Comparison of Knowledge Graph Creation Tools on Duplicates during Join. The first two (2) configurations, i.e., 1–2 on x-axis and y-axis, represent results of SDM-RDFizer on datasets with *low* (5%–20% of data) number of duplicates and *high* (30%–50% of data) number of duplicates generated during joins, respectively. The last two configurations, i.e., 3–4 on x-axis and y-axis, represent results of RMLMapper on datasets with *low* number of duplicates and *high* number of duplicates generated during joins, respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation while red bubbles show a negative correlation. Results evidence that both join duplicates and dataset size are needed for characterising an engine performance. (Color figure online)

contradicting patterns in the behaviours of the studied RDFizers. On the one hand, Figs. 5a, b, and c are composed of several red bubbles and indicate that these engines perform differently whenever the selectivity of the join condition is changed. Surprisingly, when the size of these datasets are also taken into account in the testbed (Fig. 5d), these patterns are hidden, and the results of the evaluation suggest that both RDFizers perform similarly whenever the selectivity of the join condition is changed.

The results reported in this experimental study provide clear evidence of the importance of the variables and configurations that composed the methodology devised in this work. Actually, in the four studied cases, they reveal important



(a) Dataset 1K **(b)** Dataset 10K

(c) Dataset 50K **(d)** Combination of 1K, 10K, and 50K

Fig. 5. Comparison of Knowledge Graph Tools on Join Selectivity. The first two configurations, i.e., 1–2 on x- and y-axis represent SDM-RDFizer on joins with *high* selectivity (5%–20% of data) and joins with *low* selectivity (60%–100% of data), respectively. Configurations 3 and 4 represent RMLMapper on joins with *high* selectivity (5%–20% of data) and joins with *low* selectivity (60%–100% of data), respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation while red bubbles show a negative correlation. Dataset size and join selectivity affect both engines differently. (Color figure online)

patterns that could not be observed whenever other parameters were studied simultaneously. Based on these observations, we can conclude that these variables and configurations should be included in the benchmarks in order to ensure that the characteristics of knowledge graph creation engines are uncovered. Thus, these observations allow us to answer our three research questions: **RQ1**, **RQ2**, and **RQ3**. We encourage developers and users of knowledge graph creation tools to bear in mind them during benchmarking in order to draw clear conclusions about the performance of their tools.

5 Related Work

R2RML [2] is a W3C recommendation for describing mapping rules to generate knowledge graphs from relational databases. Currently, diverse approaches focus on providing query-translation techniques (SPARQL-to-SQL) and optimisations over the resulted queries like morph-RDB [14] or Ultrawrap [17]. Additionally, other tools focus on generating RDF graphs are supported by R2RML (e.g. DB2Triples[7] and R2RMLParser[8]), while other approaches extend R2RML (e.g xR2RML [12], RMLC-Iterator [1], and RML [3]). Particularly, RML is defined as an extension of R2RML to provide support for other formats like JSON, XML or CSV. YARRRML [8] is other serialization of RML using the YAML format; it improves the creation and maintainability of the mapping rules. There are multiples tools that support the RML specification. For example, CARML[9] executes RML rules and includes additional features like MultiTermMap (to deal with arrays) and XML namespace (to improve XPath expressions). GeoTriples [10] is focused on the generation of RDF from geospatial data while RocketRML[10] is an RML engine implemented using the NodeJS framework. Similar to RMLMapper and SDM-RDFizer, all these engines are able to check compliance with the RML specification using a set of defined test-cases [7][11]. They results of the execution of the test-cases is included in the implementation report[12]. Despite the great effort conducted by the Semantic Web community, because of the lack of testbeds, reproducible empirical studies have not been conducted so far. In this paper, we conduct an evaluation involving a set of variables and configurations that will allow the community to define testbeds of different complexity, enabling thus, the understanding of the main strengths and limitations of the state of the art. Furthermore, the analysis of these variables and configurations will enable developers to better understand main features of their tools.

The Semantic Web community has also actively worked on the definition of several testbeds. As an example of the existing contributions, we can mention the work done in the area of federated query processing. Specifically in this area, FedBench [15] is an exemplar benchmark; it comprises three datasets, (i.e. cross-domain, life science and SP^{2}Bench), 25 queries, and two proposed metrics to measure a federated engine performance, (i.e. total execution time and number of requests to SPARQL endpoints). LSLOD is another benchmark [6] that consists of 20 queries –classified as simple and complex; it comprises ten real-world datasets from the Life Sciences domain. LSLOD proposes to measure the performance in terms of total triple pattern-wise sources selected (TTPWSS), the number of SPARQL queries ASK, the source selection time, the overall query execution time, and the result set completeness. Finally, Montoya et al. [13] identify a main drawback in existing benchmarks for SPARQL

[7] https://github.com/antidot/db2triples.
[8] https://github.com/nkons/r2rml-parser.
[9] https://github.com/carml/carml.
[10] https://github.com/semantifyit/RML-mapper.
[11] http://rml.io/test-cases/.
[12] http://rml.io/implementation-report/.

federated queries; particularly, Montoya et al. focus on the study of FedBench and illustrate how the lack of considering independent variables impact on the effectiveness of the benchmark, e.g. complexity of the queries, data used, platforms involved, and endpoints. They show the relevance of these variables in order to ensure reproducibility of the results observed during an empirical evaluation. In this paper, we build on the work conducted by Montoya et al. and present a similar evaluation composed of diverse variables and configurations that strike out the performance of the tools for knowledge graph creation. Our ambition is that the results of this work will facilitate the definition of suitable testbeds able to ensure reproducible experimental studies that evaluate solutions to the problem of knowledge graph creation.

6 Conclusions and Future Work

In this paper, we performed an in-depth analysis of the variables and configurations that impact on the behaviour of two RDFizers. The observation that existing RDFizers exhibit heterogeneous behaviours whenever small changes in the testbeds are conducted, motivated the need of conducting this study involving a set of parameters that can reveal patterns in the behaviour of the studied engines. Additionally, the lack of testbeds encouraged us to acquit the definition of variables and configurations that enable for the characterisation of the pitfalls of existing RDFizers and for identifying the list of challenges and research directions in the state of the art. With the proposed analysis and the results of the experimental study, we contribute with an empirical configuration that can be reused for the evaluation of other knowledge graph creation tools and mapping languages (e.g. SPARQL-Generate, TARQL, or R2RML). Furthermore, our set of variables and configurations can be utilised as a guideline during testing and benchmarking. One of the main lessons learned during the definition and evaluation of our approach, is that none of the evaluated RDFizers behaves consistently whenever the complexity of the testbeds increases. Our ambition is that the reported results inspire the community to define general testbeds that facilitate the understanding of the state of the art and the development of novel tools for the creation of knowledge graphs at large scale. In the future, we plan to define testbeds and conduct a more detailed analysis of other RDFizers and mapping languages. Moreover, we envision to motivate the community to conduct a joint effort in the definition of benchmarks that enable for fair evaluations of knowledge graph creation tools with replicable and generalizable results.

Acknowledgements. This work is partially supported by the EU H2020 RIA funded project iASiS with grant agreement No. 727658, by the Ministerio de Economía, Industria y Competitividad (Spain), by EU FEDER funds under DATOS 4.0: RETOS Y SOLUCIONES - UPM Spanish National Project (TIN2016-78011-C4-4-R), and by an FPI grant (BES-2017-082511).

References

1. Chaves-Fraga, D., Priyatna, F., Perez-Santana, I., Corcho, O.: Virtual statistics knowledge graph generation from CSV files. In: Emerging Topics in Semantic Technologies: ISWC 2018 Satellite Events, Studies on the Semantic Web, vol. 36, pp. 235–244. IOS Press, Amsterdam (2018)
2. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language, W3C recommendation, 27 September 2012. World Wide Web Consortium, Cambridge (W3C) (2012). www.w3.org/TR/r2rml
3. Dimou, A., Sande, M.V., Colpaert, P., Verborgh, R., Mannens, E., de Walle, R.V.: RML: a generic language for integrated RDF mappings of heterogeneous data. In: Proceedings of the Workshop on Linked Data on the Web Co-located with the 23rd International World Wide Web Conference (WWW 2014) (2014)
4. Gupta, S., Szekely, P.A., Knoblock, C.A., Goel, A., Taheriyan, M., Muslea, M.: Karma: a system for mapping structured sources into the semantic web. In: The Semantic Web: ESWC 2012 Satellite Events - ESWC 2012 Satellite Events, Heraklion, Crete, Greece, 27–31 May 2012. Revised Selected Papers, pp. 430–434 (2012)
5. Halevy, A.Y.: Information integration. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, 2nd edn. Springer, New York (2018)
6. Hasnain, A., et al.: BioFed: federated query processing over life sciences linked open data. J. Biomed. Semant. **8**(1), 13 (2017)
7. Heyvaert, P., et al.: Conformance test cases for the RDF mapping language (RML). In: 1st Iberoamerican Knowledge Graphs and Semantic Web Conference (2019, to appear)
8. Heyvaert, P., De Meester, B., Dimou, A., Verborgh, R.: Declarative rules for linked data generation at your fingertips!. In: Gangemi, A., et al. (eds.) ESWC 2018. LNCS, vol. 11155, pp. 213–217. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98192-5_40
9. Knoblock, C.A., Szekely, P.A.: Exploiting semantics for big data integration. AI Mag. **36**(1), 25–38 (2015)
10. Kyzirakos, K., et al.: GeoTriples: transforming geospatial data into RDF graphs using R2RML and RML mappings. J. Web Semant. **52**, 16–32 (2018)
11. Lefrançois, M., Zimmermann, A., Bakerally, N.: Flexible RDF generation from RDF and heterogeneous data sources with SPARQL-generate. In: Ciancarini, P., et al. (eds.) EKAW 2016. LNCS (LNAI), vol. 10180, pp. 131–135. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58694-6_16
12. Michel, F., Djimenou, L., Zucker, C.F., Montagnat, J.: Translation of relational and non-relational databases into RDF with xR2RML. In: 11th International Conference on Web Information Systems and Technologies (WEBIST 2015), pp. 443–454 (2015)
13. Montoya, G., Vidal, M.-E., Corcho, O., Ruckhaus, E., Buil-Aranda, C.: Benchmarking federated SPARQL query engines: are existing testbeds enough? In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012. LNCS, vol. 7650, pp. 313–324. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35173-0_21
14. Priyatna, F., Corcho, Ó., Sequeda, J.F.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In: 23rd International World Wide Web Conference, WWW 2014, Seoul, Republic of Korea, 7–11 April 2014, pp. 479–490 (2014)

15. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: a benchmark suite for federated semantic data query processing. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_37
16. Sequeda, J.F., Arenas, M., Miranker, D.P.: OBDA: query rewriting or materialization? in practice, both! In: The Semantic Web - ISWC 2014–13th International Semantic Web Conference, Riva del Garda, Italy, Proceedings, Part I, 19–23 October 2014, pp. 535–551 (2014)
17. Sequeda, J.F., Miranker, D.P.: Ultrawrap: SPARQL execution on relational data. J. Web Semant. **22**, 19–39 (2013)
18. Vidal, M., Endris, K.M., Jazashoori, S., Sakor, A., Rivas, A.: Transforming heterogeneous data into knowledge for personalized treatments - a use case. Datenbank-Spektrum **19**(2), 95–106 (2019)
19. Wilkinson, M.D., et al.: The FAIR guiding principles for scientific data management and stewardship. Sci. Data **3**, 160018 (2016)