ScheRe: Schema Reshaping for Enhancing Knowledge Graph Construction

Dongzhuoran Zhou*
dongzhuoran.zhou@de.bosch.com
Bosch Center for AI, Germany
University of Oslo, Norway

Baifan Zhou*
baifanz@ifi.uio.no
University of Oslo, Norway

Zhuoxun Zheng
zhuoxun.zheng@de.bosch.com
Bosch Center for AI, Germany
Oslo Metropolitan University, Norway

Ahmet Soylu
ahmet.soylu@oslomet.no
Oslo Metropolitan University, Norway

Ognjen Savkovic
ognjen.savkovic@unibz.it
Uni Bozen-Bolzano, Italy

Egor V. Kostylev
egork@ifi.uio.no
University of Oslo, Norway

Evgeny Kharlamov
evgeny.kharlamov@de.bosch.com
Bosch Center for AI, Germany
University of Oslo, Norway

ABSTRACT

Automatic knowledge graph (KG) construction is widely used for e.g. data integration, question answering and semantic search. There are many approaches of automatic KG construction. Among which, an important approach is to map the raw data to a given domain KG schema, e.g., domain ontology or conceptual graph, and construct the entities and properties according to the domain KG schema. However, the existing approaches to construct KGs are not always efficient enough and the resulting KGs are not sufficiently user-friendly. The main challenge arises from the trade-off: the domain KG schema should be knowledge-oriented, to reflect the general domain knowledge; while a KG schema should be data-oriented, to cover all data features. If the former is directly used for KG construction, this can cause issues like a high load of blank nodes, which are technical nodes in the KGs that represent unknown entities. To this end, we propose our *ScheRe* system in the demo, which relies on a schema reshaping algorithm and other two semantic modules for enhancing KG construction. The demo attendees will use *ScheRe* to reshape a domain KG schema to data specific KG schema, build KGs with industrial data, and experience more user-friendly querying.

CCS CONCEPTS

• **Information systems → Database utilities and tools**.

*Z. Zheng and B. Zhou contributed equally to this research as first authors.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA.

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/https://doi.org/10.1145/3511808.3557214

KEYWORDS

knowledge graph, schema reshaping, graph algorithm, automatic knowledge graph construction, ontology

ACM Reference Format:
Dongzhuoran Zhou, Baifan Zhou, Zhuoxun Zheng, Ahmet Soylu, Ognjen Savkovic, Egor V. Kostylev, and Evgeny Kharlamov. 2022. ScheRe: Schema Reshaping for Enhancing Knowledge Graph Construction. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/https://doi.org/10.1145/3511808.3557214

1 INTRODUCTION

Knowledge graphs (KG) allow to structure information in terms of nodes and edges [7]. Nodes represent entities of interests. Edges that connect entities and their data values, represent relations between entities and data properties of entities. In context of Industry 4.0 [10] and Internet of Things [8], KGs are gaining popularity in a wide range of industries [14, 16, 17, 26] and applications such as data integration [20, 25, 27], question answering [18], production monitoring in automatic manufacturing, pose estimation in robotics [1, 13] and semantic search [5, 24]. Due to complexity and variety of industrial data (e.g., relational tables), it is very desired to facilitate automation of KG construction. A common approach on KG construction is to construct entities and properties by relying on a given domain KG schema, a typical example being a domain ontology or conceptual graph, which is a graph-structured formal specification of shared knowledge [4, 15]. This approach matches raw data elements (table names and attributes names in relational tables) to nodes and edges in domain KG schema, then organise them in same pattern as domain KG schema [3, 9, 12, 28].

Consider an example scenario of KG construction that accounts for data integration and question answering in automated welding at Bosch – an impactful automated manufacturing process that is involved in production of over 50 millions of cars a year globally. Here we construct KGs by mapping raw data elements to a domain

KG schema (i.e. a domain ontology), which is designed after extensive discussion with users (domain experts) and encoding domain knowledge formally. Domain KG schema thus reflects general domain knowledge. KGs are automatically constructed by populating domain KG schema with raw data elements in relational tables and welding user manuals to support e.g., question answering, so that users can inspect and interact with data.

Challenge. Existing approaches to construct KGs are not always efficient enough and resulting KGs are not sufficiently application-oriented and user-friendly. The challenge arises from the trade-off between knowledge-orientation and data-orientation: data-specific schemata are *data-oriented* and tend to cover all raw data elements while they are hardly reusable across applications due to their limited domain scope; while domain KG schemata are typically *domain-generic* and *knowledge-oriented* they reflect domain knowledge using high level concepts rather than data particularities of given datasets [11]. Thus, KGs that result from such datasets often contain an overwhelming number of technical nodes, called blank nodes, that play the role of, e.g., labeled nulls in relational data exchange [2] and essentially represent entities that are not present in data but expected by KG schemata; note that up to 90% of information in such KGs can be blank nodes [6]. This makes KGs 'sparse' and hard to digest: applications based on such KGs e.g. query-based question answering should handle and skip hordes of blank nodes, which is unfriendly for users.

Our *ScheRe* System. To this end, we propose our *ScheRe* system that enhances KG construction with a schema reshaping algorithm to "reshape" a given domain KG schema to data-oriented KG schema [29, 30], which is free of blank nodes and still partially incorporates knowledge in domain schema. *ScheRe* has four layers that organise raw data, schema reshaping algorithm, and KG construction module for enhanced KG construction that generates KGs free of blank nodes. In addition, query reshaping module "reshapes" queries written on one KG schema to any KG schemata generated by schema reshaping algorithm, thus enabling uniform querying experience. resulting applications such as KG-based question answering and semantic search will enjoy KGs free of blanks nodes thus become more technologically friendly and user-friendly by saving space and time for storing, generating, and querying KGs.

Demo Overview. Demo attendees will experience a real industrial scenario of KG construction in automatic manufacturing with data provided by Bosch. They will perform a series of tasks via our GUI interface of *ScheRe*. Tasks include schema reshaping, KG construction and selecting queries. They will experience difference between comprehensive domain schema and reshaped schema, and see how user-friendly queries become on KG based on reshaped schema.

2 SCHERE: SCHEMATA RESHAPING SYSTEM

2.1 Architectural Overview

We now walk through the readers through our schema reshaping system (Figure 1). The system consists of four layers: *(Non-KG) Data Layer*, *Semantic Layer*, *KG Data Layer*, and *Application Layer*. From the very left, the *(Non-KG) Data Layer* contains the *Raw Data*. The *Raw Data* are in the form of relational tables and also have their corresponding *Relations*. The *Semantic Layer* contains several semantic artefacts and semantic modules. The *SchemaReshape* module takes the *domain KG schema* $\mathcal{G}^{do}$, the *Raw-to-DO Mapping* M^{do} (raw data to domain KG schema), and the *Relations* R (in addition, the user information U) as inputs, and generates a series of *Reshaped Schema* $\mathcal{G}^{ro}$ (*KG Schemata* at the same time) and their corresponding *Raw-to-RO Mappings* M^{ro}. These *KG Schemata* and *Raw-to-RO Mappings* are then used by the *KG Construction* module to construct the *KGs* from the *Raw Data*. And common *Queries* are selected by the users. The *KGs* in the *KG Data Layer* then can be used for applications like *Query-Based Analytics* [22] and *ML Analytics* [19, 21, 23, 26] in the *Application Layer*.

2.2 System Concepts and Artefacts

A *KG Schema* in our work is a directed labelled multigraph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, e.g., projected from a set of OWL 2 axioms (e.g., the domain KG schema $\mathcal{G}^{do}$ and reshaped schema $\mathcal{G}^{ro}$) as follows: The classes are projected to class nodes $\mathcal{N}^C$, the datatypes to datatype nodes $\mathcal{N}^D$, the object properties to object property edges $\mathcal{E}^O$, and the datatype properties to datatype property edges $\mathcal{E}^D$ (Fig. 1).

A *Relational Schema* $R = \{\mathsf{T}_1(\mathsf{A}), ..., \mathsf{T}_n(\mathsf{A})\}$ is a finite set of relational tables , where T_i is a table name while A is the attributes $\mathsf{A} = \{a_1, ..., a_k\}$ represented by their attribute names a_j. Among which, there exist attributes called the primary key A_p (each table only one) that uniquely identifies the rows, (optionally) foreign key attributes A^f refer to the primary keys of other tables, and normal attributes A^n that contain normal data.

A *Mapping (M)* is a bidirectional function that maps the elements in R to elements in $\mathcal{G}$. The Raw-to-DO Mapping (raw data to domain KG schema mapping) M^{do} maps the table names T in R to class nodes $\mathcal{N}^C$ in $\mathcal{G}^{do}$, normal attributes A^n to datatype property edges $\mathcal{E}^D$, and foreign keys A^f to object property edges $\mathcal{E}^O$, and vice versa. Similarly, the Raw-to-RO Mapping (raw data to reshaped schema mapping) M^{ro} maps the T, A^n, A^f to $\mathcal{N}^C$, $\mathcal{E}^D$, $\mathcal{E}^O$ in $\mathcal{G}^{ro}$. Here we assume the mapping M^{do} is one-to-one mapping that maps all elements in R to elements in $\mathcal{G}^{do}$[1]. Similarly, M^{ro} is also one-to-one.

$$M : \{\mathsf{T} \leftrightarrow \mathcal{N}^C, \mathsf{A}^n \leftrightarrow \mathcal{E}^D, \mathsf{A}^f \leftrightarrow \mathcal{E}^O | \mathsf{T}, \mathsf{A}^n, \mathsf{A}^f \in R, \mathcal{N}^C, \mathcal{E}^D, \mathcal{E}^O \in \mathcal{G}\}.$$

The *User Information (U)* can be understood as (1) a mandatory label that labels a node in $\mathcal{G}^{do}$ as the most important node for the users, named as the *main node*, n_m; (2) an extra set of mappings that map some normal attributes A^n in R to class nodes $\mathcal{N}^C$ in $\mathcal{G}^{do}$: $U : \{\mathsf{A}^n \leftrightarrow \mathcal{N}^C \mid \mathsf{A}^n \in R, \mathcal{N}^C \in \mathcal{G}^{do}\}$.

The *Dummy Nodes* $\mathcal{N}^{dummy}$ are nodes in KG schema $\mathcal{G}$ (and KG generated based on $\mathcal{G}$) that cannot be mapped to any elements in R.

The *Queries* in our system are SPARQL queries with the backbone as Basic Graph Pattern (BGP) query.

2.3 The Algorithm SchemaReshaping

Requirements and Performance. After reviewing literature and extensive discussion with users, we derive two mandatory requirements and three performance metrics:

[1] Note it is not the same case for the other way around: there normally exist many nodes or edges in $\mathcal{G}^{do}$ that cannot be mapped to any elements in R.

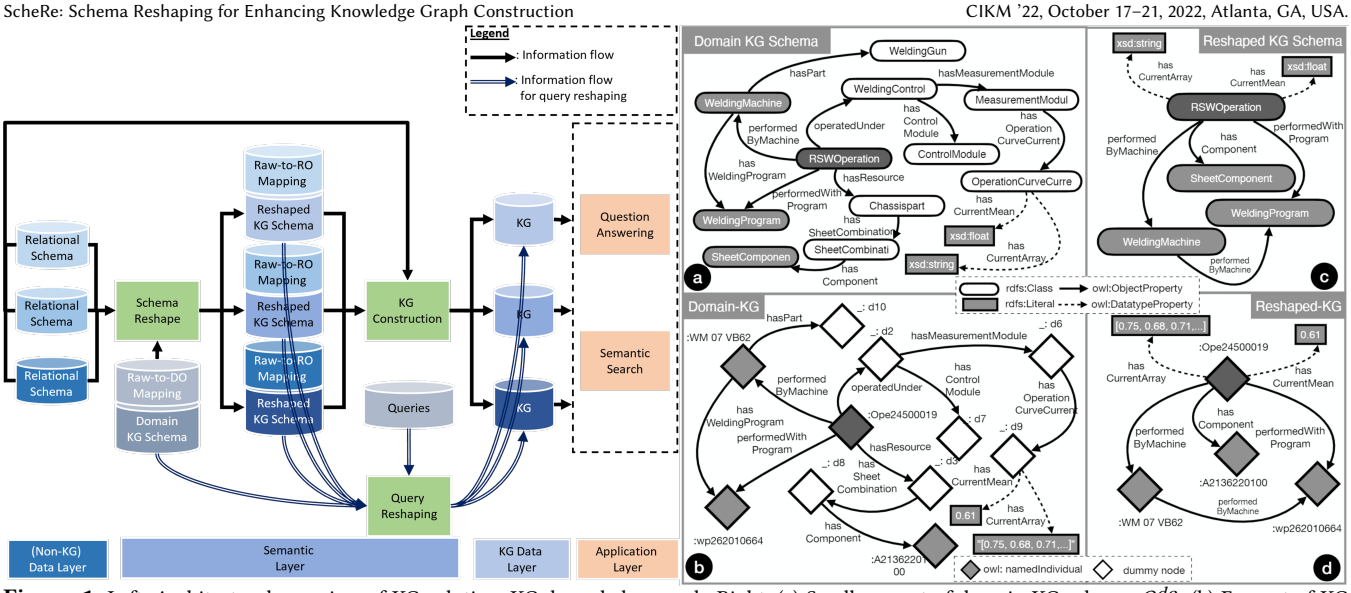

Figure 1: Left: Architectural overview of KG solution. KG: knowledge graph. Right: (a) Small excerpt of domain KG schema $\mathcal{G}^{do}$. (b) Excerpt of KG constructed by directly using domain KG schema, which has many dummy nodes due to nodes in $\mathcal{G}^{do}$ that are unmapped to raw data. (c) Reshaped KG schema $\mathcal{G}^{ro}$. (d) KG constructed based on $\mathcal{G}^{ro}$.

Algorithm 1: Schema Reshaping

Input: $\mathcal{G}^{do}$, R, M^{ro}, U
Output: $\mathcal{G}^{ro}$

$\mathcal{T}_1$, $\mathcal{E}_1^{deleted} \leftarrow$ Graph2Tree($\mathcal{G}^{do}$, U)
$\mathcal{T}_2$, $\mathcal{E}_2^{deleted}$, $M^{ro} \leftarrow$ TreeCollapse($\mathcal{T}_1$, R, M^{do}, U)
$\mathcal{G}^{ro} \leftarrow \mathcal{T}_2 \cup \{e(n_t, n_h) \mid e(n_t, n_h) \in \mathcal{E}_1^{deleted} \cup \mathcal{E}_2^{deleted}, n_t \in \mathcal{T}_2, n_h \in \mathcal{T}_2\}$

Algorithm 2: Graph2Tree

Input: $\mathcal{G}^{do}$, U
Output: $\mathcal{T}_1$, $\mathcal{E}_1^{deleted}$

Initialisation: $n_m \leftarrow$ *ReadUserInfo(U)*; $\mathcal{T}_1, \mathcal{N}^{leaf}, \mathcal{N}^{visited} \leftarrow \{n_m\}$; $\mathcal{E}_1^{deleted} \leftarrow \{\}$
while $\mathcal{N}^{leaf} \neq \emptyset$ **do**
  $\mathcal{N}^{ring} \leftarrow \{\}$
  foreach $n_i \in \mathcal{N}^{leaf}$ **do**
    $\mathcal{N}_i^{ring} \leftarrow \{\}$
    foreach $e^u(n_i, n_j) \in \mathcal{G}^{do} \setminus \mathcal{E}_1^{deleted}, n_j \in \mathcal{G}^{do}$ **do**
      if $n_j \notin \mathcal{N}^{visited}$ **then**
        $\mathcal{T}_1 := \mathcal{T}_1 \cup \{n_j, e^u(n_i, n_j)\}$
        $\mathcal{N}^{visited} := \mathcal{N}^{visited} \cup \{n_j\}$
        $\mathcal{N}_i^{ring} := \mathcal{N}_i^{ring} \cup \{n_j\}$
      else
        $\mathcal{E}_1^{deleted} := \mathcal{E}_1^{deleted} \cup \{e^u(n_i, n_j)\}$
    $\mathcal{N}^{ring} := \mathcal{N}^{ring} \cup \mathcal{N}_i^{ring}$
  $\mathcal{N}^{leaf} \leftarrow \mathcal{N}^{ring}$

Algorithm 3: TreeCollapse

Input: $\mathcal{T}_1$, R, M^{do}, U
Output: $\mathcal{T}_2$, $\mathcal{E}_2^{deleted}$, M^{ro}

Initialisation: $\mathcal{N}^{ring} \leftarrow \{n_m\}$, $\mathcal{T}_2 \leftarrow \mathcal{T}_1$, $\mathcal{E}_2^{deleted} \leftarrow \{\}$
$\mathcal{N}^{selected} \leftarrow$ GetNodes(R, M^{do}) $\cup$ ReadUserInfo(U) $\cup$ IdentifyID($\mathcal{G}^{rs}$)
while $\mathcal{N}^{ring} \neq \emptyset$ **do**
  $\mathcal{N}_{next}^{ring} \leftarrow \{\}$
  foreach $n_i \in \mathcal{N}^{ring}$ **do**
    foreach $e^u(n_i, n_j) \in \mathcal{T}_1$ **do**
      if $n_j \notin \mathcal{N}^{selected}$ **then**
        $\mathcal{T}_2 := \mathcal{T}_2 \setminus \{n_j, e^u(n_i, n_j)\}$
        $\mathcal{E}_2^{deleted} := \mathcal{E}_2^{deleted} \cup \{e^u(n_i, n_j)\}$
        if $e^u(n_j, n_k) \in \mathcal{T}_1$ **then**
          $\mathcal{T}_2 := \mathcal{T}_2 \setminus \{e^u(n_j, n_k)\}$
          $\mathcal{T}_2 := \mathcal{T}_2 \cup \{e^u(n_i, n_k)\}$, where $e^u(n_i, n_k)$ adopts the label of $e^u(n_j, n_k)$
          $\mathcal{E}^{deleted} := \mathcal{E}^{deleted} \cup \{e^u(n_j, n_k)\}$
          $\mathcal{N}_{next}^{ring} := \mathcal{N}_{next}^{ring} \cup \{n_i\}$
      else
        $\mathcal{N}_{next}^{ring} := \mathcal{N}_{next}^{ring} \cup \{n_j\}$
  $\mathcal{N}^{visited} := \mathcal{N}^{visited} \cup \mathcal{N}^{ring}$
  $\mathcal{N}^{ring} \leftarrow \mathcal{N}_{next}^{ring}$
$M^{ro} \leftarrow MappingGeneration(T2, M^{do})$

- *R1 Data Coverage*, the generated KG schema $\mathcal{G}^{ro}$ should cover all the table names and attribute names in R.
- *R2 Succinctness*, $\mathcal{G}^{ro}$ should possible have zero dummy nodes.
- *P1 Knowledge Coverage*, $\mathcal{G}^{ro}$ should possibly preserve the knowledge encoded in the domain KG schema, measured by the number of elements in $\mathcal{G}^{do}$ kept in $\mathcal{G}^{ro}$.
- *P2 Simplicity*, $\mathcal{G}^{do}$ should have possibly shallow structure, determined by the graph diameter d of $\mathcal{G}^{ro}$.
- *P3 Efficiency*, the KG generation based on $\mathcal{G}^{do}$ should be efficient in time and storage space.

Problem Formulation. We formulate the problem of *Schema Reshaping* as a problem of computing from a given KG schema and some context, a new schema that *fully* satisfies the requirement R1-R2 and achieves possibly good performance in terms of P1-P3. In this work, based on the system concepts and artefacts, we formulate:

$$\text{Schema Reshaping} : (\mathcal{G}^{do}, R, M^{do}, U) \rightarrow \mathcal{G}^{ro}, M^{ro}$$

where $\mathcal{G}^{do}$ is a given domain KG schema, R is a set of relational tables, M^{do} is a mapping between R and $\mathcal{G}^{do}$, U is optional user information, and $\mathcal{G}^{ro}$ is the "reshaped" schema, M^{ro} is a mapping between $\mathcal{G}^{ro}$ and R.

Algorithm. Intuitively, the algorithm Schema Reshaping selects subsets of nodes and edges from a given domain KG schema $\mathcal{G}^{do}$, which can be mapped to a relational schema R or included in the user information U, and then connect the selected subsets with possibly more edges in $\mathcal{G}^{do}$, thus generating the reshaped schema $\mathcal{G}^{ro}$. More specifically, Schema Reshaping does so in three steps:

- *Step 1*, it transforms $\mathcal{G}^{do}$ to a tree $\mathcal{T}_1$ by removing some edges, where the tree has the *main node* n_m given in U as the root (Alg. 2 Graph2Tree);
- *Step 2*, it selects the subsets of nodes and edges of $\mathcal{T}_1$ that are mapped in R by M^{do} or pointed by the users, creating a $\mathcal{T}_2$ (Alg. 2 TreeCollapse);

- *Step 3*, some deleted edges in Step1 and Step 2 are added back to $\mathcal{T}_2$, where these edges have both their head and tail in $\mathcal{T}_2$, resulting $\mathcal{G}^{ro}$ (Line 3 in Alg. 1).

Step 1. Graph2Tree. With n_m as the root node, Step 1 (Alg. 1) expands the tree $\mathcal{T}_1$ with nodes and edges selected from $\mathcal{G}^{do}$ layer by layer, in a way that there exists only one path between any node and n_m. We first clarify several concepts used in the step: $\mathcal{N}^{leaf}$ refer to the set of leaf nodes of $\mathcal{T}_1$, $\mathcal{N}^{ring}$ refers to the set of "ring nodes" (nodes in a outer layer of the leaf nodes) that are potential to be added to $\mathcal{T}_1$, $\mathcal{N}^{visited}$ is the set of visited nodes, and $\mathcal{E}_1^{deleted}$ is a set of the deleted edges. Then we introduce the procedure. First, Alg. 2 reads the user information to mark the main node n_m, and initialise $\mathcal{T}_1$, $\mathcal{N}^{leaf}$, $\mathcal{N}^{visited}$ with n_m, and the set $\mathcal{E}_1^{deleted}$ with the empty set (Line 1). Next, if $\mathcal{N}^{leaf}$ is not empty, Alg. 2 does the following steps: it initialises an empty set $\mathcal{N}^{ring}$ (Line3), then it enumerates each node n_i in the current $\mathcal{N}^{leaf}$ (Line 4) and create an empty set of ring nodes $\mathcal{N}_i^{ring}$ that belong to n_i. For each leaf node n_i, it enumerates the edges incidental to the node n_i in $\mathcal{G}^{do}$, $e^u(n_i, n_j)$ but not in $\mathcal{E}_1^{deleted}$, and exams the other node n_j that this edge is connected to. If n_j is not visited (not in $\mathcal{N}^{visited}$), then the node n_j and the edge $e^u(n_i, n_j)$ are added to $\mathcal{T}_1$ (Line 8), n_j is added to $\mathcal{N}^{visited}$ and a new ring set $\mathcal{N}_i^{ring}$ that belongs to n_i (Line 9-10), and . If n_j is already visited, the edge $e^u(n_i, n_j)$ is added to $\mathcal{E}_1^{deleted}$ (Line 12). After all $e^u(n_i, n_j)$ are enumerated, all elements in $\mathcal{N}_i^{ring}$ are added to $\mathcal{N}^{ring}$ (Line 13). After all n_i are numerated, the $\mathcal{N}^{ring}$ becomes the new $\mathcal{N}^{leaf}$ (Line 14).

Step 2. Tree collapse. Step 2 (in Alg. 3) selects nodes in $\mathcal{G}^{rs}$, by user or rule, and save them in $\mathcal{N}^{selected}$, then deletes the nodes not in $\mathcal{N}^{selected}$ from $\mathcal{T}_2$ which is copied by $\mathcal{T}_1$, at the same time keeps the connectivity of $\mathcal{T}_2$. It takes 4 inputs: the tree $\mathcal{T}_1$, the relational schema R, raw data to domain ontology mapping M^{do}, and user information U. The algorithm firstly initialised the ring node set $\mathcal{N}^{ring}$ with main node n_m, $\mathcal{T}_2$ with $\mathcal{T}_1$, and deleted edge set $\mathcal{E}_2^{deleted}$ for$\mathcal{T}_2$ with empty set (Line 1). Then the algorithm selects the nodes in relational schema graph $\mathcal{G}^{rs}$, or with datatype property having "ID" or "Name", or by user choices. These nodes are added into $\mathcal{N}^{selected}$ (Line 2). If $\mathcal{N}^{ring}$ is not empty, the Alg. 3 does the following steps: it initialise an empty set $\mathcal{N}_{next}^{ring}$, then it enumerate each node n_i in the current $\mathcal{N}^{leaf}$ (Line 4). For each leaf node n_i, it enumerates the edges incidental to the node n_i in $\mathcal{T}_1$, $e^u(n_i, n_j)$. If n_j is not selected (not in $\mathcal{N}^{selected}$), then the node n_j and edge $e^u(n_i, n_j)$ are deleted from $\mathcal{T}_2$ and $e^u(n_i, n_j)$ is added to $\mathcal{E}_2^{deleted}$. If the edge $e^u(n_j, n_k)$ is in $\mathcal{T}_1$,then $e^u(n_j, n_k)$ is deleted from $\mathcal{T}_2$, and a new edge $e^u(n_i, n_j)$ with same label of $e^u(n_j, n_k)$ is added to $\mathcal{T}_2$. The $e^u(n_j, n_k)$ is added to $\mathcal{N}^{selected}$ and n_i is added to $\mathcal{N}_{next}^{ring}$. If n_j is in $\mathcal{N}^{selected}$, the n_j is added to $\mathcal{N}_{next}^{ring}$. After all n_i are enumerated, The $\mathcal{N}^{ring}$ is added to $\mathcal{N}_{visited}$, $\mathcal{N}_{next}^{ring}$ becomes the new $\mathcal{N}^{ring}$. After the $\mathcal{N}^{ring}$ is empty, the items in M^{do}, of which exist in $\mathcal{T}_2$, are added in M^{ro}.

Step 3. Add edges back. The algorithm adds the edge back into $\mathcal{T}_2$, which is in $\mathcal{E}_1^{deleted}$ or $\mathcal{E}_2^{deleted}$, and the endpoints are both in $\mathcal{T}_2$. The final tree is the reshaped ontology $\mathcal{G}^{ro}$.

2.4 KG Construction

The *KG Construction* module takes the reshaped schema $\mathcal{G}^{ro}$, the *Raw-to-RO Mapping* M^{ro} and the *Raw Data* as inputs, and generates a series corresponding *KG*. We enumerate all class nodes in $\mathcal{G}^{ro}$. For each node and its datatype property edges, we find the primary keys for the node and the attributes for the edge respectively in the mapped tables and attributes in the *Raw Data* via M^{ro}, and create an entity for each key, and create datatype properties for each such edge. Next, we enumerate all object property edges in $\mathcal{G}^{ro}$, find the mapped foreign keys in the *Raw Data* via M^{ro}, and create links (object properties) between the entity represented by the primary key and the entity represented by the foreign key.

3 DEMONSTRATION SCENARIOS

During demonstration we will present *ScheRe* with an industrial scenario for Welding KG construction at Bosch. Data are collected from a factory for resistance spot welding in Germany, which is a world-widely applied process at many plants of Bosch and Bosch's renowned customers. Scenario allows demo attendees to experience KG schema reshaping, KG construction and querying with more user-friendly KGs.

Scenario Description. Dataset is an anonymised sample snippet of relational tables collected from production line, including 1000 welding operations (estimated to be relevant to 100 cars). In particular, data are comprised of 176 attributes of welding control parameters, monitor status, and quality indicators. Domain KG schema is a domain ontology designed by domain experts and knowledge engineers, consisting of 1181 axioms that describe 210 classes, 203 object properties, and 191 datatype properties. *Raw-to-DO Mapping* M^{do} is generated by domain experts that map table names and attribute names in raw data to nodes and edges in domain KG schema. For KG schema reshaping, attendees input user information to indicate main node and to elevate normal attributes to class nodes. Then *ScheRe* will reshape the domain KG schema to a data-specific KG schema. n a KG will be constructed based on reshaped KG schema. Attendees will see statistics of KG constructed based on domain KG schema (KG^{do}) and KG based on reshaped KG schema (KG^{ro}), and query over them, to experience that KGs and querying become much user-friendly.

Tasks. We selected 7 tasks that should reach a balance between using system and maintaining a controllable scope. Tasks include two types. Task Type 1 is to input user information for schema reshaping, including three tasks, pointing main node, elevating attributes, and add entity node. Task Type 2 is to select one query from four options (only one option is correct) to perform data inspection or diagnostics in KG with reshaped KG schema and in KG (KG^{do}) with domain schema as schema, including 4 query selection tasks. Task Type 2 measure usability of using our schema reshaping system, and Task Type 2 compares users' perception of querying KGs with and without schema reshaping.

Acknowledgements. We thank Xianghui Luo (ACM Member) for supporting the GUI development. The work was partially supported by H2020 projects Dome 4.0 (Grant Agreement No. 953163), OntoCommons (Grant Agreement No. 958371), and DataCloud (Grant Agreement No. 101016835) and SIRIUS Centre, Norwegian Research Council project number 237898.

REFERENCES

[1] Onur Celik, Dongzhuoran Zhou, Ge Li, Philipp Becker, and Gerhard Neumann. 2022. Specializing Versatile Skill Libraries Using Local Mixture Of Experts. In *Conference on Robot Learning*. PMLR, 1423–1433.

[2] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. 2011. Reverse Data Exchange: Coping With Nulls. *ACM Trans. Database Syst.* 36, 2 (2011), 11:1–11:42.

[3] Miao Fan, Qiang Zhou, Emily Chang, and Fang Zheng. 2014. Transition-Based Knowledge Graph Embedding With Relational Mapping Properties. In *PACLIC 28*. 328–337.

[4] Nicola Guarino, Daniel Oberle, and Steffen Staab. 2009. What is an Ontology? In *Handbook on ontologies*. Springer, 1–17.

[5] Ramanathan Guha, Rob McCool, and Eric Miller. 2003. Semantic search. In *WWW 12*. 700–709.

[6] Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. 2014. Everything You Always Wanted To Know About Blank Nodes. *Journal of Web Semantics* 27 (2014), 42–69.

[7] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. 2021. Knowledge Graphs. *CSUR* 54, 4 (2021), 1–37.

[8] ITU. 2012. *Recommendation ITU – T Y.2060: Overview of the Internet of Things*. Technical Report. International Telecommunication Union.

[9] Guoliang Ji et al. 2015. Knowledge Graph Embedding Via Dynamic Mapping Matrix. In *IJCKG*. 687–696.

[10] Henning Kagermann. 2015. Change Through Digitization – Value Creation in the Age of Industry 4.0. In *Management of Permanent Change*. Springer, 23–45.

[11] Haruhiko Kaiya and Motoshi Saeki. 2006. Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In *RE'06*. IEEE, 189–198.

[12] Dimitri Kartsaklis, Mohammad Taher Pilehvar, and Nigel Collier. 2018. Mapping Text To Knowledge Graph Entities Using Multi-Sense LSTMs. *arXiv preprint arXiv:1808.07724* (2018).

[13] Christoph Naab and Zhuoxun Zheng. 2022. Application Of The Unscented Kalman Filter In Position Estimation A Case Study On A Robot For Precise Positioning. *Robotics and Autonomous Systems* 147 (2022), 103904.

[14] Martin Ringsquandl, Evgeny Kharlamov, Daria Stepanova, Steffen Lamparter, Raffaello Lepratti, Ian Horrocks, and Peer Kröger. 2017. On Event-Driven Knowledge Graph Completion in Digital Factories. In *Big Data 2017*. IEEE, 1676–1681.

[15] Barry Smith. 2012. Ontology. In *The furniture of the world*. Brill, 47–68.

[16] Ahmet Soylu, Óscar Corcho, Brian Elvesæter, Carlos Badenes-Olmedo, Tom Blount, Francisco Yedro Martínez, Matej Kovacic, Matej Posinkovic, Ian Makgill, Chris Taggart, Elena Simperl, Till C. Lech, and Dumitru Roman. 2022. Theybuyforyou Platform And Knowledge Graph: Expanding Horizons In Public Procurement With Open Linked Data. *Semantic Web* 13, 2 (2022), 265–291.

[17] Muhammad Yahya, Baifan Zhou, Zhuoxun Zheng, Dongzhuoran Zhou, John G Breslin, Muhammad Intizar Ali, and Evgeny Kharlamov. 2022. Towards Generalized Welding Ontology In Line With Iso And Knowledge Graph Construction. *ESWC (Posters & Demos), Springer* (2022).

[18] Zhuoxun Zheng et al. 2022. Executable Knowledge Graph for Machine Learning: A Bosch Case for Welding Monitoring. In *ISWC*.

[19] Zhuoxun Zheng et al. 2022. Executable Knowledge Graph for Transparent Machine Learning in Welding Monitoring at Bosch. In *CIKM*.

[20] Zhuoxun Zheng et al. 2022. ExeKG: Executable Knowledge Graph System for User-friendly Data Analytics. In *CIKM*.

[21] Zhuoxun Zheng et al. 2022. Towards a Visualisation Ontology for Data Analysis in Industrial Applications. In *ESWC*.

[22] Zhuoxun Zheng, Baifan Zhou, Dongzhuoran Zhou, Gong Cheng, Ernesto Jiménez-Ruiz, Ahmet Soylu, and Evgeny Kharlamov. 2022. Query-based industrial analytics over knowledge graphs with ontology reshaping. *ESWC (Posters & Demos), Springer* (2022).

[23] Baifan Zhou et al. 2022. Knowledge Graph-Based Semantic System for Visual Analytics in Automatic Manufacturing. In *ISWC*.

[24] Baifan Zhou, Zhuoxun Zheng, Dongzhuoran Zhou, Gong Cheng, Ernesto Jiménez-Ruiz, Trung-Kien Tran, Daria Stepanova, Mohamed H Gad-Elrab, Nikolay Nikolov, Ahmet Soylu, et al. 2022. The Data Value Quest: A Holistic Semantic Approach At Bosch. In *ESWC*. Springer, 287–290.

[25] Baifan Zhou, Zhuoxun Zheng, Dongzhuoran Zhou, Ernesto Jimenez-Ruiz, Gong Cheng, Trungkien Tran, Daria Stepanova, Mohamed H. Gad-Elrab, Nikolay Nikolov, Ahmet Soylu, and Evgeny Kharlamov. 2022. Exploiting The Values Of Data: A Holistic Semantification Approach At Bosch. In *ESWC (Demos/Industry)*. Springer.

[26] Baifan Zhou, Dongzhuoran Zhou, Jieying Chen, Yulia Svetashova, Gong Cheng, and Evgeny Kharlamov. 2021. Scaling Usability of ML Analytics With Knowledge Graphs: Exemplified with A Bosch Welding Case. In *IJCKG*.

[27] Dongzhuoran Zhou et al. 2022. Towards Executable Knowledge Graph Translation. In *ISWC*.

[28] Dongzhuoran Zhou, Baifan Zhou, Jieying Chen, Gong Cheng, Egor Kostylev, and Evgeny Kharlamov. 2021. Towards Ontology Reshaping For Kg Generation With User-In-The-Loop: Applied To Bosch Welding. In *IJCKG*. 145–150.

[29] Dongzhuoran Zhou, Baifan Zhou, Zhuoxun Zheng, Egor V Kostylev, Gong Cheng, Ernesto Jimenez-Ruiz, Ahmet Soylu, and Evgeny Kharlamov. 2022. Enhancing Knowledge Graph Generation With Ontology Reshaping–Bosch Case. *ESWC (Demos/Industry), Springer* (2022).

[30] Dongzhuoran Zhou, Baifan Zhou, Zhuoxun Zheng, Ahmet Soylu, Gong Cheng, Ernesto Jimenez-Ruiz, Egor V. Kostylev, and Evgeny Kharlamov. 2022. Ontology Reshaping for Knowledge Graph Construction: Applied on Bosch Welding Case. In *ISWC*. Springer.