Knowledge Graph Creation Challenge: Results for SDM-RDFizer

Enrique Iglesias[1,2,*], Maria-Esther Vidal[1,2,3]

[1]L3S Research Center, Hannover, Germany
[2]Leibniz University of Hannover, Hannover, Germany
[3]TIB Leibniz Information Centre for Science and Technology, Hannover, Germany

Abstract

The amount of data being generated in recent years has increased drastically. Thus, a unified schema must be defined to bring multiple data sources into a single format. For that reason, the use of knowledge graphs has become much more commonplace. When creating a knowledge graph, different parameters affect the creation process, like the size and heterogeneity of the input data and the complexity of the input mapping. Multiple knowledge graph creation engines have been developed that handle these parameters differently. Therefore, a benchmark is needed to be defined to evaluate the performance of these engines. KGCW 2023 Challenge dataset presents a wide array of test cases to discover each engine's strengths and weaknesses and determine which engine is best suited for each case. This work reports the results of evaluating the performance of SDM-RDFizer while using this dataset.

Keywords

Knowledge Graph Creation, Data Integration System, RDF Mapping Languages

1. Introduction

Due to the substantial surge in data volume in recent times, using knowledge graphs (KGs) has become increasingly prevalent. Consequently, devising efficient methods for generating KGs has become indispensable. Multiple KG creation engines have arisen over the years, like RMLMapper [1], RocketRML [2], SDM-RDFizer [3], and Morph-KGC [4] alongside the *RDF Mapping Language* (RML) [5] which is a mapping language that defines the structure of a KG by following the rules established by the *Resource Description Framework* (RDF)[1]. However, there exist parameters that hinder the KG creation process (Chave-Fraga et.al. [6]) like the size of the data source (number of records and properties), duplicate data rate, and the complexity of the mapping. They influence cost (e.g., time and memory) of KG creation processes.
Multiple benchmarks have been defined to evaluate the performance of RML engines, like GTFS-Madrid-Bench [7] and SDM-Genomic-Datasets[2]. These benchmarks present multiple

Hersonissos'23: Fourth International Workshop On Knowledge Graph Construction, May 28, 2023, Hersonissos, GR
*Corresponding author.
†These authors contributed equally.
iglesias@l3s.de (E. Iglesias); maria.vidal@tib.eu (M. Vidal)
0000-0002-8734-3123 (E. Iglesias); 0000-0003-1160-8727 (M. Vidal)

[1]https://www.w3.org/TR/2004/REC-rdf-primer-20040210/
[2]https://doi.org/10.6084/m9.figshare.17142371.v2

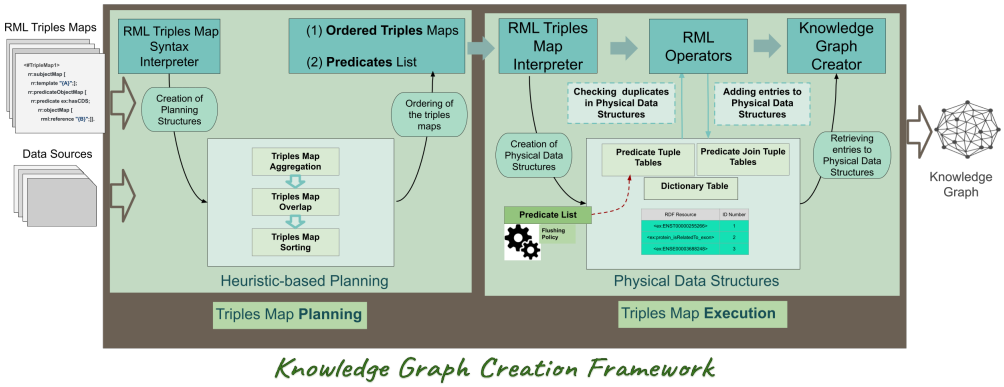

Figure 1: The Architecture of SDM-RDFizer

RML mappings with different level of complexity and data sources of different sizes, but they do not cover all possible parameter configurations that an RML engine should cover. Therefore, KGCW 2023 Challenge dataset was developed to cover as many test cases as possible. These test cases contain joins of multiple levels of complexity, data sources of multiple sizes in terms of the number of records and properties, the number of triples maps (TMs), different data duplicate rates, and cases with empty values. This challenge dataset aims to discover the strengths and weaknesses of existing state-of-the-art engines and determine which engines are suitable for which test cases. This work will present the results of evaluating the performance of SDM-RDFizer with the KGCW 2023 Challenge dataset.
This paper is organized into three additional sections. Section 2 defines everything regarding SDM-RDFizer, like what techniques, data structures, and physical operators it has for optimizing the KG graph creation process. Section 3 reports on the results of the empirical evaluation, including the definition of the dataset and the corresponding analysis. Finally, Section 4 illustrates the conclusions and future steps for SDM-RDFizer.

2. SDM-RDFizer

SDM-RDFizer [3] is a knowledge graph creation engine that is RML compliant. SDM-RDFizer is comprised of two modules: **Triples Maps Planning** (TMP) and **Triples Maps Execution** (TME). Each module has different data structures that optimize different aspects of the KG graph creation process. TMP defines an execution order for the triples maps so that the memory usage of the tool is kept at a minimum. TME is the module that generates the KG by following the order defined by TMP and using data structures and operators that optimize different aspects of the creation process, like duplicate removal, join execution and data compression. The following sections define these modules, data structures, and operators in more detail.

2.1. Triples Map Planning

The Triples Map Planning (TMP) module reorders TMs so that the most selective mappings are evaluated first; meanwhile, non-selective mappings are executed last. In other words, the TMs that have the most overlap with other TMs are executed first. TMP organizes the TMs and data sources so that the number of triples stored in memory is kept at a minimum. Therefore, when a triple is generated, the amount of comparison needed to determine if it is a duplicate is reduced and the KG creation process is quicker. TMP defines two data structures: **Organized Triples Maps List** and **Predicate List**.
Organized Triples Maps List (OTML) groups the TMs by their data source. OTML is only used when files are used as data sources (e.g., CSV, JSON and XML). During the TMP phase, TMs are classified based on the logical data source format (i.e., CSV, JSON, and XML). Afterward, they are grouped by their data source; thus, a data source is opened once to execute all the TMs.
Predicate List (PL) groups TMs by their predicates by creating a list of TMs associated with a particular predicate. PL has two purposes; the first is to determine when the Predicate Tuple Table (PTT) associated to a certain predicate can be flushed and the second is to organize the OTML. Each time a TM is evaluated, it is removed from the list of the associated predicate; when the list is empty, the corresponding PTT can be flushed.

2.2. Triples Map Execution

The Triples Map Execution (TME) module generates the KG; it follows the order established by the TMP module when executing the TMs. This module presents multiple data structures that optimize different aspects of the KG creation process, like duplicate removal, join execution, and data compression. Multiple novel operators are defined to transform different types of TMs.

2.2.1. Data Structures

Dictionary Table (DT) encodes each RDF resource generated during the KG creation process with an identification number and each identification number is encoded in base 36. Therefore, triples are stored not as a series of resources but as numbers.
Predicate Tuple Table (PTT) stores all the triples generated so far for a predicate p. PTTs correspond to hash tables where the hash key of an entry corresponds to the encoding of the subject and object of a generated RDF triple, and the value of the entry is the encoding of the RDF triple. The subject and object are stored with their corresponding identification number. The purpose of a PTT is duplicate removal; each triple generated is compared to its corresponding PTT. If the triple exists in the PTT, then it is discarded. If the triple does not exist in the PTT, it is added to the PTT and the KG.
Predicate Join Tuple Table (PJTT) stores the resulting subjects that come from executing a join condition. It is implemented as an index hash table to the data source of the parent triples map in a join condition. A PJTT key corresponds to encoding each value(s) of the attributes in the join condition. The value of a key in a PJTT corresponds to encoding the subject values in the data source of the parent TMs associated with the values of the attributes in the hash key.

2.2.2. Physical Operators

Simple Object Map (SOM) generates an RDF triple by performing a simple predicate object map statement. **Object Reference Map** (ORM) implements the object reference between two triples maps defined over the same data source. It extends SOM by using the subject of the parent triples map as the object of another TM. **Object Join Map** (OJM) implements an index join in executing a join condition between two TMs defined over two different data sources. OJM resorts to the corresponding PJTT to access the encoded values in the child map associated with the encoded values of the data source of the parent triples map. After executing a TM and generating triples, each operator will compare each triple to its corresponding PTT for duplicate removal. If they exist in the PTT, the triples are ignored. If they do not exist in the PTT, the triples are added to the PTT and the KG.

3. Empirical Evaluation

The Workshop on Knowledge Graph Construction hosted the KGCW 2023 Challenge; a series of test cases that covered a wide range of configurations that would affect the KG creation process like TMs with a large number of *predicateObjectMaps*, TMs with joins of varying levels of complexity, TMs with data sources with high duplicate rate, to name a few. From these test cases, we define the following research questions: **RQ1)** *What is the impact of the data duplicate rates in the execution time of SDM-RDFizer?* **RQ2)** *What is the impact of the input data size in the execution time of SDM-RDFizer?* **RQ3)** *How do the types of a triples maps affect the SDM-RDFizer?* **RQ4)** *How does the amount of triples maps affect the SDM-RDFIZER?* This section provides a comprehensive overview of the empirical study conducted in this work. It encompasses the benchmarks, metrics, engine, and the experimental environment utilized to evaluate the performance of the engines. Each experimental configuration is repeated five times, and the average time and memory usage are reported as the outcome. The results were analyzed to determine the strengths and weaknesses of SDM-RDFizer.

3.1. Experimental Configuration

Benchmark: The experiments were performed over the **KGCW 2023 Challenge**[3] dataset. The KGCW 2023 Challenge dataset was developed for the purpose of evaluating the performance (e.g., execution time and memory usage) of existing state-of-the-art KG creation engines. Additionally, this dataset seeks not only to determine the fastest creation pipeline but the most efficient pipeline based on the task, in other words, if the engine is the best suited for the configuration case. This dataset includes some test cases extracted from the GTFS-Madrid-Bench [7]. This benchmark presents data extracted from the Madrid Subway system with the purpose of evaluating state-of-the-art KG creation engines.
RML engine and Metrics: The engine used to perform the KGCW 2023 Challenge was SDM-RDFizer v4.7.1.5[4]. This version of SDM-RDFizer used implements all the data structures and physical operators defined in the previous section. Alongside SDM-RDFizer, a tool (called

[3] https://zenodo.org/record/7689310
[4] https://github.com/SDM-TIB/SDM-RDFizer/releases/tag/v4.7.1.5

Test Cases	Execution Time (sec)	Memory Usage (MB)
Duplicate 0%	76.77	679.24
Duplicate 25%	56.76	512.75
Duplicate 50%	38.86	312.85
Duplicate 75%	20.40	126.38
Duplicate 100%	2.41	79.60

Table 1
Duplicate Rate Test Cases

Test Cases	Execution Time (sec)	Memory Usage (MB)
Empty 0%	74.42	657.79
Empty 25%	65.87	566.82
Empty 50%	55.38	398.05
Empty 75%	44.41	246.63
Empty 100%	32.66	82.06

Table 2
Empty Values Test Cases

challenge-tool[5]) was provided for the challenge that will measure the execution time and memory usage. Therefore, the metrics used for the experimental study are *execution time* and *memory usage*. The execution time is reported in seconds (sec), and the memory usage is measured in megabytes (MB). For both execution time and memory usage, lower is better. The experiments are performed using in an Intel(R) Xeon(R) equipped with a CPU E5-1630 v4 @ 3.70GHz, 64GB memory, and with the O.S. Ubuntu 18.04LTS. The scripts used with challenge-tool, and the results generated from the challenge are available on GitHub [6].

3.2. Results and Analysis

Table 1 presents the results of the duplicate rate test cases. The percentage represents how much of the data source is defined as duplicates. It can be seen in the table that with higher duplicate rates, the execution time is shorter, and the memory usage is lower. This reduction in memory usage and execution time can be attributed to the fact that fewer unique triples will be generated with higher duplicate rates. Therefore, answering research question ***RQ1***. Additionally, the duplicate removal process is faster due to the PTT from SDM-RDFizer.
Table 2 presents the results of the empty values test cases. The percentage represents how much of the data source is defined by empty values. It can be seen in the table that the higher amount of empty values, the shorter the execution time and lower memory usage. Similar to the previous test case, a higher percentage of empty values suggests fewer unique triples are generated. The Empty 100% test case does not generate any triples.
Further, the test cases Duplicate 100% and Empty 100 % consume very little memory. This can be attributed to the fact that when only the table is given as the source of the TM, SDM-RDFizer generates based on the TM so that only necessary data is pulled from the database, thus avoiding the repetition of records. In the case of these test cases, each one only has one unique record. Thus, these test cases require very little to be transformed, leading to minimal memory usage.
Table 3 illustrates the results of the mapping test cases. The mapping test cases illustrate how the number of the TMs and the number of *predicateObjectMaps* impact the KG creation process. It can be observed in the table that the number of TMs impacts much more than the number of *predicateObjectMaps*. The higher number of TMs increases the execution time; this can be attributed to the fact that even though the data sources are smaller when there are more TMs, SDM-RDFizer has to go through each data source individually, requiring time. When there is

[5]https://github.com/kg-construct/challenge-tool
[6]https://github.com/SDM-TIB/SDM-RDFizer/tree/master/kgcw_2023_challenge

Test Cases	Execution Time (sec)	Memory Usage (MB)
1TM 15POM	66.13	485.97
3TM 5POM	104.59	251.96
5TM 3POM	147.69	229.87
15TM 1POM	362.10	439.08

Table 3
Mapping Test Cases

Test Cases	Execution Time (sec)	Memory Usage (MB)
1M rows 1 col	235.97	611.29
1M rows 10 col	539.65	2868.77
1M rows 20 col	774.69	5579.30
1M rows 30 col	1005.82	8587.83

Table 4
Properties Test Cases

Test Cases	Execution Time (sec)	Memory Usage (MB)
10k rows 20 col	9.18	134.67
100k rows 20 col	76.63	672.22
1M rows 20 col	742.15	5547.00
10M rows 20 col	13810.11	55034.55

Table 5
Records Test Cases

Test Cases	Execution Time (sec)	Memory Usage (MB)
Heterogeneity Files	2132.51	15374.90
Heterogeneity Mixed	2241.03	14542.68
Heterogeneity Nested	2210.66	15855.23
Heterogeneity Tabular	5060.82	10721.81
Scale 1	53.77	139.08
Scale 10	493.88	912.08
Scale 100	5138.59	10438.51
Scale 1000	No Result	No Result

Table 6
GTFS-Madrid-Benchmark Test Cases

only one TM, the data source is larger, but SDM-RDFizer only has to go through it once, and the KG will be generated at the end. Therefore, answering ***RQ4***.

Table 4 presents the results of the properties test cases. The properties test cases wish to present the impact of the number of *predicateObjectMaps* on the KG creation process. For that reason, the number of records in the data source was fixed to 1,000,000 rows. So, it can be seen in the table the higher number of properties, execution time, and memory usage are higher. This increase in memory usage and execution time can be attributed to the number of columns of the data source, which in turn increases the size of the data source and confirms what Chaves-Fraga et al. [6] says that larger data sources take more time to transform.

Table 5 presents the results of the records test cases; they seek to illustrate the impact of the number of rows in KG creation, so the number of columns is set to 20 for every test case. Similar to the properties test cases, these test cases reflect that with a higher number of rows, the space required to store them is more significant, and it needs more time to go through them. The results presented for the records and properties test cases illustrate the impact the size of the data sources have in the KG creation process, therefore answering research question ***RQ2***.

The results in Table 6 display the performance of test cases extracted from the GTFS-Madrid-Bench benchmark, which defines multiple sizes of data sources, denoted by the scale indicating the size of the outputted KG. For example, KG at Scale 10 is ten times larger than Scale 1. As expected, larger data sources lead to higher execution time and memory usage. Notably, the operation causing the most overhead is a self-join involving the largest table, "SHAPES." Among the cases with "Heterogeneity" in their names, which represent different combinations of data sources in various formats, all cases at Scale 100 show similar execution times and memory usage. This can be attributed to the use of XML and JSON files, which offer faster

Test Cases	Execution Time (sec)	Memory Usage (MB)
Join 1-1 0%	117.2	444.9
Join 1-1 25%	121.45	878.08
Join 1-1 50%	125.62	876.56
Join 1-1 75%	128.24	524.55
Join 1-1 100%	132.21	881.32

Table 7
Join 1-1 Test Cases

Test Cases	Execution Time (sec)	Memory Usage (MB)
Join 1-5 50%	120.03	380.46
Join 1-10 0%	114.21	419.3
Join 1-10 25%	122.4	431.3
Join 1-10 50%	125.83	445.54
Join 1-10 75%	130.51	879.32
Join 1-10 100%	134	329.97
Join 1-15 50%	126.74	870.63
Join 1-20 50%	121.48	865.32

Table 8
Join 1-N Test Cases

Test Cases	Execution Time (sec)	Memory Usage (MB)
Join 5-1 50%	118.86	448.63
Join 10-1 0%	112.24	865.68
Join 10-1 25%	115.92	874.44
Join 10-1 50%	119.47	858.12
Join 10-1 75%	121.11	527.62
Join 10-1 100%	123.79	551.01
Join 15-1 50%	118.91	456.84
Join 20-1 50%	118.05	865.24

Table 9
Join N-1 Test Cases

Test Cases	Execution Time (sec)	Memory Usage (MB)
Join 5-5 25%	123.96	461.22
Join 5-5 50%	132.44	511.55
Join 5-5 75%	140.64	856.43
Join 5-5 100%	146.57	596.07
Join 5-10 25%	121.67	494.74
Join 5-10 50%	130.27	506.34
Join 5-10 75%	139.15	844.96
Join 5-10 100%	144.8	592.04
Join 10-5 25%	123.07	497.95
Join 10-5 50%	131.93	505.68
Join 10-5 75%	140.65	858.46
Join 10-5 100%	146.48	623.54

Table 10
Join N-M Test Cases

traversal than CSV files and relational databases. Python's handling of CSV files as an array of dictionaries results in slower traversal. JSON files are uploaded as hash tables, and XML files are represented as trees, which are more efficient data structures. The "Heterogeneity Tabular" case, which exclusively uses CSV files and relational database tables, exhibits the highest execution time. This reinforces the notion that different data formats significantly impact performance. However, it is important to acknowledge the limitation that the largest scale, Scale 1000, was unable to be executed due to the characteristics of the environment in which the test cases were run. In conclusion, the analysis reveals that the choice of data format significantly affects the execution time and memory usage of the KG creation process, with XML and JSON files demonstrating better performance than CSV files and relational databases. Finally, Table 7, Table 8, Table 9, and Table 10 present the results of the test cases containing joins of different levels of complexity. It can be observed from all the tables that all test cases present very similar execution times. All test cases took around two minutes to be executed, with the highest execution time being 146.57. The similarity between the results can be linked to the usage of PJTTs. The PJTT helps SDM-RDFizer to avoid the need to upload the parent data source of a join multiple times, and by keeping the result of the join in main memory, the values can be extracted as needed, which in turn makes the execution of joins RML TMs much faster. The execution of joins requires a different approach and more resources to be transformed. That is why using PJTTs is necessary when the SDM-RDFizer executes joins. Thus, answering ***RQ3***.

4. Conclusions

The KGCW 2023 Challenge dataset aims to evaluate state-of-the-art engines and assess their strengths and weaknesses. In the case of SDM-RDFizer, it demonstrated relatively low execution time and memory usage in task-oriented test cases, such as duplicate removal, join execution, and avoiding the generation of empty values. This performance improvement in SDM-RDFizer can be attributed to the optimized data structures implemented in the engine, enhancing various aspects of the KG creation process. However, due to the lack of results from other engines for comparison, it is challenging to determine the significance of these findings definitively. Nevertheless, looking toward future versions of SDM-RDFizer, the authors plan to incorporate new techniques for handling data sources. These advancements aim to reduce memory usage further and improve overall execution time. SDM-RDFizer is poised to enhance its performance and continue its trajectory as a state-of-the-art solution by actively addressing these areas.

Acknowledgments

This work has been partially supported by the Federal Ministry for Economic Affairs and Energy of Germany (BMWK) in the project CoyPu (project number 01MK21007[A-L]). Leibniz Association partially funds Maria-Esther Vidal in the "Leibniz Best Minds: Programme for Women Professors", project TrustKG-Transforming Data in Trustable Insights with grant P99/2020.

References

[1] A. Dimou, T. De Nies, R. Verborgh, E. Mannens, R. Van de Walle, Automated Metadata Generation for Linked Data Generation and Publishing Workflows, in: Workshop on Linked Data on the Web, 2016.

[2] U. Şimşek, E. Kärle, D. Fensel, RocketRML - A NodeJS implementation of a use-case specific RML mapper, 2019. arXiv:1903.04969, accessed: 24-06-2022.

[3] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, M.-E. Vidal, SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs, in: CIKM, 2020. doi:10.1145/3340531.3412881.

[4] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, Semantic Web (2022). doi:10.3233/SW-223135.

[5] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Workshop on Linked Data on the Web, 2014.

[6] D. Chaves-Fraga, K. M. Endris, E. Iglesias, Ó. Corcho, M. Vidal, What Are the Parameters that Affect the Construction of a Knowledge Graph?, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2019, pp. 695–713.

[7] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus, O. Corcho, Gtfs-madrid-bench: A benchmark for virtual knowledge graph access in the transport domain, Journal of Web Semantics 65 (2020) 100596.