Semantic Web 15 (2024) 1–20
DOI 10.3233/SW-223135
IOS Press

Morph-KGC: Scalable knowledge graph materialization with mapping partitions

Julián Arenas-Guerrero [a,*], David Chaves-Fraga [a,b,c], Jhon Toledo [a], María S. Pérez [a] and Oscar Corcho [a]

[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*
E-mails: julian.arenas.guerrero@upm.es, david.chaves@upm.es, ja.toledo@upm.es, maria.s.perez@upm.es, oscar.corcho@upm.es

[b] *Declarative Languages and Artificial Intelligence Group, KU Leuven, Belgium*

[c] *Flanders Make, DTAI-FET, Belgium*

Editor: Elena Demidova, University of Bonn, Germany
Solicited reviews: Maria-Esther Vidal, Leibniz Information Centre For Science and Technology, Germany; Sherzod Hakimov, Leibniz Information Centre For Science and Technology, Germany; three anonymous reviewers

Abstract. Knowledge graphs are often constructed from heterogeneous data sources, using declarative rules that map them to a target ontology and materializing them into RDF. When these data sources are large, the materialization of the entire knowledge graph may be computationally expensive and not suitable for those cases where a rapid materialization is required. In this work, we propose an approach to overcome this limitation, based on the novel concept of *mapping partitions*. Mapping partitions are defined as groups of mapping rules that generate disjoint subsets of the knowledge graph. Each of these groups can be processed separately, reducing the total amount of memory and execution time required by the materialization process. We have included this optimization in our materialization engine Morph-KGC, and we have evaluated it over three different benchmarks. Our experimental results show that, compared with state-of-the-art techniques, the use of mapping partitions in Morph-KGC presents the following advantages: (i) it decreases significantly the time required for materialization, (ii) it reduces the maximum peak of memory used, and (iii) it scales to data sizes that other engines are not capable of processing currently.

Keywords: Knowledge Graphs, R2RML, RML, Scalability

1. Introduction

The amount of data that is being published in RDF has been steadily increasing in recent years. The generalized acceptance and use of knowledge graphs (KGs) [21] in a wide range of domains and organizations has contributed to this increase. Given that most of the data available inside organizations are structured in heterogeneous data formats, data integration techniques are often used in the data transformation and homogenization process required for knowledge graph construction (KGC).

KGC engines can be considered as data integration systems $DIS = \langle O, S, M \rangle$ where O is the global schema expressed in terms of an ontology (or network of ontologies), S is a set of input data sources and M are the mapping rules describing the relationships between O and S [30]. Mappings are usually expressed as declarative rules, using standard specifications such as the W3C Recommendation R2RML [9] and its well-known extension for data

[*] Corresponding author. E-mail: julian.arenas.guerrero@upm.es.

sources beyond relational databases (RDBs), RML [14]. The construction of KGs can be done using a materialization process or by virtualization [34]. Materialization (also known as semantic *extract-transform-load*) uses the rules in M to transform all data into RDF. Virtualization uses M to translate SPARQL queries into the native query language of S, i.e., data integration is performed *on-the-fly* during query processing [41].

There are many techniques and associated implementations that can be used to create knowledge graphs integrating heterogeneous data sources using declarative mapping rules [3,12,15,26,31,33,38,39]. In the specific case of materialization, different optimizations have been proposed to speed up the materialization process in complex data integration scenarios (e.g., high rate of duplicates, large data sources, or transformation functions). Approaches such as SDM-RDFizer [22], RMLStreamer [17] and FunMap [23] propose optimizations to enhance the performance of the materialization process. In our previous work [1], in which we analyzed several KGC engines, the experimental evaluation suggests that more efficient solutions are still needed, especially when the volume of data is large.

Problem and objectives We address the problem of scalability in knowledge graph construction from heterogeneous data sources using declarative mapping rules. Our main objective is to propose the theoretical background and a set of techniques that can enhance the process of KGC in complex data integration systems, increasing the performance in both time and memory consumption.

Proposed approach We present the novel concept of *mapping partitions*, which can be used to reduce the time required for the materialization of a knowledge graph and the peak amount of memory required in the process. Mapping partitions group rules in the input mapping documents ensuring the generation of disjoint sets of RDF triples by each of them. The experimental evaluation reveals that our proposal outperforms state-of-the-art engines significantly in terms of execution time and memory consumption, as well as our own implementation in the absence of this optimization.

Contributions (i) The novel concept of mapping partition, which allows the identification of rules that produce disjoint sets of RDF triples; (ii) algorithms to find a partition of a mapping document and to remove redundant self-joins within mapping documents; (iii) Morph-KGC, a scalable interpreter of R2RML and RML that implements mapping partition-based construction of the knowledge graph; (iv) an empirical evaluation of our approach and a comparison against four well-known KGC engines using three different benchmarks (GTFS-Madrid-Bench [6], SDM-Genomic-Datasets [22], and NPD [27]).

The remainder of the article is structured as follows. Section 2 introduces R2RML and RML, and presents a set of concepts, notations, and conventions that will be used throughout the rest of the paper. In Section 3 we delve into the foundations of mapping partitions. Section 4 presents the experimental evaluation comparing Morph-KGC with other R2RML and RML engines. Finally, Section 5 summarizes the related work and Section 6 wraps up and outlines future work.

2. Preliminaries

In this section, we first provide some background by introducing R2RML and RML, the mapping languages that this work focuses on. Then, we present some concepts, notations, and conventions that will be used in the following sections.

2.1. R2RML and RML

R2RML [9] is the W3C Recommendation declarative mapping language that links relational databases to the RDF data model. RML [14] is a well-known extension of R2RML that supports input data formats beyond RDBs (e.g., CSV, JSON, or XML). As RML is a superset of R2RML, in this section we present the main notions of these mapping languages focusing solely on RML.

An RML mapping is represented as an RDF graph. An *RML mapping document* is an RDF document that encodes an RML mapping, and it consists of one or more *triples maps*. A triples map has one *logical source* and contains the rules to generate the RDF triples. A triples map consists of one *subject map* and zero or more *predicate-object maps*. Each predicate-object map has in turn, one or more *predicate maps* and *object maps*. Subject, predicate, and object maps are *term maps* specifying how to generate the RDF terms in the homonymous positions of the triples. Term maps can be constant-valued (always generate the same value), reference-valued (the values are obtained directly from the logical source, e.g., a column in a table of an RDB), or template-valued (which generate RDF terms with some parts given by constants and others given by references). A template-valued term map comprises a *string template*, that defines how RDF terms are generated from one or more references (enclosed in curly braces) over the logical source. *Constant shortcut properties* are a compact method of expressing constant-valued term maps. A *referencing object map* allows to generate triples in which the object map is given by the subject map of another triples map, known as the *parent triples map*. A *join condition* is used when the logical sources of both triples maps are different. The evaluation of a triples map[1] produces an RDF graph whose triples consist of the generated RDF terms that result from applying its subject, predicate and object maps to the input logical source.

2.2. Assumptions, notation and conventions

In our work, we rely on the normalization of mapping rules as defined in [36]. A normalized mapping does not contain shortcuts, it uses predicate-object maps to type resources, and all its triples maps contain a single predicate-object map with one predicate map and one object map. This is not restrictive as any R2RML or RML document can be normalized [36].

We target KGC where the resulting RDF graph does not contain duplicated triples, as assumed by most engines [3,12,22]. Given that an RDF graph is a set of triples [8], the presence of duplicated triples in the serialization of the RDF graph does not affect the final result. We add this restriction as it has an impact on memory and time consumption, as well as on the size of the resulting files.

We use *[R2]RML* to refer to R2RML and RML. Our proposal may be easily extended to other R2RML-based mapping languages. We refer to R2RML columns and RML references indistinctly as *references*. We refer to RDF triples and quads indistinctly. *TM*, *SM*, *POM* and *OM* denote triples map, subject map, predicate-object map and object map respectively.

Let $\mathcal{T}$ be a term map, $\mathfrak{T}$ the set of all possible term maps, and $\mathbb{P}$ the set of positions that the RDF terms generated by the term map $\mathcal{T}$ can occupy in a quad, i.e., $\{subject, predicate, object, graph\}$. We define *position* as a function mapping $\mathfrak{T}$ to $\mathbb{P}$. Assume $\mathbb{T}$ to be the set of possible types of term maps, i.e., $\{IRI, Literal, BlankNode\}$, then *type* is a function mapping $\mathfrak{T}$ to $\mathbb{T}$. Assume $\mathbb{V}$ to be the set of possible values that a term map can have, i.e., $\{constant, reference, template\}$, then *value* is a function that maps $\mathfrak{T}$ to $\mathbb{V}$. Assume $\mathbb{C}$ to be the set of possible constant values that a constant-valued term map can take, then we define *const* as a function mapping $\mathfrak{T}'$ to $\mathbb{C}$ where $\mathfrak{T}' = \{\mathcal{T} \in \mathfrak{T} \mid value(\mathcal{T}) = constant\}$. Let $\mathbb{I}$ be the set of all possible values of the specified language tags or specified datatypes (as defined in [9]), then we define *literaltype* as a function mapping $\mathfrak{T}''$ to $\mathbb{I}$ where $\mathfrak{T}'' = \{\mathcal{T} \in \mathfrak{T} \mid type(\mathcal{T}) = Literal\}$.

3. The Morph-KGC approach

In this section, we introduce the foundations of Morph-KGC, an [R2]RML engine for constructing knowledge graphs at scale. First, we introduce self-join elimination at the mapping level. Next, we formalize the novel concept of mapping partitions. After that, we propose two algorithms to generate mapping partitions of [R2]RML documents and tackle KGC based on them. Finally, we validate the feasibility of this approach over different benchmarks and real use cases.

[1] https://www.w3.org/TR/r2rml/#generated-triples

3.1. Self-join elimination at the mapping level

Most virtualization engines in the state of the art (e.g., [3,35]) remove redundant self-joins in the SQL queries that they generate, with the objective of making query evaluation more efficient. However, most materialization engines do not address self-joins that can occur in a mapping, which in most cases are executed locally by the engine.

Definition 1 (*Redundant self-join* in an [R2]RML document)**.** A redundant self-join in an [R2]RML document appears when a referencing object map joins two triples maps with the same logical source and it has join conditions with the same unique references.

Redundant self-joins in an [R2]RML document can be removed by replacing the referencing object maps with object maps given by the subject map of the parent triples map, producing the same set of RDF triples.

Example 1. Consider the R2RML mapping rules with a self-join taken from GTFS-Madrid-Bench [6]:

```
<#shapesTM>
   rr:logicalTable [ rr:tableName "SHAPES" ];
   rr:subjectMap [
      rr:template "metro:shape/{shape_id}"
   ];
   rr:predicateObjectMap [
      rr:predicate gtfs:shapePoint;
      # referencing object map (self-join)
      rr:objectMap [
         rr:parentTriplesMap <#shapePoints> ;
         rr:joinCondition [
            rr:child "shape_id";
            rr:parent "shape_id";
         ];
      ];
      # object map (no join)
      rr:objectMap [
         rr:template "metro:shape_point/
            {shape_id}-{shape_pt_sequence}"
      ]
   ].
<#shapePointsTM>
   rr:logicalTable [ rr:tableName "SHAPES" ];
   rr:subjectMap [
      rr:template "metro:shape_point/
         {shape_id}-{shape_pt_sequence}"
   ].
```

Both triples maps use the same database table, and the referencing object map uses the same unique column to join both triples maps. This can be transformed into an object map without a join condition (the second object map in the triples map *#shapesTM*).

As defined in the R2RML Recommendation [9], the effective SQL query of the referencing object map in the triples map *#shapesTM* of *Example 1* is:

```
   SELECT * FROM
      ( SELECT * FROM SHAPES ) AS child,
      ( SELECT * FROM SHAPES ) AS parent
   WHERE child.shape_id=parent.shape_id
```

Removing this kind of SQL joins is widely studied in the literature, known as semantic query optimization [4,39]. Under the assumption that the join references in a mapping are unique, self-joins can be eliminated in mapping documents for any data format.

Algorithm 1: *Canonicalization* of an [R2]RML document, $\mathcal{M}$

```
   Result: Canonical $\mathcal{M}$
1  $\mathcal{M} = normalize(\mathcal{M})$ // see [36]
2  for $TM \in \mathcal{M}$ do
3      for $OM \in TM$ do
4          if $isRefOM(OM)$ then
5              $parentTM = OM.parentTM$
6              if $TM.source == parentTM.source$ then
7                  $removeJoin = True$
8                  for $joinCond \in OM$ do
9                      if $joinCond.child \neq joinCond.parent$ then
10                         $removeJoin = false$
11                 end
12                 if $removeJoin$ then
13                     $OM = parentTM.SM$
14     end
15 end
```

The impact of redundant self-joins in materialization engines has been previously reported by us in [1]. We propose to remove redundant self-joins within the mapping documents to improve the performance of KGC engines without the need to modify their current materialization procedures. In this way, an [R2]RML document without redundant self-joins does not contain referencing object maps involving two triples maps with the same logical source and with the same unique references in the join conditions. This redundant self-join elimination approach is independent from the underlying data format, as opposed to the previous techniques (e.g., [3,35] address RDBs only).

Definition 2 (*Canonical* [R2]RML document w.r.t. joins)**.** A canonical [R2]RML document is a normalized [R2]RML document without redundant self-joins.

An [R2]RML document and its canonicalization are equivalent, this entails that any transformation of a mapping document comprised in *Definition 2* (i.e., normalization and redundant self-join elimination) also generates an equivalent mapping document. Algorithm 1 obtains the canonicalization of any mapping document. First, it normalizes the document (see [36]). Next, it discards referencing object maps with different logical sources in the triples map and the parent triples map (*lines 4–6*). After that, the algorithm checks that the fields in all join conditions match (*lines 7–11*). When that happens, the self-join can be removed, and the object map is replaced by the subject map of the parent triples map (*lines 12–13*).

3.2. Mapping partitions

Given an initial set of mapping rules, we aim at identifying those that produce disjoint sets of triples, i.e., the initial mapping rules will be grouped so that those in different groups generate sets of RDF triples that do not overlap. In the following, when we refer to the sets of generated triples, we consider them to be composed of all the triples that a mapping rule, group of mappings rules, or mapping document generate given a data source.

Definition 3. (*Mapping Partition* of an [R2]RML document)**.** Let $\mathcal{M}$ be a normalized [R2]RML document with a set of mapping rules $m_1, m_2, \ldots, m_n$ that generates the triple set T. Then, a mapping partition $\mathcal{P}$ of $\mathcal{M}$ is a set of subsets of $\mathcal{M}$, designated as mapping groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$, that generate the triple sets $t_1, t_2, \ldots, t_k$, satisfying the following conditions:

- $\bigcap_{i=1}^{k} t_i = \emptyset$, i.e., the triple sets generated by each mapping group are disjoint.
- $\bigcup_{i=1}^{k} t_i = T$, i.e., the union of the triple sets generated by all the mapping groups is equivalent to T.

Multiple mapping partitions can exist for $\mathcal{M}$. The most trivial mapping partition is the one with only one mapping group (i.e., the singleton set), and we denote it with $\mathcal{P}_{\emptyset}$. Mind that this definition of mapping partition does not entail that a mapping group can be considered as a new mapping document. A mapping rule in a mapping group can still have a join condition involving a rule from a different group of mappings.

Example 2. Consider the mapping rules (taken from [9]):

```
<#TM1>
   rr:logicalTable [ rr:tableName "DEPT" ];
   rr:subjectMap [
      rr:template "ex:department/{DEPTNO}";
   ];
   rr:predicateObjectMap [
      rr:predicate ex:name;
      rr:objectMap [ rr:column "DNAME" ];
   ].
<#TM2>
   rr:logicalTable [ rr:tableName "EMP" ];
   rr:subjectMap [
      rr:template "ex:employee/{EMPNO}";
   ];
   rr:predicateObjectMap [
      rr:predicate ex:department;
      rr:objectMap [
         rr:parentTriplesMap <#TM1> ;
         rr:joinCondition [
            rr:child "DEPTNO";
            rr:parent "DEPTNO";
         ];
      ];
   ].
```

Both mapping rules can be assigned to different mapping groups as they do not generate common triples, given that the predicates are constants and that they are different. Nonetheless, the mapping rule in *#TM2* is dependent on *#TM1*, since the object map of the former is given by the subject map of the latter, and this results in a join dependency between those mapping groups. This prevents both mapping groups from becoming independent mapping documents.

Figure 1 depicts an example of mapping partitioning that involves three initial mapping documents with eleven normalized mapping rules in total. Six mapping groups are formed, which have between one and three mapping rules. As can be seen, there are join dependencies between different groups of mappings, nonetheless, they are still disjoint in terms of the set of triples that they generate.

We now delve into the rationale to obtain partitions of a mapping document $\mathcal{M}$. This is done incrementally by first examining the disjointness of term maps, next of mapping rules, and finally of mapping groups.

Definition 4 (*Prefix* of a template)**.** We define the prefix of a template as the constant (or immutable) part of its string template preceding the first reference in it. If a template starts with a reference, then its prefix is empty ($\emptyset$). Note that this is different to the notion of prefix declaration in RDF documents.

Example 3. Consider the following string templates along with their prefixes:

```
Template:  ex:employee={EMPNO}/department={DEPTNO}
Prefix:    ex:employee=
```

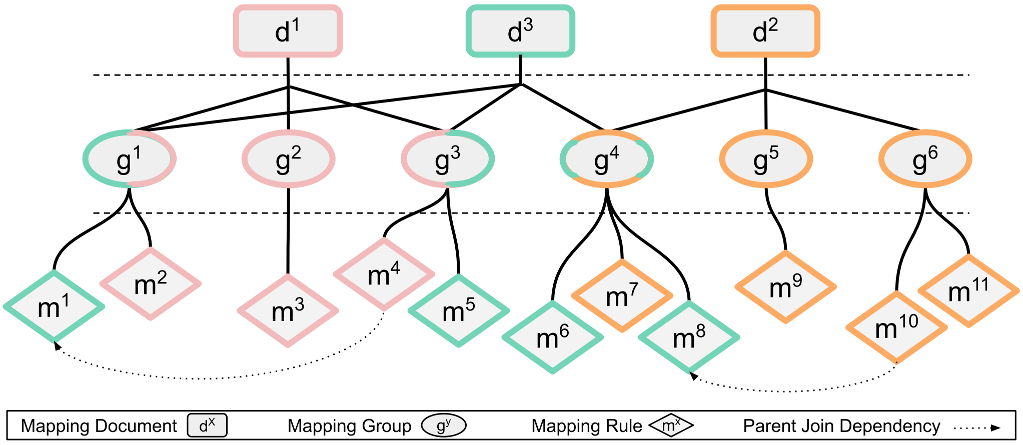


Fig. 1. **Mapping Partition.** Mapping partition of three mapping documents with eleven normalized mapping rules in total. The mapping partition is composed of six mapping groups, which have between one and three mapping rules. In addition, there are two join dependencies among different groups of mappings.

```
Template:  ex:roles/{ROLE}
Prefix:    ex:roles/

Template:  {EMPNO}-{DEPTNO}
Prefix:     ∅
```

The prefixes are obtained by eliminating the first reference and what follows. The beginning of the latter string template is data source-dependent and therefore its prefix is ∅.

Definition 5 (*Invariant* of an [R2]RML term map)**.** The invariant $\mathcal{I}$ of a term map $\mathcal{T}$ is the longest common initial part of all the RDF terms that can be generated by $\mathcal{T}$. $\mathcal{I}$ is an intrinsic property of $\mathcal{T}$ and remains immutable regardless of data coming from the input sources. $\mathcal{I}$ depends on $\mathbb{V}$, and it is obtained as follows:

- If $\mathit{value}(\mathcal{T}) = \mathit{constant}$, then $\mathcal{I} = \mathit{const}(\mathcal{T})$.
- If $\mathit{value}(\mathcal{T}) = \mathit{template}$, then $\mathcal{I} = \mathit{prefix}(\mathcal{T})$.
- If $\mathit{value}(\mathcal{T}) = \mathit{reference}$, then $\mathcal{I} = \emptyset$.

Notation 1 (*Invariants*)**.** For simplicity and clarity, we define the following notation for invariants:

- $\mathcal{I}_\emptyset$ denotes the empty invariant.
- $\mathcal{I}_1 < \mathcal{I}_2$, denotes that the length of $\mathcal{I}_1$ is shorter than $\mathcal{I}_2$, with the length given by the number of characters of the invariants.
- $\mathcal{I}_1 \subset \mathcal{I}_2$, denotes that $\mathcal{I}_1$ matches the beginning of $\mathcal{I}_2$. This entails $\mathcal{I}_1 < \mathcal{I}_2$.

Example 4. Consider the three templates in *Example 3* and their invariants $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ respectively (given by their prefixes). Then, the following applies:

- $\mathcal{I}_3 = \mathcal{I}_\emptyset$.
- $\mathcal{I}_3 < \mathcal{I}_2 < \mathcal{I}_1$.
- $\mathcal{I}_3 \subset \mathcal{I}_1, \mathcal{I}_3 \subset \mathcal{I}_2, \mathcal{I}_2 \not\subset \mathcal{I}_1$ and $\mathcal{I}_1 \not\subset \mathcal{I}_2$.

Definition 6 (*Disjoint Term Maps*)**.** Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two term maps and $\mathcal{I}_1$, $\mathcal{I}_2$ their respective invariants. $\mathcal{T}_1$ and $\mathcal{T}_2$ are disjoint iff the sets of RDF terms that they can generate are in turn disjoint, regardless of the input data. The disjoint property for $\mathcal{T}_1$ and $\mathcal{T}_2$ applies iff at least one of the following conditions hold:

1. $\mathit{type}(\mathcal{T}_1) \neq \mathit{type}(\mathcal{T}_2)$.
2. $\mathcal{I}_1 \neq \mathcal{I}_2, \mathcal{I}_1 \not\subset \mathcal{I}_2$ and $\mathcal{I}_2 \not\subset \mathcal{I}_1$.

3. $\mathcal{I}_1 < \mathcal{I}_2$, *type*$(\mathcal{T}_1) =$ *constant*, or *vice versa*.
4. *type*$(\mathcal{T}_1) =$ *type*$(\mathcal{T}_2) =$ *Literal*, and *literaltype*$(\mathcal{T}_1) \neq$ *literaltype*$(\mathcal{T}_2)$.

Disjointness of term maps depends on: $\mathbb{T}$, invariants, and $\mathbb{I}$. Term maps with different $\mathbb{T}$ enforces the generation of distinct (disjoint) RDF terms (*first condition*). We focus now on term maps with similar $\mathbb{T}$. For term maps with distinct invariants, the generated triple sets are disjoint if none of the invariants matches the beginning of the other (*second condition*). The latter is necessary to not make any assumptions on data coming from the sources. Note that in this case $\mathcal{I}_\emptyset$ prevents term map disjointness to apply. When the value of the term map with the shortest invariant is constant, the *second condition* can be relaxed, and it is only necessary that the invariant of this term is shorter than the other invariant (*third condition*). This is because the term map with the shortest invariant is not dependent on data and the RDF terms will always be shorter (and therefore distinct) than those generated by the other term map. For the specific case that two term maps generate literals, they are disjoint if they have distinct $\mathbb{I}$ (*fourth condition*). This is because RDF literals with different datatypes or language tags are different. This is also true for the empty literal type, e.g., a typed literal will always be different from a non-typed literal. Abusing of notation, given two term maps $\mathcal{T}_1$ and $\mathcal{T}_2$, we use $\mathcal{T}_1 \cap \mathcal{T}_2 = \emptyset$ to denote that they are disjoint.

Definition 7 (*Disjoint Mapping Rules*)**.** Two mapping rules m_1, m_2 that generate triple sets t_1, t_2 respectively, are disjoint iff they do not generate common triples, i.e., $t_1 \cap t_2 = \emptyset$, regardless of the input data sources. Disjointness of m_1 and m_2 can be determined as follows:

$$\exists \mathcal{T}_1 \in m_1, \exists \mathcal{T}_2 \in m_2 \mid \mathcal{T}_1 \cap \mathcal{T}_2 = \emptyset,$$
$$\mathit{position}(\mathcal{T}_1) = \mathit{position}(\mathcal{T}_2)$$

For two mapping rules to be disjoint, it is required that at least two position-wise term maps are disjoint. This is true because once two triples have a different subject, predicate, object or graph then the triples are immediately distinct. Abusing of notation, given two mapping rules m_1 and m_2, we use $m_1 \cap m_2 = \emptyset$ to denote that they are disjoint.

Definition 8 (*Disjoint Mapping Groups* of an [R2]RML document)**.** Two mapping groups $\mathcal{G}_1$ and $\mathcal{G}_2$ of $\mathcal{M}$ are disjoint iff they generate disjoint sets of triples. This property holds when all the mapping rules in $\mathcal{G}_1$ are disjoint of all the mapping rules in $\mathcal{G}_2$. As a consequence, a mapping rule cannot belong simultaneously to disjoint mapping groups. Formally:

$$\forall m_1 \in \mathcal{G}_1, \forall m_2 \in \mathcal{G}_2 \mid m_1 \cap m_2 = \emptyset$$

Definition 9 (*Maximal Mapping Partition* of an [R2]RML document)**.** The maximal mapping partition of $\mathcal{M}$ (denoted with $\mathcal{P}_{\max}$) is the one with the largest number of mapping groups.

Given a mapping document, its maximal mapping partition is not necessarily unique, i.e., there may be several maximal mapping partitions for the original mapping document. When all the mapping rules in a mapping document are disjoint, each mapping group in $\mathcal{P}_{\max}$ is a singleton set.

3.3. Mapping partition-based knowledge graph construction

Knowledge graph construction can leverage mapping partitions to reduce execution time and memory consumption. Before this, a partition of the mapping needs to be performed. We propose Algorithm 2, which generates partial mapping partitions by $\mathbb{P}$, and aggregates them for further partitioning. The purpose of this algorithm is to find a good partition (i.e., with a high number of mapping groups) while keeping it simple and with a low computational cost.

The outer for-loop (*line 3*) of Algorithm 2 iterates four times to retrieve partial mapping partitions by subject, predicate, object and graph. The normalized mappings are sorted lexicographically by the $\mathbb{T}$, $\mathbb{I}$ and invariants (*line 5*) in the position being processed. Term maps for each $\mathbb{P}$ are then iterated (*line 10*). The *first condition* in *Definition 6* is fulfilled with *lines 12–15*, that create a new $\mathcal{G}$ when a $\mathcal{T}$ with a different $\mathbb{T}$ is reached. The *fourth condition* in

Algorithm 2: *Partial-Aggregations Partitioning* of an [R2]RML document, $\mathcal{M}$

```
   Result: $\mathcal{P}$ of $\mathcal{M}$
1  $\mathcal{M} = normalize(\mathcal{M})$
2  // Repeat for subj., pred., obj., and graph
3  for $p \in \mathbb{P}$ do
4      // Lexicographic sort
5      $\mathcal{M} = sortByTermTypeAndLitTypeAndInv(\mathcal{M}, p)$
6      $curTermType = \emptyset$
7      $curLitType = \emptyset$
8      $curGroup = 0$
9      // Iterate over TMs in $\mathcal{M}$ with a certain $\mathbb{P}$
10     for $\mathcal{T} \in \mathcal{M}[p]$ do
11         $\mathcal{I} = invariant(\mathcal{T})$
12         if $type(\mathcal{T}) \neq curTermType$ then
13             $curTermType = type(\mathcal{T})$
14             $curInv = \emptyset$
15             $curGroup + +$
16         else if $type(\mathcal{T}) = Literal \wedge literaltype(\mathcal{T}) \neq curLitType$ then
17             $curLitType = literaltype(\mathcal{T})$
18             $curInv = \emptyset$
19             $curGroup + +$
20         else
21             if $allTMsConstants(\mathcal{M}, p) \wedge curInv < \mathcal{I}$ then
22                 $curGroup + +$
23             else if $curInv \not\subset \mathcal{I}$ then
24                 $curGroup + +$
25             $curInv = \mathcal{I}$
26         $\mathcal{T}.group = curGroup$
27     end
28 end
29 $\mathcal{P} = aggregatePartialPartitions(\mathcal{M})$
```

Definition 6 is satisfied with *lines 16–19*, which create a new a new $\mathcal{G}$ if $\mathbb{I}$ differs from that of the previous $\mathcal{T}$. If it was not possible to generate a new $\mathcal{G}$ with the former, we proceed to partition by invariant. When all the term maps in a specific $\mathbb{P}$ have constant values (*line 21*), the *third condition* in *Definition* 6 can be applied. Otherwise, it is checked if *condition 2*, which is more restrictive, is fulfilled (*line 23*). The final $\mathcal{P}$ is the result of aggregating the partial mapping partitions by $\mathbb{P}$ (*line 29*), e.g., a mapping rule with partial mapping partitions *subject(4)*, *predicate(23)*, *object(11)* and *graph(1)* would be assigned the final partition *4–23–11–1*.

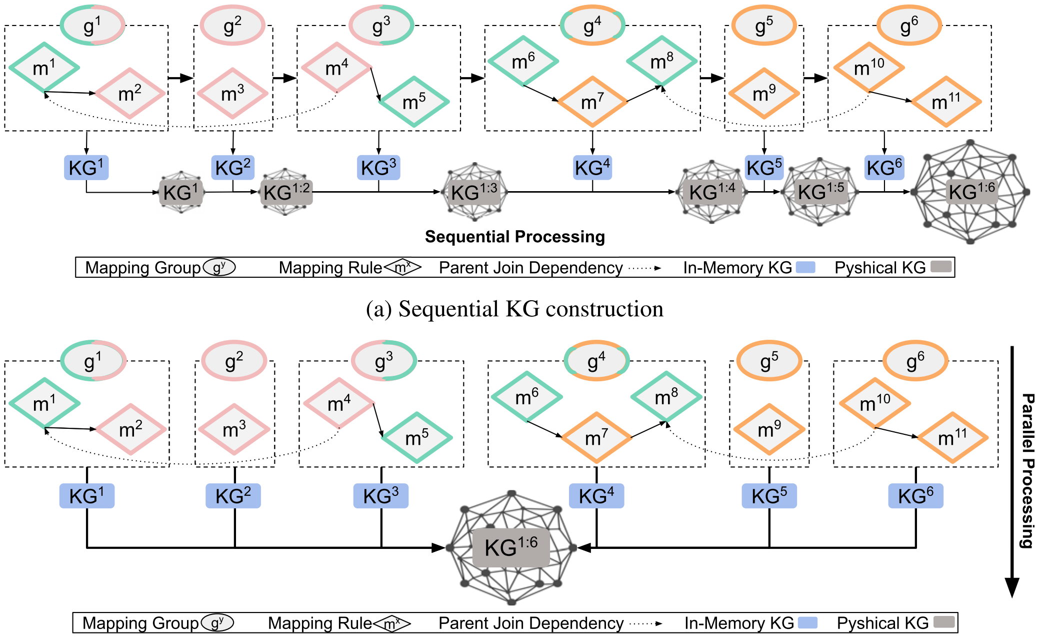
The time complexity of Algorithm 2 is $\mathcal{O}(n \log n)$, where n is the number of mapping rules. This is because the outer for-loop can be removed by repeating its inner code four times (once per each $\mathbb{P}$). The complexity is then determined by the lexicographic sorting of the mapping rules.

We also propose Algorithm 3 which generates the maximal mapping partition of an [R2]RML document, i.e., it solves the problem of finding $\mathcal{P}_{\max}$ of a mapping document. The assumption here is that exploring every possible mapping partition of an [R2]RML document guarantees obtaining the maximal one. To do so, it considers all orderings of $\mathbb{P}$ (*line 3*) and iterates over them (*line 4*). Partitioning is done independently by $\mathcal{G}$ (*lines 6–7*), thus the aggregation of the partial partitions is done before (*line 5*) to generate these groups. Once the full partition has been created for an order of $\mathbb{P}$ (*line 10*), it is checked whether it has more groups than any other previously created (*lines 11–12*). Finally, the partial mapping partitions are reset (*line 13*) to prepare them for the next order processing.

```
Algorithm 3: Maximal Partitioning of an [R2]RML document, $\mathcal{M}$
   Result: $\mathcal{P}_{\max}$ of $\mathcal{M}$
 1 $\mathcal{M} = normalize(\mathcal{M})$
 2 $\mathcal{P}_{\max} = \emptyset$
 3 for $order \in permutations(\mathbb{P})$ do
 4     for $p \in order$ do
 5         $\mathcal{P} = aggregatePartialPartitions(\mathcal{M})$
 6         for $\mathcal{G} \in \mathcal{P}$ do
 7             Apply lines 4–27 of Algorithm 2 to mapping rules in $\mathcal{G}$
 8         end
 9     end
10     $\mathcal{P} = aggregatePartialPartitions(\mathcal{M})$
11     if $size(\mathcal{P}) > size(\mathcal{P}_{\max})$ then
12         $\mathcal{P}_{\max} = \mathcal{P}$
13     $\mathcal{M} = resetPartition(\mathcal{M})$
14 end
```

The time complexity of Algorithm 3 is upper bounded by $\mathcal{O}(n \log n)$, where n is the number of mapping rules. The worst case for *lines 6–7* happens when the mapping partition has only one mapping group and they all need to be sorted (*line 5* of Algorithm 2). It must be noted that *lines 3–4* do not affect the time complexity since they are fixed by $\mathbb{P}$. Although the time complexity of Algorithms 2 and 3 are similar, the latter needs to perform more operations, since it considers every permutation of $\mathbb{P}$.

The construction of a knowledge graph based on a mapping partition can be done in two different ways. The first one (Fig. 2a) processes each mapping group sequentially. Hence, only the triples of a single mapping group are kept



(a) Sequential KG construction

(b) Parallel KG construction

Fig. 2. **Mapping partition-based KGC.** Example of sequential and parallel processing of a mapping partition for constructing a knowledge graph. While the former creates the KG by processing one mapping group at a time, reducing memory consumption, the latter generates triples for several mapping groups simultaneously, reducing execution time.

in memory simultaneously to remove duplicated triples. Memory usage is bounded by the largest group of mappings (in terms of the number of triples that it generates). The second one (Fig. 2b) processes each group of mappings in parallel. As a consequence, the execution time is reduced at the cost of increasing the maximum memory required, as multiple triple sets of different groups of mappings are maintained in memory at the same time.

3.4. Significance of mapping partitions

The performance of partition-based KGC strongly depends on the ability to partition mapping documents. If the conditions required to generate mapping partitions are not generally met, then mapping partitioning would not be feasible in practice (for instance, $\mathcal{I}_\emptyset$ prevents partitioning in the general case). In addition, the ability to generate a high number of mapping groups affects the improvement in the performance. In general, a higher number of mapping groups entails a higher parallelization capacity (bounded by the number of CPU cores), and a lower number of mapping rules in each of the groups, and therefore less memory consumption in the case of sequential processing.

We have compiled information on mapping partitioning for several well-known benchmarks (namely, NPD [27], BSBM [2], GTFS-Madrid-Bench [6] and LSLOD [18]), the DevOps ICT knowledge graph [7], and other real uses cases from the KGC W3C Community Group[2] in Table 1. We have included whether all the predicate maps are constant-valued, so that the *third condition* in *Definition 6* applies. We select predicate maps for this purpose because in real settings constant-valued term maps usually appear in this position (and in graph maps, but they are not used in the selected cases). The number of mapping groups and the maximum number of mapping rules in a group have been obtained using Algorithms 2 and 3. In all cases it has been possible to obtain a mapping partition beyond $\mathcal{P}_\emptyset$. In most of the cases the partitioning conditions are very advantageous, and low $\frac{\#mappingrules}{\#groups}$ ratios as well as small groups of mappings (with few mapping rules) are obtained. It can also be observed that both algorithms obtain a similar number of mapping groups in many cases. The most significant difference is found in the case of Data Hub – Ontopic, for which Algorithm 3 obtains a partition with a number of groups more than three times higher and drastically reduces the number of mapping rules in the largest group.

Table 1
Mapping partitioning of benchmarks and real use cases

Benchmark or Real Use Case	All pred. constants	# mapping rules	Partial-aggregations (Alg. 2)		Maximal (Alg. 3)	
			# $\mathcal{G}$	max # rules in $\mathcal{G}$	# $\mathcal{G}$	max # rules in $\mathcal{G}$
GTFS-Madrid-Bench	Yes	86	83	2	84	2
LSLOD – Bio2RDF	Yes	182	117	37	117	37
LSLOD – Linkedct	Yes	143	126	14	126	14
LSLOD – TCGA	Yes	2450	388	380	409	55
LSLOD – Dailymed	Yes	261	212	18	212	18
NPD	Yes	1177	477	116	745	14
BSBM	Yes	75	57	4	62	3
Open Cities – UPM	Yes	122	99	6	99	6
Btw Our Worlds – IDLab	Yes	62	47	4	47	4
SDM-Genomics – TIB	Yes	169	105	8	105	8
Drugs4Covid – UPM	Yes	75	27	29	38	19
Data Hub – Ontopic	Yes	270	69	83	232	6
DevOps ICT KG	Yes	326	299	3	299	3

[2] https://github.com/kg-construct/use-cases

4. Empirical evaluation

In this section, we experimentally evaluate our proposal. The research questions that we aim to answer are: **RQ1**: What is the impact of mapping partitions in the execution time and the memory consumption during the materialization of KGs? **RQ2**: How does the number of groups in a mapping partition affect the materialization process? **RQ3**: What are the benefits of this approach for constructing KGs at scale w.r.t. state-of-the-art techniques? In the following, we describe the setup of the evaluation.

Benchmarks We evaluate our proposal on three different testbeds. First, we use GTFS-Madrid-Bench [6], a benchmark in the transport domain, for testing the performance and scalability of our proposal over different tabular data formats and sizes. After that, we use the SDM-Genomic-Datasets [22] from the biomedical domain to evaluate our proposal over different mapping configurations. Finally, we use the Norwegian Petroleum Directorate (NPD) benchmark [27], from the energy domain, to compare different configurations of Morph-KGC. We have used MySQL 8.0 as DBMS for RDB. The NPD benchmark provides the mappings in R2RML, the SDM-Genomic-Datasets in RML, and GTFS-Madrid-Bench provides the mappings in both languages.

Engines We use Morph-KGC v1.1.0[3] and consider five configurations of it: (i) Morph-KGC as the baseline (without mapping partitioning); (ii) Morph-KGC^p, which uses partial-aggregations for mapping partitioning and sequential processing; (iii) Morph-KGC$^p_+$, which uses partial-aggregations for mapping partitioning and parallel processing, (iv) Morph-KGC^m which uses maximal partitioning and sequential processing; and (v) Morph-KGC$^m_+$ which uses maximal partitioning and parallel processing. We also compare our proposal against state-of-the-art KGC engines. Based on the results reported in [1], we select two R2RML engines, Ontop v4.1.0 and R2RML-F v1.2.3, and two RML interpreters, SDM-RDFizer v4.1.1 and Chimera v2.1. In our evaluation, we consider relational databases and CSV files, SDM-RDFizer and R2RML-F can process both of them, Ontop only processes the former and Chimera only the latter. It is important to mention that both selected RML processors parallelize the execution of the mappings.

Metrics *Execution time*: Elapsed time spent by an engine to complete the construction of a KG; it is measured as the absolute wall-clock system time as reported by the *time* command of the Linux operating system. *Memory consumption*: The memory used by an engine to construct the KG measured in time slots of 0.1 seconds. In addition, we have verified that the generated RDF are the same for all engines in terms of the number of triples and its correctness. All experiments were executed three times and the average execution time and memory consumption are reported. A timeout of 24 hours is used. The experiments are run on a CPU Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, 20 cores, 128 GB RAM, and a SSD SAS Read-Intensive 12 GB/s. The absolute values obtained for these metrics are available online[4] and we use a logarithmic scale for the figures in this section.

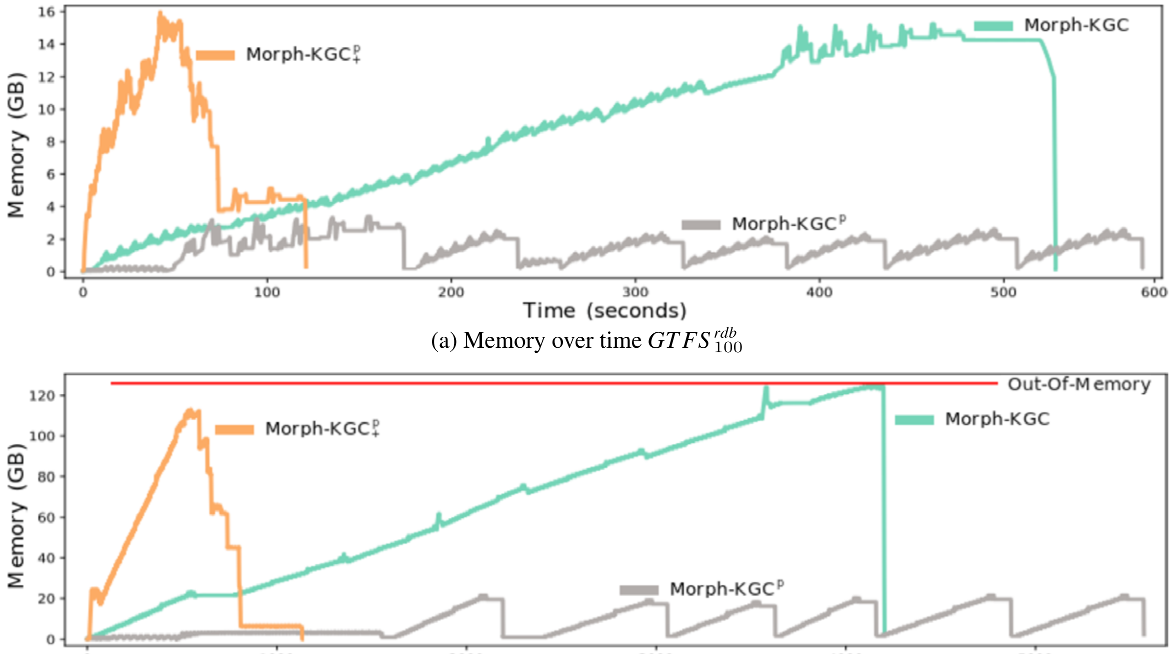
4.1. GTFS-Madrid-Bench

We consider two distributions of the GTFS-Madrid-Bench based on the data format: GTFS^{csv} and GTFS^{rdb}. We have also generated different data sizes of these distributions considering the scaling factors: 1, 10, 100, and 1000. As reported in Table 1, the partial-aggregations and maximal partitioning algorithms return very similar mapping partitions (differing only in one mapping group). Thus, in this experiment, we only take into account partial-aggregations for mapping partitioning, avoiding the extra computational cost of maximal partitioning. While the performance of Morph-KGC and Ontop are not impacted by self-joins because they remove them, the rest of the considered engines are extraordinarily affected by them. For this reason, we have manually removed self-joins from the original mappings. Mind that Ontop is only able to process GTFS^{rdb} and Chimera GTFS^{csv}.

The impact of mapping partitions on the materialization of large input data sources can be observed in Fig. 3. Regarding memory consumption, we can observe that the baseline, Morph-KGC, follows a growing trend over time. The reason is that it keeps the entire KG in memory to avoid the generation of duplicate triples. Indeed, Fig. 3b shows

[3] https://github.com/oeg-upm/morph-kgc

[4] https://doi.org/10.5281/zenodo.6542009



(a) Memory over time $GTFS^{rdb}_{100}$

(b) Memory over time $GTFS^{rdb}_{1000}$

Fig. 3. **Morph-KGC over GTFS-Madrid-Bench.** Memory over time in the materialization of $GTFS^{rdb}$ for three different configurations of the Morph-KGC engine: without mapping partitions (Morph-KGC), with mapping partitions and sequential processing (Morph-KGC^p), and with mapping partitions and parallel processing (Morph-KGC$^p_+$).

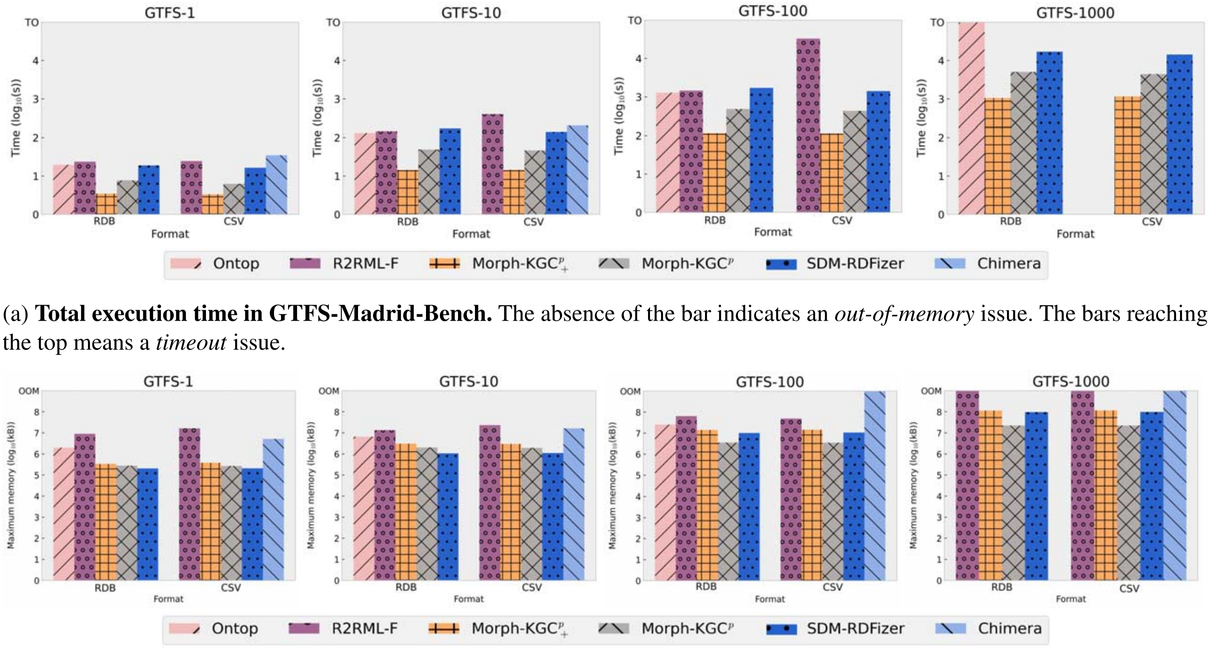
that this approach produces an *out-of-memory* issue due to the size of the final KG. In the case of Morph-KGC^p, it is observed how the memory is freed every time a group of mappings is materialized. In this configuration, the maximum peak of memory is given by the largest group of mapping rules (in terms of the total number of triples generated), and it is significantly lower than the other two configurations. However, this comes at the cost of a small overhead in the execution time w.r.t. the baseline. Morph-KGC$^p_+$ demonstrates a great improvement w.r.t. the baseline regarding execution time, although the maximum peak of memory used is similar due to Morph-KGC$^p_+$ maintaining multiple groups of mappings in memory at the same time, as they are being processed concurrently. Note that mapping partition-based KGC is bounded by the parallelization capacity of the processor and by the mapping partition itself (e.g., the number of mapping groups or the differences in size among them).

Figure 4a shows the total execution time of Morph-KGC compared to the rest of the selected engines. Morph-KGC$^p_+$ clearly outperforms the rest of the engines for all data formats and data scaling factors. The optimizations implemented by SDM-RDFizer make them the only competitor able to scale to GTFS$_{1000}$. Figure 4b depicts the maximum peak of memory used by each engine. It is observed that the operators compressing data in the data structures of SDM-RDFizer achieve the lowest memory usage for GTFS$_1$ and GTFS$_{10}$. In the case of larger distributions of GTFS, Morph-KGC^p outperforms SDM-RDFizer given that it reduces the maximum peak of memory used to that of the largest mapping group.

4.2. SDM-Genomic-Datasets

The SDM-Genomic-Datasets[5] provide a set of configurations taking into account different parameters that are relevant for constructing knowledge graphs [5] such as the type of mappings, the number of duplicates, and data size. More in detail, regarding the latter, four different datasets are provided with different number of rows: 10K,

[5] https://figshare.com/articles/dataset/SDM-Genomic-Datasets/14838342/1



(a) **Total execution time in GTFS-Madrid-Bench.** The absence of the bar indicates an *out-of-memory* issue. The bars reaching the top means a *timeout* issue.

(b) **Memory consumption peak in GTFS-Madrid-Bench.** The absence of the bar indicates a *timeout* issue. The bars reaching the top means an *out-of-memory* issue.

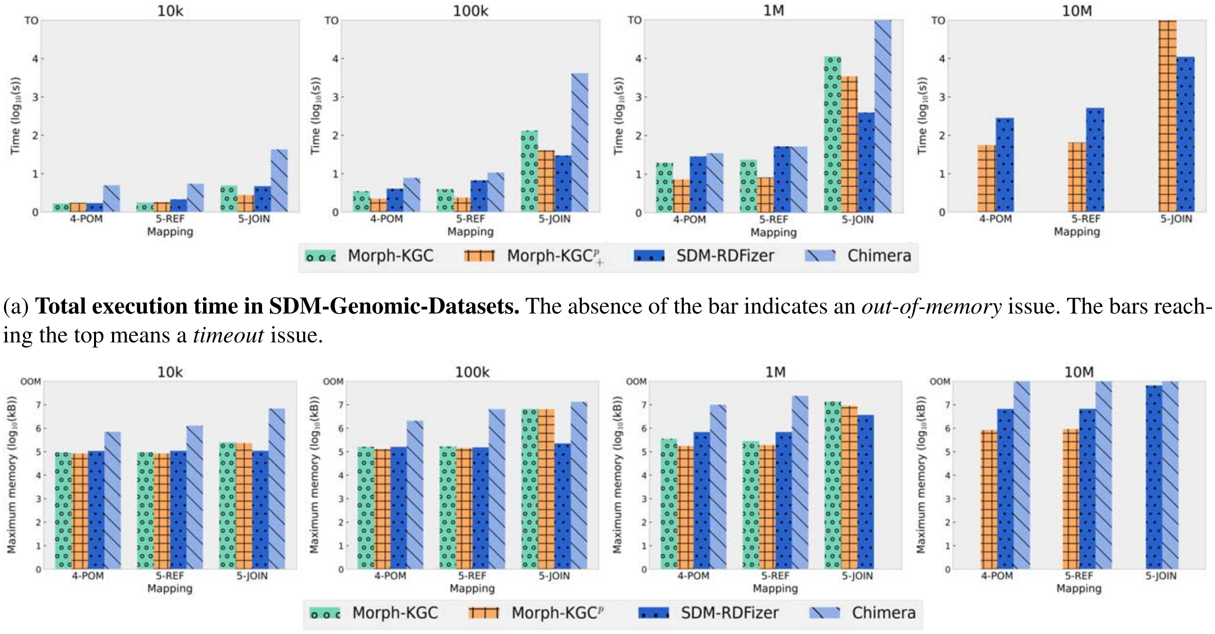
Fig. 4. **Total execution time and memory consumption peak in GTFS-Madrid-Bench.** KGC time in seconds and memory consumption time in kB (logarithmic scale) of the tabular datasets from GTFS-Madrid-Bench with data scaling factors 1, 10, 100 and 1000.

100K, 1M, and 10M. Although simpler configurations are also provided in terms of the number of duplicates, for this experimental evaluation we select the most complex one, i.e., 75% of duplicates with each duplicated value repeated 20 times. In addition, three mapping files with different types of predicate-object maps are considered: simple object map (POM), referencing object map with self-reference (REF), and referencing object map (JOIN). A number is used together with the name of each mapping type to specify the number of rules (e.g., 4-POM indicates 4 object maps). The testbeds provide the mappings in RML and data is in the form of CSV files. For this reason, we do not consider the R2RML processors in this experiment. In the same manner as for GTFS-Madrid-Bench, we only report the time and memory consumption for the case of partial-aggregations partitioning as the maximal partitioning algorithm generates the same mapping partition.

The total execution times for SDM-Genomic-Datasets are reported in Fig. 5a. As Morph-KGC$^p_+$ and Morph-KGC perform a self-join elimination over the REF mappings, they obtain similar results as in the POM ones, which is not the case of SDM-RDFizer and Chimera. While Morph-KGC$^p_+$ obtains the best results for the former configurations, SDM-RDFizer clearly outperforms it for JOIN mappings. The main reason is that SDM-RDFizer implements the Predicate Join Tuple Table as a specific physical data structure for improving the join conditions during the construction of the KG, and Morph-KGC does not implement any join optimization beyond redundant self-join elimination. Figure 5b shows that Morph-KGC^p outperforms its baseline and state-of-the-art engines regarding memory consumption for POM and REF mappings. In the case of JOIN mappings, SDM-RDFizer obtains the best results given the data compression techniques of its structures.

4.3. Norwegian Petroleum Directorate Benchmark

The NPD benchmark [27] presents a comprehensive evaluation system for virtual KGC engines. It provides a set of SPARQL queries, a scalable instance of an RDB from the energy domain, and the corresponding mapping rules in R2RML. Although it has not been previously used for testing the performance of materialization engines, we notice that it is the only benchmark among those considered in Table 1 in which the difference in the number of mapping



(a) **Total execution time in SDM-Genomic-Datasets.** The absence of the bar indicates an *out-of-memory* issue. The bars reaching the top means a *timeout* issue.

(b) **Memory consumption peak in SDM-Genomic-Datasets.** The absence of the bar indicates a *timeout* issue. The bars reaching the top means an *out-of-memory* issue.

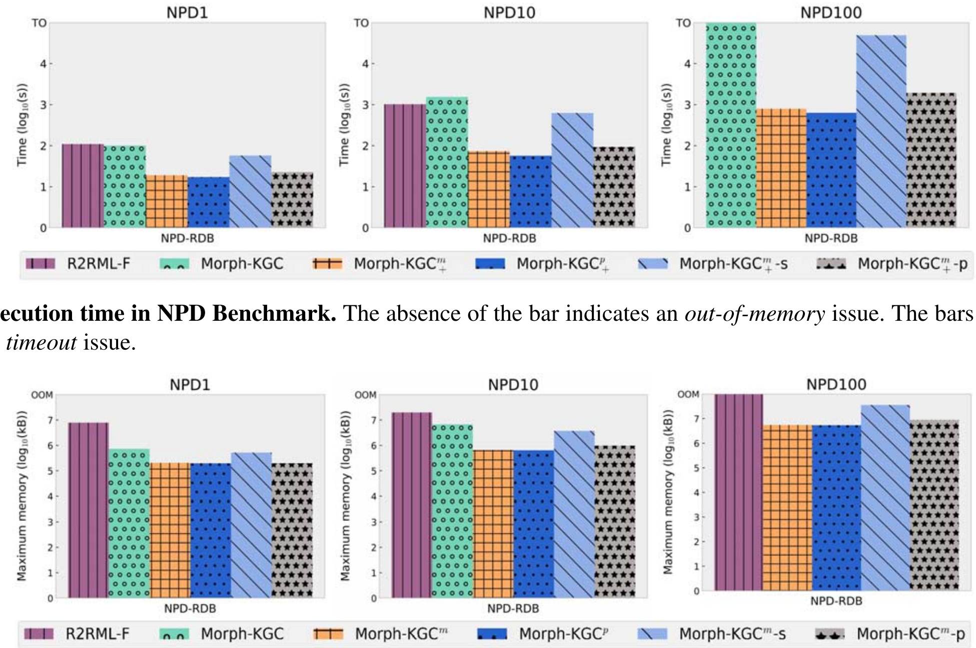
Fig. 5. **Total execution time and memory consumption peak in SDM-Genomic-Datasets.** KGC time in seconds and memory consumption time in kB (logarithmic scale) of the SDM-Genomic-Datasets with data scale factors 10K, 100K, 1M and 10M rows.

groups obtained by the partial-aggregations and the maximal partitioning algorithms is significant. This will allow us to compare the impact of maximal partitioning when it achieves a mapping partition with more groups than partial-aggregations. To further evaluate mapping partitions with different number of groups, we also include the configurations Morph-KGC$^m_+$-s and Morph-KGC$^m_+$-p which means that only subject and predicate, respectively, are taken into account to perform mapping partitioning (instead of every $\mathbb{P}$). We use the data generator of the benchmark [28] to obtain three different distributions with data scaling factors: 1, 10 and 100. Apart from our proposal, the only engine able to parse the R2RML mappings and generate the correct KG is R2RML-F.

The results obtained are shown in Fig. 6. We observe that the best performance regarding execution time is obtained by Morph-KGC$^p_+$. Surprisingly, the partition generated by Morph-KGC$^p_+$ reports better results than the Morph-KGC$^m_+$ one, showing that a higher number of mapping groups does not always entail a better execution time in the construction of the KG. A possible reason for this could be that the parallel processing is bounded by the number of cores of the machine (20 in our case), and increasing the number of mapping groups (477 for Morph-KGC^p while there are 745 for Morph-KGC^m) does not result in a higher parallelization rate. Moreover, a higher number of mapping groups introduces an overhead as we saw previously in Fig. 3, and maximal partitioning is computationally more expensive. However, we observe that the materialization time of Morph-KGC$^m_+$ is very close to Morph-KGC$^p_+$, and that these two perform significantly better than Morph-KGC^m-s and Morph-KGC^m-p, with a lower number of mapping groups (17 and 327 respectively). This indicates that in general, it is desirable to have a high number of mapping groups to increase the parallelization rate. Regarding memory consumption, we see that Morph-KGC^p and Morph-KGC^m obtain similar results. Note that in sequential processing, the peak in the amount of memory used is determined by the largest mapping group. If maximal partitioning is not able to further partition that specific mapping group, then a reduction in the peak of memory consumption is not expected.

4.4. Discussion

The experiments with SDM-Genomic-Datasets show that redundant self-join elimination is an effective technique and that it reduces the execution time of materialization. The same behaviour was also reported by us in [1]



(a) **Total execution time in NPD Benchmark.** The absence of the bar indicates an *out-of-memory* issue. The bars reaching the top means a *timeout* issue.

(b) **Memory consumption peak in NPD Benchmark.** The absence of the bar indicates a *timeout* issue. The bars reaching the top means an *out-of-memory* issue.

Fig. 6. **Total execution time and memory consumption peak in NPD.** KGC time in seconds and memory consumption time in kB (logarithmic scale) of the NPD benchmark with data scaling factors 1, 10, 100.

for GTFS-Madrid-Bench. Since our technique applies directly to mappings, it is engine independent and can also be applied to any data format, as opposed to other proposals such as [3], that only work for RDBs. Any engine can benefit from this technique by preprocessing the mapping rules using Algorithm 1. However, this only applies to redundant self-joins, and the experiments with SDM-Genomics-Datasets also show that Morph-KGC is outperformed by SDM-RDFizer in complex joins, given that it implements the Predicate Join Tuple Table. Morph-KGC could implement this data structure to speed up other kinds of joins. The impact of high join selectivity in KG materialization has been shown in [5].

In general, our empirical evaluation shows that mapping partitioning is an effective technique for knowledge graph materialization that avoids the generation of duplicate triples. Concurrent processing of mapping partitions has achieved the best execution times in many of the evaluation configurations, except for those involving complex joins as previously mentioned. We have seen that in scenarios with several types of mappings (POM+REF+JOIN) (e.g., GTFS-Madrid-Bench or NPD), Morph-KGC$^{p}_{+}$ obtains better results than the rest of the engines included in our evaluation. Moreover, sequential processing has obtained the lowest peak of memory used in most of the experiment configurations, reducing the amount of memory used to that of the largest mapping group. It has also been observed that sequential processing adds a small overhead in the execution time. Overall, parallel processing is suitable for those cases in which knowledge graph materialization needs to be rapidly executed, and sequential processing for those cases in which it is necessary to keep memory usage low. Mapping partitioning can be implemented by other engines, in particular, those that still report performance issues such as Chimera, could benefit from this technique.

We have seen in the experiment with NPD benchmark that the number of mapping groups in a partition has an impact on the performance of materialization. Regarding the execution time, a higher number of mapping groups entail a faster execution in parallel processing. However, once the maximum parallelization capacity of the machine is reached, a higher number of mapping groups does not reduce the execution time and can even slightly increase it. In respect to memory consumption, sequential processing bounds the maximum peak of memory to that of the largest mapping group. A higher number of groups only reduces this peak in the case that the largest mapping group has been further partitioned.

5. Related work

Ontology-based data integration [34,41] systems differentiate between the data layer, composed of the data sources, and the conceptual layer, in which an ontology or network of ontologies are used to abstract the heterogeneity of the former. Mappings are used to specify how to populate entities and relationships in the ontology with data from the underlying data sources, i.e., mappings are the link mechanism between both layers.

Several languages have been proposed to define mappings [16,19,29], but the one that stands out most notably is R2RML [9], the W3C Recommendation to map relational databases to RDF. However, R2RML does not generalize the underlying data model and cannot deal with heterogeneous data formats. RML [14] is a well-known superset of R2RML that removes specific references to the relational data model and enables data formats beyond RDBs. In addition, other R2RML-related proposals have addressed transformation functions [11,12], mixed content and RDF collections [33], usability [20] or scalability [40].

There are two approaches to process the mappings: virtualization (or query translation) and materialization (or data translation). Virtualization uses mappings to translate SPARQL queries into the native query languages of the underlying data sources. Research around this technique has focused primarily on relational databases and on efficiently processing the generated SQL queries. We refer the interested reader on virtualization to [3,15,35,39,42]. Materialization uses the mappings to transform all data in the underlying data sources to the corresponding RDF.

There are several solutions targeting the materialization of knowledge graphs [1]. For the specific case of RDBs, Ontop [3,43] leverages the fact that predicate maps are generally constant-valued. It generates one SPARQL query for each predicate with unbounded subject and object. These queries are then translated to SQL and optimized by applying a set of structural optimizations (e.g., subqueries elimination) and semantic query optimizations (e.g., redundant self-joins removal). It also avoids to retrieve large query result sets at once by doing it in chunks.

The work presented in [25] exploits knowledge encoded in the mapping documents to project the attributes appearing in a triples map, reducing the size of the data sources that need to be processed. Similarly, to diminish the impact of duplicates in the evaluation of join conditions, it also pushes down projections into joins. SDM-RDFizer [22] proposes physical data structures to store the knowledge graph in memory in a way that allows to efficiently remove duplicates and avoid unnecessary operations. In particular, it uses one hash table for each predicate (called *Predicate Tuple Table*, where the hash key combines the subject and the object of the triple, and the value is the triple itself. It also proposes to speed up joins by creating the *Predicate Join Tuple Table*, a hash table using the values matching the join condition as the hash key, and being the values of the hash the set of the generated values by the parent triples map. These hash tables are checked every time a new triple is to be generated, in the case that the triple already exists, it is discarded, otherwise it is added to the knowledge graph, and the corresponding hash table is updated. Moreover, SDM-RDFizer considers data compression techniques that reduce the memory usage of the data structures that store intermediate results during materialization.

Recently, triples map planning has been proposed in [24]. Here a bushy tree plan is created specifying an optimized execution order for executing mapping assertions. These bushy tree plans are obtained heuristically by relying on a greedy algorithm. Operating system commands are obtained from these execution plans that allow to efficiently execute different knowledge graph materialization engines.

Karma [26] tackles the scalability of knowledge graph construction from large data sources using batch processing. Instead of loading all data into memory, when operating in batch mode, the engine continuously loads fractions of the data, transforming them to the nested relational model and then materializing the corresponding triples. In this manner, memory usage is reduced, as it is not required to maintain the entire knowledge graph in memory.

Parallelization has also been proposed to speed up the materialization process. The work presented in [17] divides this process in three tasks following the *producer-consumer* paradigm: ingestion of data from the sources, mapping to RDF, and combination of the RDF. Parallelization is done up to the data record level. Nevertheless, the proposed approach does not tackle duplicate elimination. [22,37] also parallelize at the triples map level.

Regarding the efficient processing of transformation functions, FunMap [23] addresses the materialization of these functions specified using FnO [10]. FunMap proposes a set of lossless rewriting rules so that the functions are executed in the initial steps of the KG materialization process. This approach generates a set of mapping rules that are function-free and that can be executed by any RML engine.

Mapping partitioning can be used together with existing techniques to further optimize the process of knowledge graph materialization. For instance, the Predicate Join Tuple Table could be implemented along with mapping partitioning to speed up joins in parallel and sequential processing. Moreover, redundant self-join elimination could be implemented as a preprocessing step (similarly to other techniques such as the rewriting of rules in FunMap) which generates mappings without redundant self-joins that can be executed by any engine.

6. Conclusions and future work

We address the problem of scalability in the materialization of knowledge graphs from heterogeneous data sources using declarative mapping rules. We present the novel concept of mapping partitions, which consists in grouping mapping rules that generate disjoints sets of RDF triples. Mapping partitions can be used to reduce memory consumption in KG materialization by processing each mapping group in a partition sequentially, or to decrease the execution time by processing multiple mapping groups in parallel. We implement this novel approach in an [R2]RML engine, Morph-KGC, and we empirically demonstrate that, in the general case, it outperforms state-of-the-art proposals in terms of the total execution time and the amount of memory required in the materialization process.

Our future lines of work include the extension of Morph-KGC and mapping partitions to RML-star [13], which poses the challenge of recursive and more complex mapping rules. We also plan to address the limitation of the current approach that prevents minimizing materialization time and memory consumption at the same time, by using standards such as MPI [32].

Acknowledgements

This work was funded partially by the project Knowledge Spaces: Técnicas y herramientas para la gestión de grafos de conocimientos para dar soporte a espacios de datos (Grant PID2020-118274RB-I00, funded by MCIN/AEI/ 10.13039/501100011033) and also received partial financial support in the frame of the Euratom Research and Training Programme 2019–2020 under grant agreement No 900018 (ENTENTE project). David Chaves-Fraga is supported by the Spanish Minister of Universities (Ministerio de Universidades) and by the NextGenerationEU funds through the Margarita Salas postdoctoral fellowship.

References

[1] J. Arenas-Guerrero, M. Scrocca, A. Iglesias-Molina, J. Toledo, L. Pozo-Gilo, D. Doña, O. Corcho and D. Chaves-Fraga, Knowledge graph construction with R2RML and RML: An ETL system-based overview, in: *Proceedings of the 2nd International Workshop on Knowledge Graph Construction, CEUR Workshop Proceedings*, Vol. 2873, CEUR-WS.org, 2021.

[2] C. Bizer and A. Schultz, The Berlin SPARQL benchmark, *International Journal on Semantic Web and Information Systems, IJSWIS* **5**(2) (2009), 1–24. doi:10.4018/jswis.2009040101.

[3] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8**(3) (2017), 471–487. doi:10.3233/SW-160217.

[4] U.S. Chakravarthy, J. Grant and J. Minker, Logic-based approach to semantic query optimization, *ACM Transactions on Database Systems* **15**(2) (1990), 162–207. doi:10.1145/78922.78924.

[5] D. Chaves-Fraga, K.M. Endris, E. Iglesias, O. Corcho and M.-E. Vidal, What are the parameters that affect the construction of a knowledge graph? in: *Proceedings of the Confederated International Conferences*, Springer International Publishing, 2019, pp. 695–713. ISBN 978-3-030-33246-4. doi:10.1007/978-3-030-33246-4_43.

[6] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus and O. Corcho, GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain, *Journal of Web Semantics* **65** (2020), 100596. doi:10.1016/j.websem.2020.100596.

[7] O. Corcho, D. Chaves-Fraga, J. Toledo, J. Arenas-Guerrero, C. Badenes-Olmedo, M. Wang, H. Peng, N. Burrett, J. Mora and P. Zhang, A high-level ontology network for ICT infrastructures, in: *Proceedings of the 20th International Semantic Web Conference, ISWC*, Springer International Publishing, 2021, pp. 446–462. ISBN 978-3-030-88361-4. doi:10.1007/978-3-030-88361-4_26.

[8] R. Cyganiak, D. Wood and M. Lanthaler, *RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, W3C*, 2014, https://www.w3.org/TR/rdf11-concepts/.

[9] S. Das, S. Sundara and R. Cyganiak, *R2RML: RDB to RDF Mapping Language, W3C Recommendation, W3C*, 2012, http://www.w3.org/TR/r2rml/.

[10] B. De Meester, A. Dimou, R. Verborgh and E. Mannens, An ontology to semantically declare and describe functions, in: *Extended Semantic Web Conference, ESWC, P&D*, Springer International Publishing, 2016, pp. 46–49. ISBN 978-3-319-47602-5.

[11] B. De Meester, W. Maroy, A. Dimou, R. Verborgh and E. Mannens, Declarative data transformations for linked data generation: The case of DBpedia, in: *Proceedings of the 14th Extended Semantic Web Conference, ESWC*, Springer International Publishing, 2017, pp. 33–48. ISBN 978-3-319-58451-5. doi:10.1007/978-3-319-58451-5_3.

[12] C. Debruyne and D. O'Sullivan, R2RML-F: Towards sharing and executing domain logic in R2RML mappings, in: *Proceedings of the 9th Workshop on Linked Data on the Web, CEUR Workshop Proceedings*, Vol. 1593, CEUR-WS.org, 2016.

[13] T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, O. Corcho, D. Chaves-Fraga and A. Dimou, RML-star: A declarative mapping language for RDF-star generation, in: *International Semantic Web Conference, ISWC, P&D, CEUR Workshop Proceedings*, Vol. 2980, CEUR-WS.org, 2021.

[14] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens and R. Van de Walle, RML: A generic language for integrated RDF mappings of heterogeneous data, in: *Proceedings of the 7th Workshop on Linked Data on the Web, CEUR Workshop Proceedings*, Vols 1184, CEUR-WS.org, 2014, ISSN 1613-0073.

[15] K.M. Endris, P.D. Rohde, M.-E. Vidal and S. Auer, Ontario: Federated query processing against a semantic data lake, in: *Proceedings of the 30th International Conference on Database and Expert Systems Applications*, DEXA, Springer International Publishing, 2019, pp. 379–395. ISBN 978-3-030-27615-7. doi:10.1007/978-3-030-27615-7_29.

[16] H. García-González, I. Boneva, S. Staworko, J.E. Labra-Gayo and J.M.C. Lovelle, ShExML: Improving the usability of heterogeneous data mapping languages for first-time users, *PeerJ Computer Science* **6** (2020), e318. doi:10.7717/peerj-cs.318.

[17] G. Haesendonck, W. Maroy, P. Heyvaert, R. Verborgh and A. Dimou, Parallel RDF generation from heterogeneous big data, in: *Proceedings of the International Workshop on Semantic Big Data*, Association for Computing Machinery, 2019, pp. 1–6. ISBN 9781450367660. doi:10.1145/3323878.3325802.

[18] A. Hasnain, Q. Mehmood, S. Sana e Zainab, M. Saleem, C. Warren, D. Zehra, S. Decker and D. Rebholz-Schuhmann, BioFed: Federated query processing over life sciences linked open data, *Journal of Biomedical Semantics* **8** (2017), 13. ISBN 2041-1480. doi:10.1186/s13326-017-0118-0.

[19] M. Hert, G. Reif and H.C. Gall, A comparison of RDB-to-RDF mapping languages, in: *Proceedings of the 7th International Conference on Semantic Systems, I-Semantics '11*, Association for Computing Machinery, New York, NY, USA, 2011, pp. 25–32. ISBN 9781450306218. doi:10.1145/2063518.2063522.

[20] P. Heyvaert, B. De Meester, A. Dimou and R. Verborgh, Declarative rules for linked data generation at your fingertips! in: *Extended Semantic Web Conference, ESWC*, Springer International Publishing, 2018, pp. 213–217. ISBN 978-3-319-98192-5. doi:10.1007/978-3-319-98192-5_40.

[21] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G.D. Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, A.-C.N. Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab and A. Zimmermann, Knowledge graphs, *ACM Computing Surveys* **54**(4) (2021). doi:10.1145/3447772.

[22] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana and M.-E. Vidal, SDM-RDFizer: An RML interpreter for the efficient creation of RDF knowledge graphs, in: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, CIKM, Association for Computing Machinery, 2020, pp. 3039–3046. ISBN 9781450368599. doi:10.1145/3340531.3412881.

[23] S. Jozashoori, D. Chaves-Fraga, E. Iglesias, M.-E. Vidal and O. Corcho, FunMap: Efficient execution of functional mappings for knowledge graph creation, in: *Proceedings of the 19th International Semantic Web Conference, ISWC*, Springer International Publishing, 2020, pp. 276–293. ISBN 978-3-030-62419-4.

[24] S. Jozashoori, E. Iglesias and M.-E. Vidal, Scaling Up Knowledge Graph Creation to Large and Heterogeneous Data Sources, 2022.

[25] S. Jozashoori and M.-E. Vidal, MapSDI: A scaled-up semantic data integration framework for knowledge graph creation, in: *Proceedings of the Confederated International Conferences*, Springer International Publishing, 2019, pp. 58–75. ISBN 978-3-030-33246-4. doi:10.1007/978-3-030-33246-4_4.

[26] C.A. Knoblock and P. Szekely, Exploiting semantics for big data integration, *AI Magazine* **36**(1) (2015), 276–293. doi:10.1609/aimag.v36i1.2565.

[27] D. Lanti, M. Rezk, G. Xiao and D. Calvanese, The NPD benchmark: Reality check for OBDA systems, in: *Proceedings of the 18th International Conference on Extending Database Technology, EDBT*, OpenProceedings.org, 2015, pp. 617–628. doi:10.5441/002/edbt.2015.62.

[28] D. Lanti, G. Xiao and D. Calvanese, VIG: Data scaling for OBDA benchmarks, *Semantic Web* **10**(2) (2019), 413–433. doi:10.3233/SW-180336.

[29] M. Lefrançois, A. Zimmermann and N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: *Proceedings of the 14th Extended Semantic Web Conference, ESWC*, Springer International Publishing, 2017, pp. 35–50. ISBN 978-3-319-58068-5.

[30] M. Lenzerini, Data integration: A theoretical perspective, in: *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, Association for Computing Machinery, 2002, pp. 233–246. ISBN 1581135076. doi:10.1145/543613.543644.

[31] M.N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer and J. Lehmann, Squerall: Virtual ontology-based access to heterogeneous and large data sources, in: *Proceedings of the 18th International Semantic Web Conference, ISWC*, Springer International Publishing, 2019, pp. 229–245. ISBN 978-3-030-30796-7. doi:10.1007/978-3-030-30796-7_15.

[32] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard Version 4.0, 2021, https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf.

[33] F. Michel, L. Djimenou, C.F. Zucker and J. Montagnat, Translation of relational and non-relational databases into RDF with xR2RML, in: *Proceedings of the 11th International Conference on Web Information Systems and Technologies*, Vol. 1, SciTePress, 2015, pp. 443–454, ISSN 2184-3252. ISBN 978-989-758-106-9. doi:10.5220/0005448304430454.

[34] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, *Journal on Data Semantics X* (2008), 133–173. ISBN 978-3-540-77688-8. doi:10.1007/978-3-540-77688-8_5.

[35] F. Priyatna, O. Corcho and J. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: *Proceedings of the 23rd International Conference on World Wide Web, WWW '14*, Association for Computing Machinery, 2014, pp. 479–490. ISBN 9781450327442. doi:10.1145/2566486.2567981.

[36] M. Rodríguez-Muro and M. Rezk, Efficient SPARQL-to-SQL with R2RML mappings, *Journal of Web Semantics* **33** (2015), 141–169. doi:10.1016/j.websem.2015.03.001.

[37] M. Scrocca, M. Comerio, A. Carenini and I. Celino, Turning transport data to comply with EU standards while enabling a multimodal transport knowledge graph, in: *Proceedings of the 19th International Semantic Web Conference, ISWC*, Springer International Publishing, 2020, pp. 411–429. ISBN 978-3-030-62466-8. doi:10.1007/978-3-030-62466-8_26.

[38] J.F. Sequeda, M. Arenas and D.P. Miranker, OBDA: Query rewriting or materialization? In practice, both! in: *Proceedings of the 13th International Semantic Web Conference, ISWC*, Springer International Publishing, 2014, pp. 535–551. ISBN 978-3-319-11964-9. doi:10.1007/978-3-319-11964-9_34.

[39] J.F. Sequeda and D.P. Miranker, Ultrawrap: SPARQL execution on relational data, *Journal of Web Semantics* **22** (2013), 19–39. doi:10.1016/j.websem.2013.08.002.

[40] J. Slepicka, C. Yin, P. Szekely and C.A. Knoblock, KR2RML: An alternative interpretation of R2RML for heterogeneous sources, in: *Proceedings of the 6th International Workshop on Consuming Linked Data, CEUR Workshop Proceedings*, Vol. 1426, CEUR-WS.org, 2015.

[41] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati and M. Zakharyaschev, Ontology-based data access: A survey, in: *Proceedings of the 27th International Joint Conference on Intelligence, IJCAI, International Joint Conferences on Artificial Intelligence Organization*, 2018, pp. 5511–5519. doi:10.24963/ijcai.2018/777.

[42] G. Xiao, L. Ding, B. Cogrel and D. Calvanese, Virtual knowledge graphs: An overview of systems and use cases, *Data Intelligence* **1**(3) (2019), 201–223. doi:10.1162/dint_a_00011.

[43] G. Xiao, D. Lanti, R. Kontchakov, S. Komla-Ebri, E. Güzel-Kalaycı, L. Ding, J. Corman, B. Cogrel, D. Calvanese and E. Botoeva, The virtual knowledge graph system ontop, in: *Proceedings of the 19th International Semantic Web Conference, ISWC*, Springer International Publishing, 2020, pp. 259–277. ISBN 978-3-030-62466-8. doi:10.1007/978-3-030-62466-8_17.