Received July 1, 2021, accepted August 7, 2021, date of publication August 12, 2021, date of current version August 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3104293

ProGOMap: Automatic Generation of Mappings From Property Graphs to Ontologies

NAGLAA FATHY, WALAA GAD, NAGWA BADR, AND MOHAMED HASHEM
Department of Information Systems, Faculty of Computer and Information Sciences, Ain Shams University, Cairo 11566, Egypt
Corresponding author: Naglaa Fathy (naglaa_fathy@cis.asu.edu.eg)

ABSTRACT Property Graph databases (PGs) are emerging as efficient graph stores with flexible schemata. This raises the need to have a unified view over heterogenous data produced from these stores. Ontology based Data Access (OBDA) has become the most dominant approach to integrate heterogeneous data sources by providing a unified conceptual view (ontology) over them. The corner stone of any OBDA system is to define mappings between the data source and the target (domain) ontology. However, manual mapping generation is time consuming and requires great efforts. This paper proposes ProGOMap (**Pro**perty **G**raph to **O**ntology **M**apper) system that automatically generates mappings from property graphs to a domain ontology. ProGOMap starts by generating a putative ontology with direct axioms from PG. A novel ontology learning algorithm is proposed to enrich the putative ontology with subclass axioms inferred from PG. The putative ontology is then aligned to an existing domain ontology using string similarity metrics. Another algorithm is proposed to align object properties between the two ontologies considering different modelling criteria. Finally, mappings are generated from alignment results. Experiments were done on eight data sets with different scenarios to evaluate the effectiveness of the generated mappings. The experimental results achieved mapping accuracy up to 97% and 81% when addressing PG-to-ontology terminological and structural heterogeneities, respectively. Ontology learning by inferring subclass axioms from a property graph helps to address the heterogeneity between the PG and ontology models.

INDEX TERMS Property graph database, resource description framework, ontology engineering, ontology alignment, graph model heterogeneity.

I. INTRODUCTION

Property Graph databases (PGs) are extensively used in different domains e.g., social networks and web applications because of their scalability, persistent data, flexible schemata, etc. However, the lack of standardization in this kind of data sources raises the need to have a unified view over them. Ontology based data Access (OBDA) approaches best fit this need because ontologies provide a formal specification of an application domain. A typical OBDA system comprises three layers: a data source (storing the data), a domain ontology (conceptual model), and a set of mappings between them which is the key component [1], [2].

Mappings describe how different parts of the data source model correspond to different concepts of the ontology. They are defined based on one of two approaches; global-as-view (GAV), or local-as-view (LAV) [3]. In GAV approach, the global schema is expressed from the local data source perspective. In LAV approach, the local data source is expressed from the global schema perspective. These mappings are expressed in a standard language i.e. R2RML, for relational-to-RDF [4] or xR2RML, for non-relational-to-RDF [5]. Creating these mappings manually is hard, time-consuming and requires domain expertise. Therefore, efforts towards automated generation of these mappings from a data source to a domain ontology are gaining attention.

In [6], An automated process for generating relational-to-ontology mappings is proposed through three main steps. At first, an ontology, called putative ontology, is automatically derived from the relational database schema. The generated ontology has a flat structure as opposed to a domain ontology created by experts. In the next step, ontology matching methods are applied to align the putative ontology with a target domain ontology. Finally, mappings are derived from those alignments between both ontologies.

In this paper, a novel ProGOMap system (**P**roperty **G**raph to **O**ntology **M**apper) is proposed. It automatically generates mappings from NoSQL property graphs to an existing domain

The associate editor coordinating the review of this manuscript and approving it for publication was Adnan Abid.

ontology expressed in OWL. Mapping generation process is in line with the general strategy described in [6]. More specifically, ProGOMap starts by generating a putative ontology with direct axioms from the PG model. It then expands the putative ontology by inferring class hierarchies from PG. Next, the putative ontology is aligned to an existing domain ontology using string similarity metrics. Object properties in the putative ontology are aligned by considering class hierarchies of the property's domain/range to enrich the alignment process. Finally, mappings are generated using the alignment results and the PG model. The main contributions of this paper include:

- Proposing an ontology learning method that infers class hierarchies from a property graph database, considering different modelling patterns (Algorithm1).
- Proposing an ontology alignment method that consumes the inferred class hierarchies to align object properties from the putative ontology to an existing ontology (Algorithm2).
- Generating PG-to-Ontology mappings from the aligned axioms and enriching them with inferred data/object properties.

The rest of this paper is organized as follows: related work is covered in section 2. Challenges in mapping property graphs to ontologies are discussed in section 3. The proposed system is described in section 4. Experiments and evaluation are discussed in section 5. Conclusion and future work are presented in section 6.

II. RELATED WORK

Various research studies were introduced for mapping diverse data sources to ontologies. These studies could be categorized into mapping: Relational Database-to-Ontology, Knowledge Graph-to-Ontology, and Property Graph-to-Ontology.

A. RELATIONAL DATABASE-TO-ONTOLOGY MAPPING GENERATION

State-of-the-art systems addressed diverse challenges for automatically mapping relational databases to ontologies. MIRROR [7] automatically generated R2RML mappings either by following direct mapping approach, or by using implicit information (i.e. subclasses and many-to-many relationships) encoded in relational database schema. However, the system assumed using a complete normalized relational database, which is not true in real-world scenarios. IncMap [8] merged the graphs of the relational schema and the ontology to create a pairwise connectivity graph (PCG) for representing possible matches. It addressed the semantic heterogeneity between both models by applying reasoning over the target ontology. However, it supported only 1:1 and 1:n class-to-table mapping generation but did not provide correspondences for n:m or n:1 relational-to-ontology mapping.

BootOx [9] generated bootstrapped mappings by following W3C direct mapping rules as well as advanced mapping techniques. The system provided mappings for different ontological profiles depending on the required application scenario. It also offered to integrate a domain ontology to the bootstrapped one through alignment, or directly mapped it to the database. However, it provided only one-to-one relational-to-ontology mapping correspondences but cannot match multiple ontology classes to the same table. AutoMap4OBDA [10] automatically generated R2RML mappings from the schema and content of relational sources by applying ontology learning techniques and string similarity metrics.

Milan [11] proposed a multi-level algorithm to find table-to-class, column-to-data property and foreign key-to-object property correspondences between a relational database and an ontology. The system employed Levenstein distance-based fuzzy technique for label matching and improved the matching scores using combinatorial optimization technique. However, further improvements are needed for class-table mapping generation.

B. KNOWLEDGE GRAPH-TO-ONTOLOGY MAPPING GENERATION

Research papers were proposed for the integration of different knowledge graphs into a unified ontology. In [12], statistical data analysis measures were used to generate ontology axioms from large RDF datasets by running SPARQL queries on them. RDF data were divided into blocks, depending on disjoint properties, to execute the querying process in parallel. MostoDEx prototype [13] presented an automatic generation of schema mapping between RDF knowledge bases in Global-Local-As-View manner (GLAV). Matches between entities in the source and target knowledge bases are referred to as Correspondences. The proposed method depended on using a set of input n:m correspondences and a single input exchange sample. An exchange sample was composed of two subsets from the exchanging source and expected target data, respectively. However, it depended on user-provided informative examples to correctly generate mapping rules, which required a user to be completely aware of knowledge base translation.

In [14], authors proposed a method that exploited existing ontologies to build a unified schema over knowledge graphs in an incremental manner. The incremental building process started by loading hierarchical axioms (rdfs:subClassOf) from ontologies into a single graph. Then, additional axioms (i.e., owl:equivalentClass) were added to the ontology graph. Finally, an ontology matching process was applied to the set of ontologies to enrich edges with related concepts. Although, the connectedness of the obtained graph was improved by adding more axioms and mappings, some limitations to the approach were highlighted by authors.

C. PROPERTY GRAPH-TO-ONTOLOGY MAPPING GENERATION

Only few approaches were proposed for mapping a Property Graph (PG) to an ontology. In [15], three models were proposed to represent key/value properties of property graph edges in RDF. These models were based on named graphs, extended reification, and sub properties, respectively.

The proposed approach was implemented on a commercial database engine which demonstrated its feasibility, although some limitations were presented. In the first model, not all RDF systems support named graphs. Also, there is no formal semantics for reification required in the second model, which makes it difficult for querying data.

In [16], authors defined a new ontology model, called PGO, and proposed an automatic mapping procedure from property graphs to PGO. Axioms of PGO ontology describe different elements of the property graph (pgo:Node, pgo:Edge, pgo:property, etc.) and connections between them (pgo:hasProperty, pgo:startNode, pgo:endNode, etc.). Mapping a property graph to PGO ontology is carried out through iterative transformations of labels and properties for graph nodes and edges. However, the mapping method is suitable only for PGO ontology that has its axioms defined specifically for property graph elements. The proposed procedure could not be employed to map the property graph to an existing domain ontology with different axioms. Yet, current literature lacks proposals for automatic generation of mapping from PG to an existing domain ontology.

In the opposite direction, some proposals were presented for mapping RDF-to-PG. In [17], a Graph to Graph Mapping Language (G2GML) is defined to map RDF graphs to PGs. Five types of mapping were designed, namely, (i) resource to node, (ii) datatype property to node property, (iii) object property to edge, (iv) resource to edge and (v) datatype property to edge property. Authors also proposed several serialization formats for the PG model considering the differences in existing models. Converted RDF data could be loaded into various graph database engines for further analysis. Although G2GML has several advantages over direct RDF-to-PG mapping, it requires user intervention to write the mapping file for RDF subgraphs that match certain SPARQL patterns.

In [18], three database mappings were proposed to transform RDF into PG. The first kind of mappings was classified as a simple mapping that ignores schema restriction from both sides. The second kind was a generic mapping that followed PG schema restrictions only, and the last kind was a complete mapping that considered PG and RDF schema and instances. Proposed mappings are all semantics preserving, which indicates that a valid database always results from a database mapping. However, they do not support RDF inference rules and reified RDF data.

III. PG-TO-ONTOLOGY MAPPING CHALLENGES

A property graph (PG) [19] consists of a set of labeled vertices connected by a set of labeled directed edges. Unique identifiers within the graph are assigned to each vertex and edge. A collection of properties (key/value pairs) may be associated to vertices and edges of PG. On contrary, a knowledge graph (aka as RDF graph) [20] modeled by an ontology, is a directed graph represented by triples. Each triple consists of a subject (a resource), a predicate (a labeled resource) and an object (a resource or a literal). SPARQL query language is used to query Knowledge graphs [21], whereas there is no standard query language for PGs although some popular declarative languages are proposed i.e. Cypher [22]. According to [18], [23], [24] and [25], PG-to-ontology mapping challenges could be classified into terminological heterogeneity, structural heterogeneity, and semantic heterogeneity:

A. TERMINOLOGICAL HETEROGENEITY

It occurs when different naming conventions are used by designers of ontologies and PGs to describe the same domain. These include abbreviations (e.g., conference vs. conf.), hyphenation, camel case, plural vs. single nouns, etc. Therefore, the main challenge is to find matches despite these differences.

B. STRUCTURAL HETEROGENEITY

It depends on the way used to model data of the same domain in either ontologies or PGs. Structural heterogeneity could result from: key conflicts, edge property conflicts, and class hierarchy conflicts:

1) KEY IDENTIFIERS

In PGs, key identifiers are defined by unique constraints, whereas identifiers for individuals in ontologies are defined by IRIs (Internationalized Resource Identifiers). Therefore, appropriate IRIs should be generated from the unique keys when representing ontology individuals.

2) EDGE PROPERTIES

They represent the *core* difference between PGs and ontologies [15]. Edges in PGs may be assigned a set of properties in the form of key/pairs. However, edges in RDF are only assigned labels that define the object property. Therefore, the main challenge is to correctly map an edge property in a property graph to its equivalent structure/axiom in an ontology.

3) CLASS HIERARCHIES

Contrary to ontologies, property graphs model class hierarchies (subsumption relationships) implicitly, following different design patterns discussed in [26]. Finding implicit class hierarchies from PG is the main challenge in this type. Table 1 shows different types of class hierarchies mapping challenge with respect to property graph modeling patterns.

C. SEMANTIC HETEROGENEITY

It refers to differences in semantic expressiveness between two data sources modeling the same domain. Ontology axioms are interpreted differently from similar statements in property graphs. For instance, a subsumption relationship in RDF results in inheriting all data/object properties from a superclass to its subclass by reasoning. Conversely, a subsumption relationship in PG is simply manipulated as an edge between two nodes with no more implicit information that could be derived. Also, existence constraints in PGs are used to ensure integrity of data, thus adopting a Closed World Assumption (assuming complete knowledge

TABLE 1. Possible class hierarchies from different modeling patterns in PG.

Mapping Challenge Type	PG Modeling Pattern Description
[n:1] Class-to-Node Match without categorical property	In this pattern, there is one node in the property graph for each ontology class in the hierarchy, including the superclass. There is one edge between the superordinate node and each subordinate node in the property graph which leads to having n sub nodes matched to a single node. This helps inheriting properties from the super-node to sub-nodes. The challenge is to find the central node that connects other subordinate nodes.
[n:1] Class-to-Node Match with categorical property	In this pattern, a single node in the property graph corresponds to several subclasses of an ontology. Such node uses additional properties to indicate each subclass. The challenge is to apply appropriate filtering operations to retrieve information about different classes from the same node.
[1: n] Class Property-to- Node Property Match	In this pattern, each ontology subclass is represented by a separate node in the property graph. But the superclass of the class hierarchy is not materialized in the property graph, and therefore, inherited properties are materialized in each node separately. In consequence, the same ontology property must be mapped to several nodes. The challenge is to find taxonomic relations between node labels.
[1:1] Class-to-Edge Match	In this pattern, multiple edges exist between the same couple of nodes, forming a multigraph. Each edge in the multigraph is a candidate class in the hierarchy that is subordinate to either the edge's source or target nodes. The challenge is to find semantic relatedness between edges in the multigraph and their source/target nodes.

about the world). On contrary, existence qualifiers in ontologies do not check data integrity, rather they imply new facts (i.e., unknown information). Therefore, ontologies are related to Open World Assumption (assuming incomplete knowledge about the world) [27], [28].

IV. THE PROPOSED ARCHITECTURE

ProGOMap (Property Graph to Ontology Mapper) is proposed to automatically generate mappings from a property graph to an existing target ontology (referred as domain ontology). As shown in Fig.1, ProGOMap consists of three main modules: (i) Ontology Bootstrapping from PG, (ii) Aligning Bootstrapped (Putative) Ontology to Domain Ontology, and (iii) Generating PG-to-Ontology Mappings.

The first module generates a putative ontology by deriving a set of direct axioms from the PG schema, followed by inferring subclass axioms from PG database through the proposed Algorithm 1. The second module aligns the putative ontology to the domain ontology using string matchers for classes and data properties, then it aligns object properties through the proposed Algorithm 2. The last module generates final PG-to-ontology mappings from the alignments (correspondences)

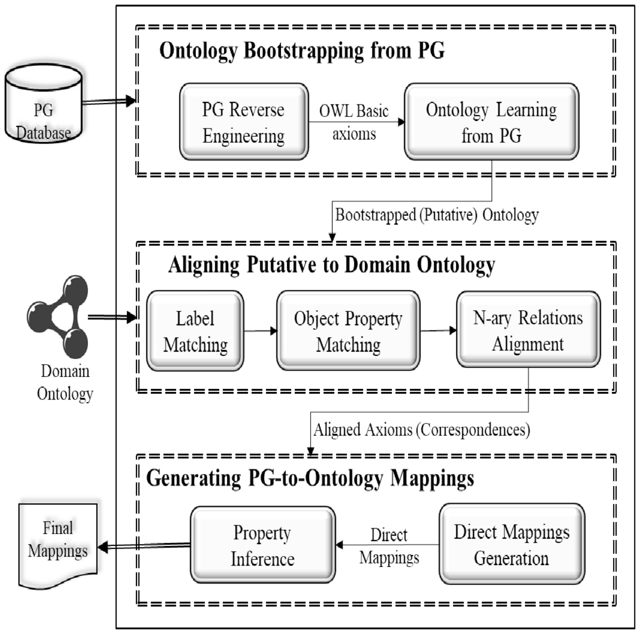

Ontology Bootstrapping from PG
PG Database
PG Reverse Engineering
OWL Basic axioms
Ontology Learning from PG
Bootstrapped (Putative) Ontology
Aligning Putative to Domain Ontology
Domain Ontology
Label Matching
Object Property Matching
N-ary Relations Alignment
Aligned Axioms (Correspondences)
Generating PG-to-Ontology Mappings
Final Mappings
Property Inference
Direct Mappings
Direct Mappings Generation

FIGURE 1. ProGOMap proposed architecture.

obtained from module 2. In this module, property inference is applied to enrich the generated mappings.

A. ONTOLOGY BOOTSTRAPPING FROM PG

Ontology bootstrapping refers to the process of automatically generating an ontology, named putative, from the database schema. The input to this module is the property graph database (Definition 3), and the output is the putative ontology generated in two steps: PG Reverse Engineering, and Ontology Learning. Ontology basic axioms are extracted from the property graph schema (Definition 2) in the reverse engineering step. Then, PG database content (Definition 1) is used together with PG schema in the ontology learning step to enrich the putative ontology with subclass axioms.

- *Definition 1 (A Property Graph [26]):* Let G = (V, E, ρ, λ, σ) be a property graph, where:
  - V is a finite set of nodes (vertices),
  - E is a finite set of edges.
  - ρ: E $\rightarrow$ v $\times$ v is a total function that assigns a pair of nodes to each edge.
  - λ: (V $\cup$ E) $\rightarrow$ SET$^+$(L) is a labeling function for nodes and edges with a set of labels from L.
  - σ: (V $\cup$ E) $\times$ P $\rightarrow$ SET$^+$(U) is a partial function that assigns properties (P) and their values from (U) to nodes/edges.
- *Definition 2 (A Property Graph Schema [26]):* Let S = (T$_V$, T$_E$, β, δ) be a property graph schema, where:
  - T$_V$ $\subset$ L is a finite set of labels representing node types.

- $T_E \subset L$ is a finite set of labels representing edge types.
- β: $(T_V \cup T_E) \times P \rightarrow T$ is a partial function that defines the properties and their datatypes for node and edge types.
- δ: $(T_V, T_V) \rightarrow SET^+(T_E)$ is a partial function that defines the edge types allowed between a given pair of node types.

- *Definition 3 (A Property Graph Database [26]):* Is a pair D(S, G) where S is a property graph schema and G is a property graph data.

1) PG REVEERSE ENGINEERING

Reverse engineering of a property graph database allows to infer its schema and represent the node labels and edge types with their property keys, indexes, and constraints [29]. PG Schema is used to extract basic axioms of the putative ontology according to a set of rules, as shown in Table 2. Fig.2 shows an example of the generated putative ontology after applying these rules to the property graph schema:

Rule 1 (R1): Each node label in the property graph schema is mapped to a class axiom in the putative ontology.

TABLE 2. Rules to represent property graph (PG) parts as OWL axioms (written in Turtle syntax).

#	PG part	Ontology Part	OWL Axiom
R1	Node with one label (L)	A class with name L	: CL rdf:type owl:class.
R2	Node with many labels (L_1, L_2…, L_n)	n classes named (L_1…L_n) with an equivalentClass axiom between these classes	: CL_1 rdf:type owl: class. : CL_2 rdf:type owl:class. … : CL_n rdf:type owl:class : CL_1 owl:equivalentClass : CL_2…: CL_n

R3	Property (P) in node labeled (L) of PG type (t)	A data property P associated to the class CL and datatype dt (matched from PG to XSD)	: P rdf:type owl:DatatypeProperty;
			rdfs:domain : CL;
			rdfs:range dt.

R4	Edge labeled (E) from source node (S) to target node (T)	An object property E associated to the classes CS and CT	: E rdf:type owl:ObjectProperty;
			rdfs:domain : CS;
			rdfs:range CT.

R5	Property (P) of type (t) in edge labeled (E) from source node (S) to target node (T)	**5.1.** A class with name (S-E-T) and A data property P associated to the class CS-E-T and datatype dt	: CS-E-T rdf:type owl: class.
			: P rdf:type owl:DatatypeProperty;
			rdfs:domain : CS-E-T;
			rdfs:range dt.

		5.2. An object property (E-S) associated to the classes CS and CS-E-T (left-side of original edge)	: E-S rdf:type owl:ObjectProperty;
			rdfs:domain : CS;
			rdfs:range CS-E-T.

		5.3. An object property (E-T) associated to the classes CT and CS-E-T (right-side of original edge)	: E-T rdf:type owl:ObjectProperty;
			rdfs:domain : CT;
			rdfs:range CS-E-T.

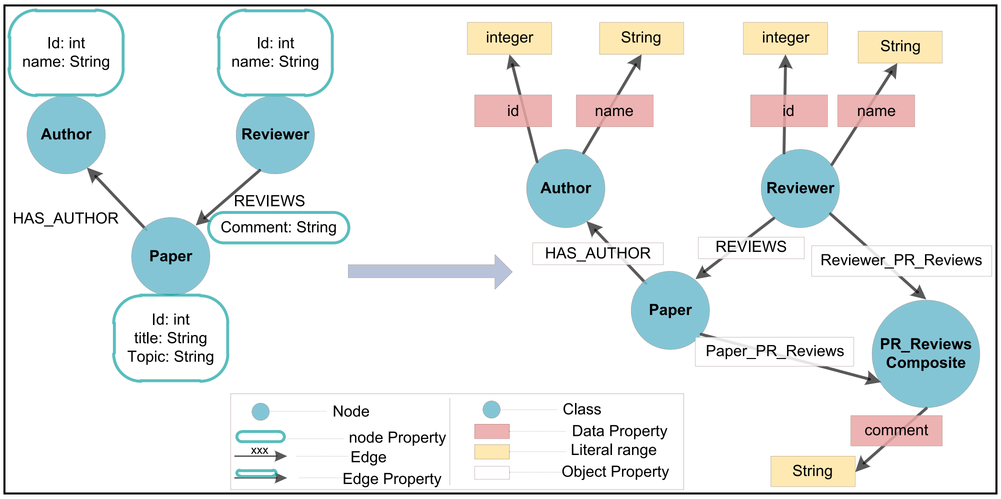


FIGURE 2. Example of basic putative ontology generation (right) from PG (left).

Rule 2 (R2): Multiple labels assigned to the same node in the property graph schema are referred to as equivalent classes in the putative ontology.

Rule 3 (R3): Data types from PG are mapped to their counterpart types in OWL according to XML schema datatypes. Top datatype (rdfs:Literal) is used when a given datatype in PG is not supported by OWL2 (i.e., xsd:double). If same property name is already added to ontology, then the domain of this property is changed to (owl:unionOf) axiom that consists of names of all classes having this property.

Rule 4 (R4): Each edge label in the property graph schema is mapped to an object property in the putative ontology. The edge's source and target nodes represent the object property's domain and range, respectively.

Rule 5 (R5): Edge properties in PG are represented as n-ary relations in ontology. Traditionally, in binary relations, an owl property links one individual to another individual or a value. On contrary, n-ary relations link an individual to more than one individual or value. For example, there is a "*comment*" property that is written by a certain "*Reviewer*" who reviewed a specific "*Paper*". In this case, "*comment*" property does not only link the "*Paper*" to a value but also to the "*reviewer*" who wrote this comment. N-ary relations problem is usually addressed in ontologies by adding a new class that represents the relation rather than just a property [30]. Binary links from the new class to each argument of the relation are provided by additional properties. As shown in the example in Fig.2, a new class "*PR_Reviews_Composite*" has an associated property "*comment*" and has two binary links to "*Reviewer*" and "*Paper*" classes, respectively. This pattern is called (reified relation).

2) ONTOLOGY LEARNING FROM PG

In this step, the putative ontology is enriched with subclass axioms inferred from PG schema and content. This is done by proposing Algorithm 1 that addresses the class hierarchy mapping challenge discussed in section III. The algorithm infers class hierarchies from five different PG patterns as follows: (i) Node property name, (ii) Node property values, (iii) Edge labels, (iv) Edge nodes, and (v) Nodes with different labels sharing a common category. Fig.3 shows an example for the first four patterns in the context of Conference domain.

The ***first pattern*** (lines 1-5) addresses the [n:1 class-to-node with categorical property] class hierarchy challenge, where node properties might describe different roles for the same node. For example, in Fig.3 (a), node "*Person*" has Boolean properties (*isAuthor*, *isReviewer*) which reveals that a person might be a reviewer or an author in the conference. In this regard, two subclasses, "*Author*" and "*Reviewer*" might be generated to the super class corresponding to "*Person*" node. This could be achieved when the property name has one of its hypernyms (its broader category) equal to the super class name. Individuals of each subclass are extracted from *true* values of the corresponding node property. That is why only Boolean properties could be considered in subclass generation from property names.

The ***second pattern*** (lines 6-10) also addresses the [n:1 class-to-node with categorical property] class hierarchy challenge, where some node properties might contain categorical values. For example, in Fig.3 (b), node "*Person*" has property "*Type*" that contains values (*Author*, *and Reviewer*). These values may contribute as subclasses to the superclass corresponding to "*Person*" node. Entropy-based estimation method of data diversity [31] is utilized to find categorical properties from PG nodes. In information theory, entropy measures the level of uncertainty in the possible outcomes of a given variable [32]. This method has achieved good performance when used for characterizing attributes in relational databases [33]. Therefore, it is adopted in this paper to find properties with the most balanced distribution of values. Given a set of nodes labeled (L), with property (Y), the entropy H(Y) of property Y is calculated as [31]:

$$H(Y) = \sum_{v \in \pi_y(L)} P_y(v) \cdot log P_y(v) \tag{1}$$

where π_Y(L) is the set of values in property Y over all nodes labeled L. P_Y(v) is the probability of having a node ∈ L with Y property equal to v. A Property with highest entropy (all its values are unique), and a property with lowest entropy (has highly duplicate values closest to single one) are not considered as categorical property candidates. Properties with entropy value less than the ontology entropy are stored as candidates for categorical properties. The entropy of the ontology is the maximum entropy over all classes. The class entropy is also calculated with the same equation above where P_Y(v) is the fraction of subclasses reachable at certain depth to the total number of subclasses at the ontology's maximum depth.

The ***third pattern*** (lines 11-17) addresses the [1:1 class-to-edge] class hierarchy challenge, where subclasses could be obtained from multigraphs. Multigraphs exist when multiple edges have the same end nodes. Each edge label is considered a candidate subclass when its hypernym equals to one of its end nodes. For example, in Fig.3 (c), there are two edges labeled (*reviews* and *hasAuthor*) between "*Person*" and "*Paper*" nodes. The noun words for these edges respectively are "*Reviewer*" and "*Author*", which have the same hypernym "*Person*". Therefore, both words are candidate subclasses to the "*Person*" node.

The ***fourth pattern*** (lines 18-23) addresses the [n:1 class-to-node without categorical property] class hierarchy challenge, where a central node is superordinate (hypernym) to other nodes connected to it. For example, in Fig.3 (d), "*Author*" and "*Reviewer*" nodes have "*isAuthor*" and

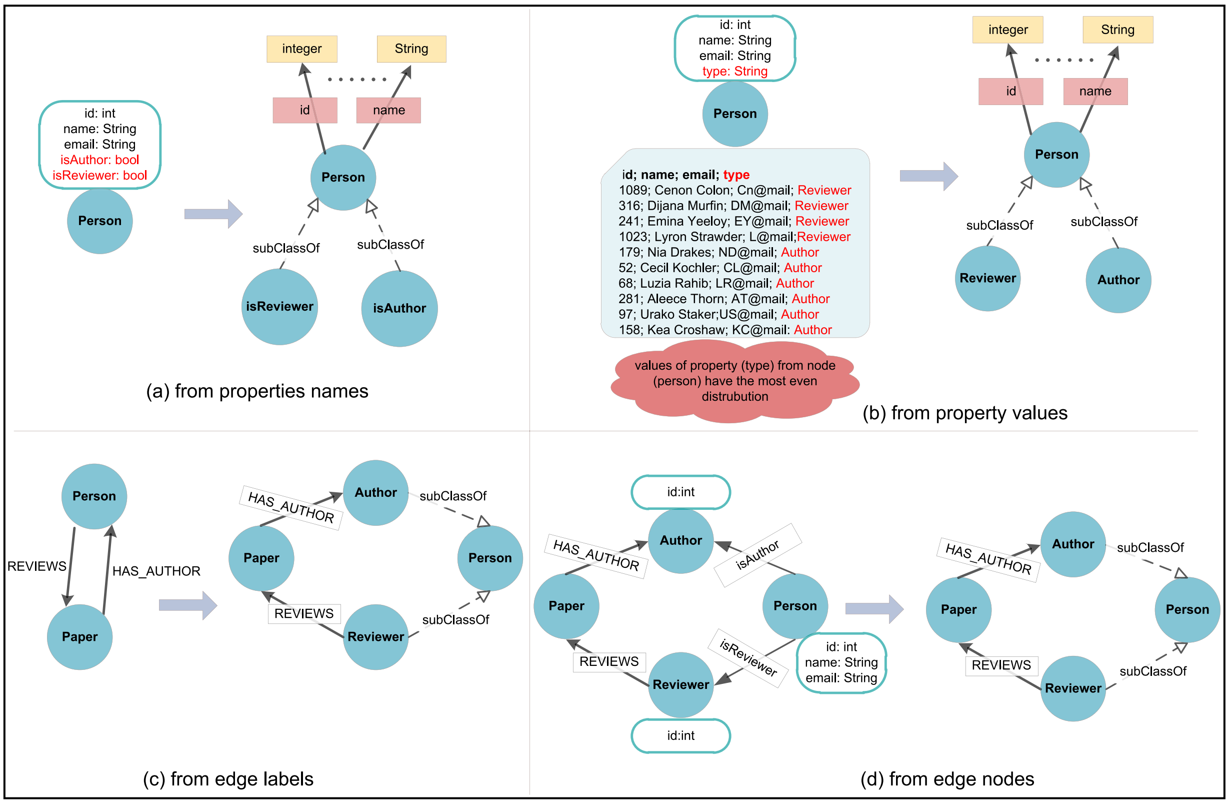
Algorithm 1 Class Hierarchy Inference From PG

```
     Input: Property Graph Schema (PG_S), Property Graph Data (PG_D), Initial Putative
     Ontology (putOnt)
     Output: Expanded Putative Ontology (putOnt)
     //A. Generating subClasses from node properties
1    FOR EACH node in PG_S.nodeList DO
2      FOR EACH p in node.properties DO
3        IF p.HasHypernym(node.label) AND p.IsBoolean() THEN
4          putOnt ← CreateOWLClass(p.name)
5          putOnt ← AddsubClassAxiom(p.name, node.label)
     //B. Generating subclasses from values of node properties
6        ELSE
7          p_entropy ← calculatePropertyEntropy(p)
8          IF p_entropy <= ontology_entropy AND p.IsValidProperty() THEN
9            p_values ← GetPropertyValues(PG_D,node.label, p.name)
10           putOnt ← CreateSubClassFrompropValues(p_values)
     //C. Generating subClasses from edge labels
11    multiGraph ← FindMultiGraph (PG_S.edgeList)
12    FOR EACH triple_{edge,source,target} in mutliGraph DO
13      matchedPair_{edge,node} ←
          FindMatchingHypernyms(triple.edgeType, triple.sourceLabel, triple.targetLabel)
14      IF matchedPair.isEmpty() AND putOnt.hasClass(matchedPair.edgeType) THEN
15        CONTINUE
16      putOnt ← CreateOWLClass(matchedPair.edgeType)
17      putOnt ← AddsubClassAxiom(matchedPair.edgeType,matchedPair.nodeLabel)
     //D. Generating subclasses from edge nodes
18    FOR EACH edge in PG_S.edgeList DO
19      IF HypernymRelated(edge.sourceLabel, edge. targetLabel) THEN
20        sourceKeyValues ← GetKeyValues(edge.sourceLabel)
21        targetKeyValues ← GetKeyValues(edge.targetLabel)
22        IF isSubsetOf(sourceKeyValues, targetKeyValues) OR
              isSubsetOf(targetKeyValues, sourceKeyValues)  THEN
23          putOnt ← AddsubClassAxiom(edge.sourceLabel, edge. targetLabel
      //E. Generating subclasses from labels of nodes sharing the same category
24    FOR EACH n1 in PG_S.nodeList DO
25      FOR EACH n2 in PG_S.nodeList DO
26      commonProps ← FindCommonProperties(n1.properties, n2.properties)
27      commonHypernym ← FindCommonHypernym(n1.label, n2.label)
28      IF commonProps.found() AND commonHypernym.found()  THEN
29        putOnt ← CreateOWLClass (commonHypernym)
30        putOnt ← AddsubClassAxiom(commonHypernym, n1.label)
31        putOnt ← AddsubClassAxiom(commonHypernym, n2.label)
```

"*isReviewer*" edges with "P*erson*" node, respectively. Additionally, the key property values of "*Author*" and "*Reviewer*" nodes are subset of "*Person*" node's key property. Therefore, both nodes are candidate subclasses to the "*Person*" node.

The ***fifth pattern*** (lines 24-31) addresses the [1:n class property-to-node property] class hierarchy challenge. It occurs when different labels of two or more nodes share the same category and have common properties in between but not connected by an edge. For example, a PG graph with "*Reviewer*" and "*Author*" nodes having common properties (*name* and *email*). The common hypernym "*Person*" between both nodes is retrieved and a new superclass is created with its name, then both nodes are candidate subclasses to it.

Information about correspondences between putative ontology and PG is stored separately to be used during alignment and mapping generation. This information describes each ontology axiom (owl:class, owl:DatatypeProperty, rdfs:subClassOf) and the original PG element associated to it (i.e., node, property name, property value, edge, etc.).



```
id; name; email; type
1089; Cenon Colon; Cn@mail; Reviewer
316; Dijana Murfin; DM@mail; Reviewer
241; Emina Yeeloy; EY@mail; Reviewer
1023; Lyron Strawder; L@mail;Reviewer
179; Nia Drakes; ND@mail; Author
52; Cecil Kochler; CL@mail; Author
68; Luzia Rahib; LR@mail; Author
281; Aleece Thorn; AT@mail; Author
97; Urako Staker;US@mail; Author
158; Kea Croshaw; KC@mail: Author
```

FIGURE 3. Example of SubClasses generation for putative ontology.

B. ALIGNING PUTATIVE ONTOLOGY TO DOMAIN ONTOLOGY

This module aligns the bootstrapped (putative) ontology to an existing domain ontology. Ontology alignment [34] is the process of finding correspondences between entities (i.e. classes, data properties, etc.) of two ontologies, which can be used for query answering, ontology merging, etc. A correspondence represents the semantic relationship between given two entities (i.e. equivalence and subsumption relationship) [35]. Finding correspondences between elements of two ontologies might be difficult for off-the-shelf ontology alignment systems, such as LogMap [36], when the source ontology is a putative ontology derived from another database. This is because the syntactical structure is specified on a high-level of abstraction in an existing domain ontology, whereas it is described on a very low level of granularity in a putative ontology derived from another database (PG). Therefore, detecting correspondences might be hindered when using structural metrics for ontology alignment. In this context, string similarity metrics are better used for the alignment between the putative and domain ontologies.

1) LABEL MATCHING

In this paper, two string matchers, namely StringAuto [37] and PropString [38] are used to match entity labels between the putative and domain ontologies, to address the terminological heterogeneity challenge. These matchers provide a list of string similarity metrics with a set of guidelines to help choosing the proper metric with the goal of maximizing either the precision or recall measures. These guidelines are based on the language of the ontologies, the number of words per tokenized entity label, and the embedded synonyms. In this step, two lists of correspondences are generated between the classes and data properties of the putative and domain ontologies, respectively. Class correspondences will be used in the next step to align object properties between both ontologies.

2) OBJECT PROPERTY MATCHING

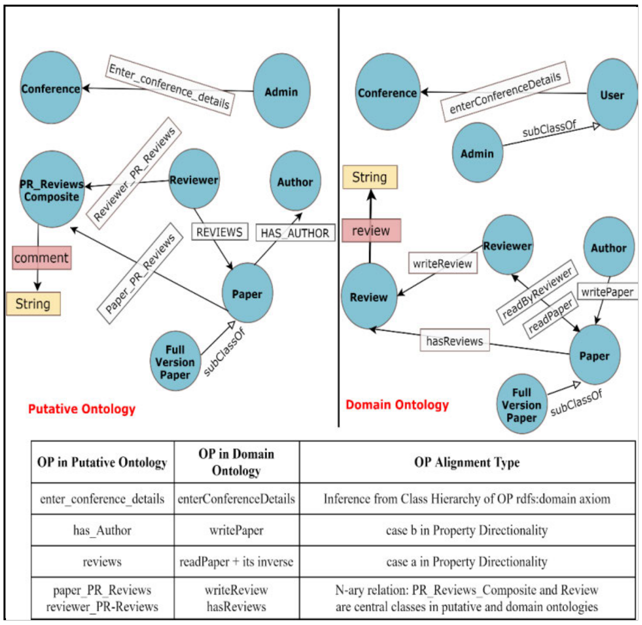
In this step, Algorithm 2 (lines 1-8) is proposed to align object properties between the putative and domain ontologies. Given a list of class correspondences between the putative and domain ontologies, two sets of object properties are obtained, one from a pair of putative classes, and the other from a pair of their equivalent domain classes (lines 1-5). In case the second set is not found between the pair of domain classes, then the class hierarchies for both domain classes are traversed upwards to search for possible object properties (lines 6-7). Then, object properties from the first pair are aligned to their counterparts from the second pair using string matchers while considering property directionality (line 8).The example in Fig. 4 shows that the object property between "*Conference*" and "*Admin*" classes in the putative ontology is aligned to its counterpart between the "*Conference*" and "*User*" classes in the domain ontology. This is because there is a subsumption relationship between "*User*" and "*Admin*" classes in the domai' ontology.

Algorithm 2 Ontology Alignment for Object Properties and N-ary Relations

```
   Input: Domain Ontology (domOnt), Putative
   Ontology (putOnt), ClassCorrespondences (CLC)
   Output: Aligned object properties(op_matches)
1  FOR EACH c1 in CLC.GetMatchedClasses() DO
2    FOR EACH c2 in CLC. GetMatchedClasses()  DO
3      op_putative ←
         GetObjectProperties(c1.putativeName,
         c2.putativeName, putOnt)
4      op_domain ←
         GetObjectProperties(c1.domainName,
         c2.domainName, domOnt)
5      IF op_putative.isNotEmpty() THEN
6        IF op_domain.isEmpty() THEN
7          op_domain ←
             GetObjectPropertiesFromClassHierarchy
             (c1.domainName, c2.domainName, domOnt)
8        op_matches ← ApplyStringMatcher
           &PropDirectionality(op_putative, op_domain)
9  FOR EACH put_rel in CLC.Get_Putative
     _Nary_Relations() DO
10   dom_rel ←
       Find_Domain_Nary_Relation
       (put_rel.domSourceClass,  put_rel.domTargetClass,
       domOnt)
11   IF dom_rel IS NOT NULL THEN
12     put_rel.domHeadClass = dom_rel.headClass
13     dp_matches ← MatchDataProperties
         (rel.putHeadClass, rel.domHeadClass,domOnt)
14     CLC ← Update_DP_Correspondences
         (dp_matches)
15     op_matches.ADD (put_rel.GetSourceOP,
         dom_rel.GetSourceOP)
16     op_matches.ADD put_rel.GetTargetOP,
         dom_rel.GetTargetOP)
```

Object Property Directionality for a pair of putative classes and a pair of aligned domain classes might differ. For example, the putative object property (Author - writes -> Paper) has different direction from the domain one (Paper- writtenBy-> Author). Additionally, there might be owl:inverseOf axioms to some object properties in domain ontology. Therefore, to properly align object properties between putative and domain ontologies, three cases should be considered:

a) There is only one putative/domain object property (op) in the ***same direction*** for a single pair of putative/domain classes, respectively. Then the putative op is aligned to the domain op and its inverse (if any). Fig.4 shows an example of aligning the object property "*reviews*" in the putative ontology to the object property "*readPaper*" in the domain ontology though they are not lexically similar.



OP in Putative Ontology	OP in Domain Ontology	OP Alignment Type
enter_conference_details	enterConferenceDetails	Inference from Class Hierarchy of OP rdfs:domain axiom
has_Author	writePaper	case b in Property Directionality
reviews	readPaper + its inverse	case a in Property Directionality
paper_PR_Reviews reviewer_PR-Reviews	writeReview hasReviews	N-ary relation: PR_Reviews_Composite and Review are central classes in putative and domain ontologies

FIGURE 4. Alignment example from putative (left) to domain (right) ontologies in the conference domain.

b) There is only one putative/domain object property (op) in the opposite direction for a single pair of putative/domain classes, respectively. Then the putative op is inferred to be aligned to domain op. Fig.4 shows an example of aligning the object property "Has_Author" in the putative ontology to the object property "writePaper" in the domain ontology although they are in different directions.

c) There are multiple putative/domain object properties in different directions for a single pair of putative/domain classes, respectively. Then string matcher is applied with direction prioritization. It means that the similarity score between a domain object property and a putative one is always prioritized over the similarity score between the inverse of the same domain object property and other putative one. This way avoids having a domain property and its inverse aligned to two different putative properties.

3) N-ARY RELATIONS ALIGNMENT

In this step, Algorithm 2 (lines 9-16) aligns n-ary relations between putative and domain ontologies. As mentioned in (sec. 3.1.1), n-ary relation in ontology represents edge properties in PG. The main component of this relation is a head (central) class that links edge properties together with edge's source and target nodes. Two object properties are created between this head class and putative classes corresponding to source and target nodes. In addition, one or more data properties representing the edge properties are associated to the head class. N-ary relation alignment is achieved by looking for an unaligned class in domain ontology which is linked to the pair of putative classes corresponding to source and target nodes.

If match is found, then data properties are aligned between this class and the putative central class. If multiple n-ary relations exist between same putative classes, then string matcher is applied to these relations to align each domain central class to its correct putative central class. As shown in Fig.4, the "*Review*" class in the domain ontology is considered the head class that is aligned to the "PR_*ReviewComposite*" head class in the putative ontology. Data property "*review*" is aligned to its counterpart "*comment*" property from head classes in the domain and putative ontologies, respectively.

C. PG-TO-ONTOLOGY MAPPING GENERATION

Generating mappings from the ontology alignment (set of correspondences) is the last module in the ProGOMap workflow. In this module, xR2RML mapping language [5] is used to write mappings. Triple maps are the main components of an xR2RML mapping document. Each triple map represents one RDF triple extracted from the domain ontology. It consists of a *logical source* query and three *term maps*. The logical source query (xrr:logicalSource) contains the PG nodes and/or edges to be mapped to current RDF triple. Term maps include subject map, predicate map, and object map. They are used to map data from the logical source to RDF terms (literal, IRI, blank node). This module consists of two steps: *Direct mapping generation*, and *Property_Inference*.

1) DIRECT MAPPING GENERATION

In this step, one TripleMap is generated for each single correspondence to provide a normalized mapping document. Fig.5 shows the structure of correspondences obtained from the second module for class, N-ary relation, and object property, respectively. **For each class correspondence**, an xR2RML *rr:logicalSource* includes a Cypher query that matches the node name and returns the node's key property. An xR2RML *rr:subjectMap* includes the node key property appended to the URI of the aligned class instance. An xR2RML *rr:predicateMap* includes the constant axiom [rdf:type], and an xR2RML *rr:objectMap* includes the aligned class name. **For each data property inside the class correspondence**, an xR2RML *rr:logicalSource* includes a Cypher query that matches the node name and returns a certain node property. An xR2RML *rr:subjectMap* includes the node's key property appended to the URI of the aligned class instance. An xR2RML *rr:predicateMap* includes the aligned data property name, and an xR2RML *rr:objectMap* includes the node property name.

For each subclass inside the class correspondence, two types of xR2RML triple maps might be generated. When the subclass is inferred from a node property, an xR2RML *rr:logicalSource* includes a Cypher query that matches the node name and returns that node property. An xR2RML *rr:subjectMap* includes the value of the returned node property appended to its aligned subclass URI. An xR2RML *rr:predicateMap* includes the constant axiom [rdf:type], and an xR2RML *rr:objectMap* includes the aligned subclass name. When the subclass is inferred from an edge, an xR2RML *rr:logicalSource* includes a Cypher query that matches the edge type between the source and target nodes to return the edge identifier (id). An xR2RML *rr:subjectMap* includes the edge id appended to its aligned subclass URI. xR2RML *rr:predicateMap* and *rr:objectMap* include [rdf:type] axiom and the aligned subclass name, respectively.

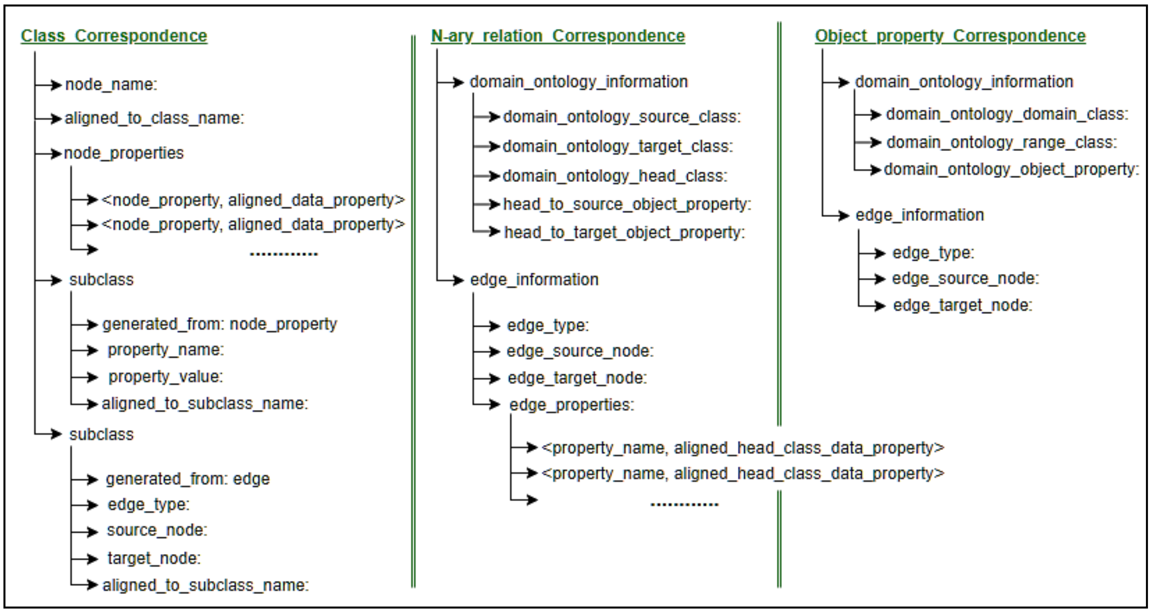

Class_Correspondence
node_name:
aligned_to_class_name:
node_properties
<node_property, aligned_data_property>
<node_property, aligned_data_property>
....
subclass
generated_from: node_property
property_name:
property_value:
aligned_to_subclass_name:
subclass
generated_from: edge
edge_type:
source_node:
target_node:
aligned_to_subclass_name:
N-ary_relation_Correspondence
domain_ontology_information
domain_ontology_source_class:
domain_ontology_target_class:
domain_ontology_head_class:
head_to_source_object_property:
head_to_target_object_property:
edge_information
edge_type:
edge_source_node:
edge_target_node:
edge_properties:
<property_name, aligned_head_class_data_property>
<property_name, aligned_head_class_data_property>
....
Object_property_Correspondence
domain_ontology_information
domain_ontology_domain_class:
domain_ontology_range_class:
domain_ontology_object_property:
edge_information
edge_type:
edge_source_node:
edge_target_node:

FIGURE 5. The structure of ontology alignment (correspondences) obtained for class (left), N-ary relation (middle), and object property (right).

For each N-ary relation correspondence, at least three xR2RML triple maps are generated. In the first two triple maps, the xR2RML *rr:logicalSource* terms include the same Cypher query that matches the edge type between the source and target nodes. The xR2RML *rr:subjectMap* for both triple maps include the source and target nodes' key properties appended to the URI of the head class instance. The xR2RML *rr:predicateMap* of the first triple map includes the head-to-source object property name, and the xR2RML *rr:objectMap* includes the source node's key property appended to the URI of its aligned class instance. The xR2RML *rr:predicateMap* of the second triple map includes the head-to-target object property name, and the xR2RML *rr:objectMap* includes the target node's key property appended to the URI of its aligned class instance. Finally, an xR2RML triple map is generated for each edge property aligned to the head class's data property.

```
<CLmapping_1>
  xrr:logicalSource [xrr:query  "Match (p:Paper) return p.id as Pid"];
   rr:subjectMap [rr:template
"http://www.examples.com/resource/Paper/{Pid}";];
   rr:predicateObjectMap [
      rr:predicateMap [ rr:constant rdf:type ];
      rr:objectMap    [ rr:constant  <http://cmt#Paper>; ];].
<CLmapping_3>
   xrr:logicalSource [xrr:query  "Match (p:Paper) where
p.isFullVersion=true return p.id as FpId"];
   rr:subjectMap [rr:template
"http://www.examples.com/resource/FullversionPaper/{FpId}";];
   rr:predicateObjectMap [
      rr:predicateMap [ rr:constant rdf:type ];
      rr:objectMap    [ rr:constant  <http://cmt#FullVersionPaper>; ];].
<DPmapping_10>
    xrr:logicalSource [xrr:query  "Match (r:Reviewer)-[e:reviews]-
>(p:Paper)  return r.id as Rid, p.id as Pid, e.comment as Comments"];
    rr:subjectMap [rr:template
"http://www.examples.com/resource/Review/{Rid}-{Pid}";];
    rr:predicateObjectMap [
       rr:predicateMap [ rr:constant <http://cmt#review> ];
       rr:objectMap    [ rr:column "Comments"];].
<OPmapping_5>
   xrr:logicalSource [xrr:query  "Match (r:Reviewer)-[e:reviews]->
(p:Paper) return r.id as Rid, p.id as Pid"];
   rr:subjectMap [ rr:template
"http://www.examples.com/resource/Paper/{Pid}";];
   rr:predicateObjectMap [
      rr:predicateMap [ rr:constant <http://cmt#readByReviewer> ];
      rr:objectMap    [ rr:template
"http://www.examples.com/resource/Reviewer/{Rid}" ];].
```

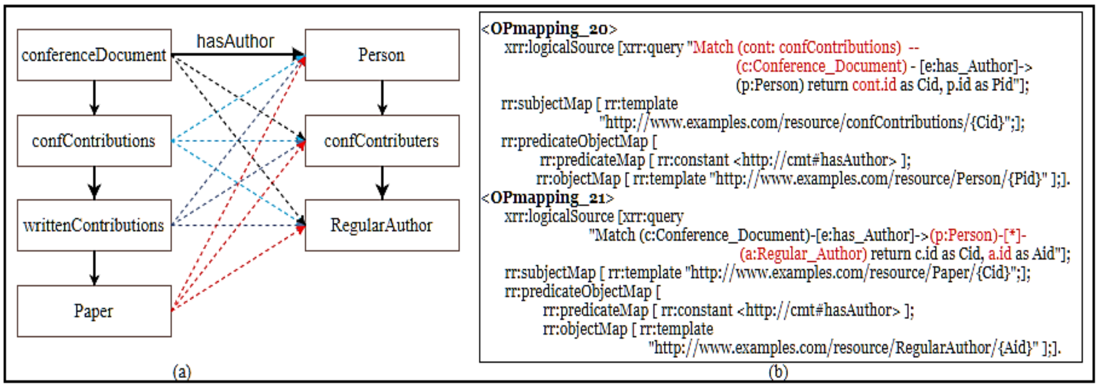
FIGURE 6. A fragment of the generated xR2RML mappings.

For each object property correspondence, an xR2RML *rr:logicalSource* includes a Cypher query that matches the edge type between the source and target nodes. An xR2RML *rr:subjectMap* includes the source node's key property appended to the URI of its aligned class instance. An xR2RML *rr:predicateMap* includes the aligned object property name, and an xR2RML *rr:objectMap* includes the target node's key property appended to the URI of its aligned class instance. Fig.6 shows a fragment of xR2RML mappings generated for the alignments from Fig.4. TripleMap ⟨***CLmapping_1***⟩ represents a class aligned to a PG node. TripleMap ⟨**CLmapping_3**⟩ represents a class aligned to a PG node property. TripleMap ⟨**DPmapping_10**⟩ represents an edge property (comment) aligned to property (review) of ontology class (Review). TripleMap ⟨**OPmapping_5**⟩ represents an edge aligned to an inverse object property.

2) PROPERTY INFERENCE

This step refers to the inheritance of data/object properties from super classes in the domain ontology to all their subclasses downwards the class hierarchy. This is applied only on class hierarchies that has corresponding hierarchies in the putative ontology inferred during the ontology learning step. xR2RML mappings are then generated for the inherited properties. This step improves the quality of the generated mappings, by covering a broad range of RDF triples that might be issued in SPARQL query. A new triple map is generated for each subsumption relationship in the class hierarchy of an ontology property's domain (or range) class. **For the hierarchy of a data property's domain class**, the xR2RML *rr:logicalSource* includes a Cypher query that matches one edge type. The edge connects the two nodes aligned to the property's domain class (superclass) and its subclass in the hierarchy. The query returns the key property of the node aligned to the subclass and the value of the property aligned to the super class's data property. An xR2RML *rr:subjectMap* includes the key property of the node aligned to the property's domain subclass appended to the URI of the subclass instance. An xR2RML *rr:predicateMap* includes the aligned data property name. An xR2RML *rr:objectMap* includes the node property name.

For the hierarchy of an object property's domain class, the xR2RML *rr:logicalSource* includes a Cypher query matching two edge types. The first edge type is between the two nodes aligned to the property's domain class (superclass) and its subclass in the hierarchy. The second edge type is between the two nodes aligned to the property's domain class (superclass) and property's range class. An xR2RML *rr:subjectMap* includes the key property of the node aligned to the property's domain subclass appended to the URI of the subclass instance. An xR2RML *rr:predicateMap* includes the aligned object property name. An xR2RML *rr:objectMap* includes the key property of the node aligned to the property's range class appended to the URI of this class instance. **For the hierarchy of an object property's range class**, the xR2RML *rr:subjectMap* includes the key property of the node aligned to the property's domain class appended to the URI of this instance. the xR2RML *rr:objectMap* includes the key property of the node aligned to the property's range subclass appended to the URI of the subclass instance. As shown in the example of Fig. 7, class "*conferenceDocument*" has object property "*hasAuthor*" to class " *Person*". Inference (dotted lines in Fig.7 (a)) results in adding xR2RML mappings (Fig.7 (b)) for the same object property between all subclasses downwards the "*conferenceDocument*" and "*Person*" class hierarchies.



```
<OPmapping_20>
   xrr:logicalSource [xrr:query "Match (cont: confContributions) --
                                 (c:Conference_Document) - [e:has_Author]->
                                 (p:Person) return cont.id as Cid, p.id as Pid"];
  rr:subjectMap [ rr:template
                 "http://www.examples.com/resource/confContributions/{Cid}";];
  rr:predicateObjectMap [
        rr:predicateMap [ rr:constant <http://cmt#hasAuthor> ];
       rr:objectMap [ rr:template "http://www.examples.com/resource/Person/{Pid}" ];].
<OPmapping_21>
   xrr:logicalSource [xrr:query
              "Match (c:Conference_Document)-[e:has_Author]->(p:Person)-[*]-
                                 (a:Regular_Author) return c.id as Cid, a.id as Aid"];
   rr:subjectMap [ rr:template "http://www.examples.com/resource/Paper/{Cid}";];
   rr:predicateObjectMap [
        rr:predicateMap [ rr:constant <http://cmt#hasAuthor> ];
        rr:objectMap [ rr:template
                      "http://www.examples.com/resource/RegularAuthor/{Aid}" ];].
```

FIGURE 7. Example of property inference during mapping generation.

V. EXPERIMENTAL EVALUATION

In this section, experiments are conducted to measure the effectiveness of mappings generated by the proposed system from a given PG to an existing domain ontology. Mapping effectiveness refers to the ability of mappings to properly support query answering in the mapped domain compared to answers retrieved from the original domain. In relational-to-ontology mapping context, a comprehensive benchmark suite, RODI, is released to evaluate R2RML mappings generated from relational databases [39], [40]. The suite consists of a broad range of real-world benchmarking scenarios. Each scenario includes a database, a target ontology, and SQL-SPARQL query pairs that evaluate different mapping challenges. However, regarding property graphs, no comparable approach or benchmark exists, to the best of our knowledge, to evaluate the proposed approach. Therefore, RODI default scenarios are adopted in this paper to evaluate mappings from property graph perspective. This is because RODI is considered the most comprehensive evaluation methodology that test for many requirements such as structural heterogeneity, terminological heterogeneity (naming conflicts), inter-model mismatch, etc.

A. DESCRIPTION OF DATASETS

In this paper, eight RODI default scenarios were selected for the evaluation from property graph perspective. Selected scenarios represent the conference domain through three different conference ontologies (cmt, conf, sigkdd) that vary in size, modeling style and expressiveness. Each ontology is associated with a set of relational database instances that are modeled to fit into specific mapping challenge. These databases are imported into property graph database engine using neo4j ETL tool[1] as shown in Table 3. The transformed PG schema and the complete query set for each scenario are found at [41].

Modeling patterns of the imported graphs are modified to fit into different mapping challenges related to property graphs according to the following rules:

[1] https://neo4j.com/developer/neo4j-etl/

TABLE 3. Description of eight scenarios used in the experiment from the conference domain.

Mapping Challenge	Scenario	#Ontology classes	#PG nodes
Adjusted Naming	cmt_renamed	32	31
	conf_renamed	67	59
	sigkdd_renamed	51	49
Restructured Hierarchies	cmt_structured	32	14
	conf_structured	67	24
	sigkdd_structured	51	12
Combined Case	cmt_mixed	32	14
	sigkdd_mixed	51	12

1. Two tables with one-to-one PK/FK relationship are converted into two nodes with an edge in between. This modeling pattern tests for [n:1 class-to-node without categorical property] class hierarchy challenge.
2. A table that has two foreign keys is considered a junction table (that breaks M-N relation) and is converted into an edge with edge properties. This modeling pattern tests for [edge property-to-N ary relation] structural hierarchy challenge.
3. A table that has more than two foreign keys is converted into a node with multiple edges. This modeling pattern tests for [1:1 class-to-edge] class hierarchy challenge.
4. A categorical column per table is converted into a categorical property per node. This modeling pattern tests for [n:1 class-to-node with categorical property] class hierarchy challenge.

B. EXPERIMENTAL QUERIES

Each scenario adopted from RODI was originally associated with SPARQL-SQL query pairs. In this paper, SPARQL queries provided for each scenario are manually translated into Cypher queries considering various graph patterns and paths. The following example shows a query pair from the cmt_renamed scenario that tests variable-length pattern

TABLE 4. Category tags used in SPARQL-Cypher query pairs for each mapping challenge.

Mapping Challenge	Category Tag
Matching class	class
Matching data type property	prop
Matching object property	link
1-1 node-class/edge-class match	1-1
n-1 class-node with categorical property	n-1
1-n class-node with categorical label	superclass
Finding the data value in the same node matched with the related class (data property match)	in-node
Finding the data value in a node other than the one matched with the related class (data property match)	other-node
Finding 1-length or 2-length edge(s) between two nodes corresponding to related classes (object property match)	1-hop, 2-hops
Finding n-length edges (n>2) between two nodes corresponding to related classes (object property match)	n-hops
Additional tag for all queries that are tagged n-hop, with any n > 1 (variable-length pattern matching)	X-hops
Matching composite edge to an object property in corresponding N-ary relation	OP-nary
Matching edge property to a data property in corresponding N-ary relation	DP-nary

matching. In this example, Cypher pattern 1 describes a graph of four nodes and three relationships, all in one path (a path of length 3). Cypher pattern 2 shows another pattern for the same query by identifying the four nodes that should be traversed in that path:

name = Q46 (PCs ⟨-⟩ Persons)
Cypher(Pattern 1) = match(progCom: program
_committees) -[∗3]->
(p:persons) return count (*)
Cypher (Pattern 2) = match(progCom: program
_committees)
-[ ] - (pcMem:pc_members)
-[ ]- (conMem:conf_members
-[ ] - (p:persons) return
count (*)
SPARQL =
prefix rdf: ⟨http://www.w3.org/1999/02/22-rdf
-syntax-ns#⟩
prefix: ⟨http://cmt#⟩ \n\
SELECT (COUNT(∗) AS ?cnt) \n\
WHERE {?c rdf:type:ProgramCommittee;
:hasProgramCommitteeMember ?p.
?p rdf:type:Person}
categories = link, X-hops, 3 hops

Each query in the eight scenarios evaluates specific PG-to-ontology mapping challenge(s) categorized as shown in Table 4. The number of queries tested for each mapping challenge along with each scenario is shown in Table 5.

TABLE 5. Number of queries per challenge category for eight scenarios.

Scenario / Category Tag	cmt_renamed	cmt_structured	cmt_mixed	conf_renamed	conf_structured	sigkdd_renamed	sigkdd_structured	sigkdd_mixed	Total per tag
class,1-1,superclass	2	0	0	2	0	2	0	0	**6**
class,1-1	10	7	7	14	8	13	7	7	**73**
class,n-1	0	5	4	0	8	0	8	6	**31**
prop, in-node	7	11	9	10	13	8	9	8	**75**
prop, other-node	4	0	0	5	2	2	1	1	**15**
OP-nary	0	0	3	0	0	0	0	2	**5**
DP-nary	0	0	3	0	0	0	0	1	**4**
link,1-hop	1	5	2	2	4	3	4	2	**23**
link,X-hops,2-hops	3	0	0	1	3	1	0	0	**8**
link,X-hops,3-hops	1	1	0	1	1	0	0	0	**4**
link,X-hops,hops>=4	1	0	0	4	0	0	0	0	**5**
Total Queries per scenario:	**29**	**29**	**28**	**39**	**39**	**29**	**29**	**27**	**249**

C. EXPERIMENTAL METHODOLOGY

Experiment encompasses eight scenarios, each one constitutes of a property graph database, a domain ontology, and a set of SPARQL-Cypher query pairs. For each test, input will be xR2RML mappings generated from the proposed ProGOMap system. Evaluation starts by executing rr:logicalSource queries from these mappings over the Neo4j PG database [42].Then, PG data results are materialized into A-Box facts in Triple store (Sesame) together with T-Box (axioms) of the domain ontology. Finally, query pairs of the scenario are evaluated on the PG database and on the triple store to measure the accuracy of the produced results.

D. EVALUATION METRICS

A per-query precision and recall metrics are used to measure the correctness and completeness of query results, respectively. Then F-measure is calculated as the mean of precision and recall values for each single query test in each RODI scenario. In addition, the average of F-measure values for query tests is calculated to obtain the overall score for each scenario. Given a single SPARQL-Cypher query pair, the precision, recall and F-measure scores are calculated as follows [39]:

$$\text{Precision} = \frac{|s\_res| - |unmatched(s\_res)|}{|s\_res|} \tag{2}$$

$$\text{Recall} = \frac{|c\_ref| - |unmatched(c\_ref)|}{|c\_ref|} \tag{3}$$

$$\text{F-measure} = 2 \times \frac{|\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

where *s_res, c_ref* are the result sets returned by SPARQL query and Cypher query, respectively. unmatched(s_res), unmatched(c_ref) are tuples that could not be matched in the SPARQL results and Cypher reference results, respectively.

E. RESULTS AND DISCUSSION

Table 6 shows the overall average scores calculated based on per-test F-measures in eight default scenarios. All scenarios perform best in the "adjusted renaming" challenge because their PG and ontology modelling patterns are very close. Scores for all scenarios in the "structured hierarchies" are more than 0.5, which indicates the effectiveness of the proposed algorithms for class hierarchy inference and object property alignment.

TABLE 6. Overall scores in eight default scenarios (calculated as the average of per-test F-measure scores).

Mapping Challenge	Scenario	Score (AVG)
Adjusted Naming	cmt_renamed	0.84
	conf_renamed	0.90
	sigkdd_renamed	0.97
Restructured Hierarchies	cmt_structured	0.62
	conf_structured	0.78
	sigkdd_structured	0.67
Combined Case	cmt_mixed	0.71
	sigkdd_mixed	0.81

The relative performance of the proposed ProGOMap system is further drilled down into four main categories. These categories test mapping effectiveness of class instances, datatype properties, object properties and n-ary relationships which are identified by class, prop, link, OP/DP-nary tag IDs of mapping challenges, respectively.

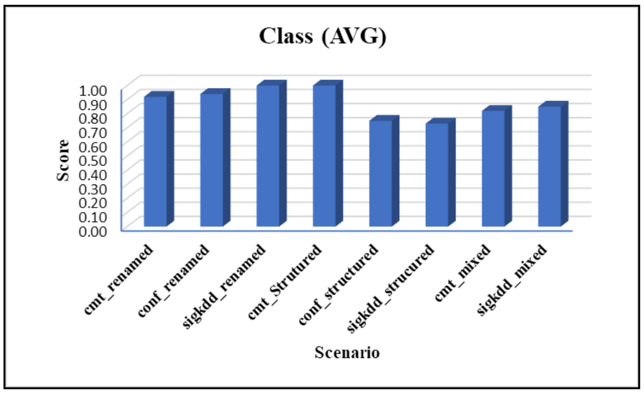


FIGURE 8. Score break-down for queries with "class" tagID for eight scenarios.

In Fig.8, all scenarios succeeded to easily map PG data to class types (All scores > 0.70). Scenarios for conf and sigkdd ontologies perform better in the "adjusted naming" challenge than in "restructured hierarchy" challenge. However, cmt_renamed scenario failed to map the abbreviated PG node (PC_Members) to the ontology class (Program_Committee_Members). Therefore, the scenario of cmt_structured outperforms the renamed one for class mapping.

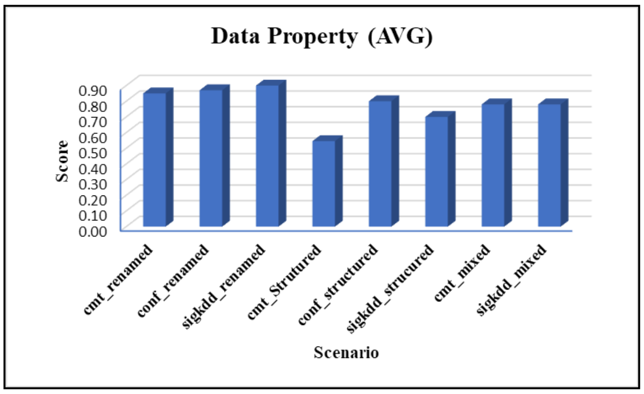


FIGURE 9. Score break-down for queries with "prop" tagID for eight scenarios.

In Fig.9, most scenarios successfully answered queries about data properties linked to classes/nodes other than the queried ones because of applying property inference during mapping generation (most scores $>= 0.70$). However, cmt_structured scenario received the lowest score in mapping data properties. This is because it failed to find mappings for ontology subclassses: *paperFullVersion* and *paperAbstract*. They are represented in PG as numeric values in Type property of Paper node (categorical property). Therefore, SPARQL queries about data properties for those subclasses did not return answers, causing a negative impact on the overall score.

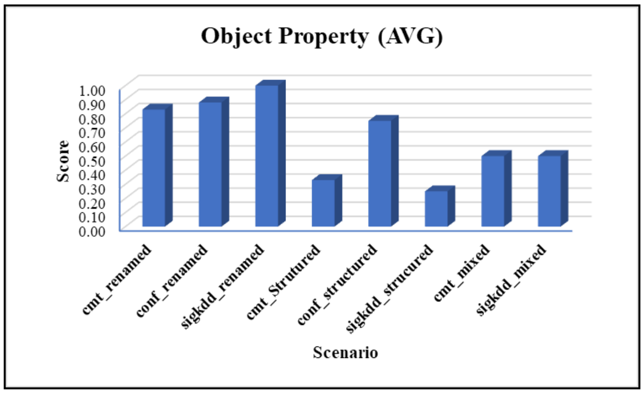
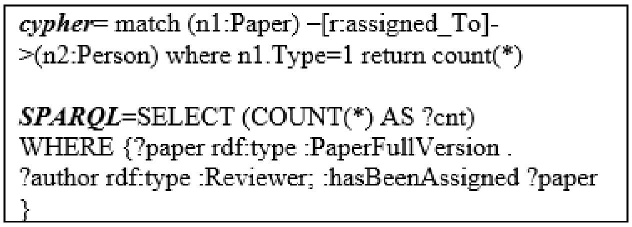


FIGURE 10. Score break-down for queries with "link" tagID for eight scenarios.



```
cypher= match (n1:Paper) –[r:assigned_To]->(n2:Person) where n1.Type=1 return count(*)

SPARQL=SELECT (COUNT(*) AS ?cnt)
WHERE {?paper rdf:type :PaperFullVersion .
?author rdf:type :Reviewer; :hasBeenAssigned ?paper
}
```

FIGURE 11. Cypher_SPARQL query pair example on numerical categorical properties.

In Fig.10, cmt_structured and sigkdd_structured scenarios received the worst scores for mapping object properties. This originates from failure in inferring class hierarchies from categorical properties that include numeric values as in the PaperType property in the example of Fig.11. Therefore, all SPARQL queries about object properties from/to those unmapped subclasses return no results. Overall, (n-1 categorical property) mapping challenge has a great effect on the success of mapping generation process.

Finally, Fig.12 shows scores of mapping PG edge property to ontology n-ary relations. In, sigkdd_mixed scenario, the "*submits*" PG edge between "*Person*" and " *document*" PG nodes has two date properties. This edge is correctly mapped to the following ontology ternary relationship:

- An object property "*notification_until*" between "*Author*" and "*notificationDeadline*" classes.
- An object property "*submit_until*" between "*Document*" and "*submissionDeadline*" classes.
- A data property "*date*" for the "*deadline*" class.

This ternary mapping is found only when applying class hierarchy traversal on the domain and range classes of the object properties. In this case, classes "*notificationDeadline*" and "*submissionDeadline*" are matched with the base class "*Deadline*" that represents the head class of the mapped ternary relationship. Also, the data property "*date*" is mapped to the "*notificationDate*" and '*submissionDate*" properties of the " *submits*" PG edge with two different triple maps; one for each property.

The score in the cmt_mixed scenario decreases because there exist two different edges, each with additional properties. n-ary relation is found for one edge but missed for the other edge due to data type mismatch between the head class data property and the edge property.

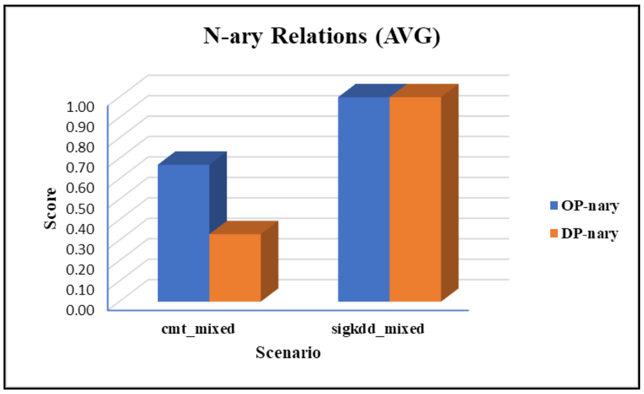


FIGURE 12. Score break-down for queries with "OP/DP-nary" tagID for eight scenarios.

TABLE 7. F-measures averaged over mapping challenge categories (tag ID) for each scenario.

Scenario / Category Tag	cmt_renamed	cmt_structured	cmt_mixed	conference_renamed	conference_structured	sigkdd_renamed	sigkdd_structured	sigkdd_mixed	Overall Score per category
class (AVG)	0.92	1.00	0.82	0.94	0.75	1.00	0.73	0.85	**0.87**
n-1 (AVG)	-	0.6	0.75		0.63	-	0.5	0.67	**0.61**
1-1 (AVG)	0.92	1.00	0.89	0.94	0.89	1.00	1.00	1.00	**0.95**
superclass (AVG)	1.00	-	-	1.00	-	1.00	-	-	**1.00**
prop (AVG)	0.85	0.55	0.78	0.87	0.80	0.90	0.70	0.78	**0.73**
in-node (AVG)	1.00	0.55	0.78	0.90	0.85	0.99	0.78	0.87	**0.83**
other-node (AVG)	0.71	-	-	0.80	0.50	0.50	NaN	NaN	**0.59**
link (AVG)	0.83	0.33	0.50	0.88	0.75	1.00	0.25	0.50	**0.68**
X-hops (AVG)	0.60	NaN	-	0.80	0.80	1.00	-	-	**0.71**
hops->=4 (AVG)	NaN	-	-	0.75	-	-	-	-	**0.60**
3-hops (AVG)	NaN	NaN	-	1.00	1.00	-	-	-	**0.5**
2-hops (AVG)	1.00	-	-	1.00	0.66	1.00			**0.87**
1-hop (AVG)	1.00	0.40	0.50	1.00	0.75	1.00	0.25	0.50	**0.61**
OP-nary (AVG)	-	-	0.67	-	-	-	-	1	**0.80**
DP-nary (AVG)	-	-	0.33	-	-	-	-	1	**0.50**
ProGOMap AVG Score									**0.79**

A more detailed information about F-measure scores averaged over mapping challenge categories (tag ID) for each scenario is provided in Table 7. The overall average score for mapping an ontology class to a property graph node/property is 0.87. Algorithm 1 managed to find classes from categorical properties of PG nodes with overall score of 0.61. The overall score of mapping each ontology class to a single PG node is 0.95. This proves the effectiveness of string similarity measures to address the terminological heterogeneity between PG and ontology models. The overall score for mapping data properties is 0.73. Applying property inference during mapping generation succeeded in mapping data properties from other classes with an average score of 0.59. Mapping object properties to a variable length graph pattern (x-hops) achieved a score of 0.71. This indicates the significance of applying class hierarchy traversal to the domain/range classes of the object properties during the alignment between the putative and domain ontologies.

VI. CONCLUSION

This paper presents a novel approach that automatically generates mappings from property graph (PG) data sources to ontologies by addressing various mapping challenges. The mapping process comprises three main steps: (i) generating putative ontology from PG schema, (ii) aligning putative ontology to a target domain ontology, (iii) generating xR2RML mappings from the resulting alignments. Experiments to measure mapping effectiveness were conducted on eight datasets in the conference domain. Each dataset is designed to assess the quality of a specific mapping challenge. Terminological heterogeneity challenge is addressed by applying various string similarity measures between the two models. The overall mapping scores achieved in this challenge for mapping classes, data properties, and object properties are 0.96, 0.73, and 0.90, respectively. The data property score is the least because some data properties in the domain ontology are modeled as nodes in the property graph (e.g., email). Also, a class label ‘‘conference_www’’ could not be mapped to a node label ‘‘website’’, and in turn their properties are not mapped. Structural heterogeneity challenges are addressed by proposing an ontology learning algorithm to infer class hierarchies from the PG model, and an ontology alignment algorithm for object properties considering various criteria. The overall mapping scores achieved in this challenge for mapping classes, data properties, and object properties are 0.81, 0.70, and 0.50, respectively. The drop in these scores arises from having many categorical properties in PG nodes that have numerical values which cannot be used to infer class hierarchies during ontology learning/alignment steps. Further investigation is required for mapping properties with this challenge.

In the future, it is planned to conduct experiments for cross-matching scenarios (e.g., mapping cmt with sigkdd). In these scenarios, the level of semantic heterogeneity is higher than default ones because it involves mapping a schema derived from one ontology to another different ontology in the same domain. Further extension includes mapping generation for other types of NoSQL databases (e.g., document database), which requires additional ontology learning methods to infer class hierarchies from them.

REFERENCES

[1] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, ‘‘Linking data to ontologies,’’ in *Journal on Data Semantics X*. Berlin, Germany: Springer, 2008, pp. 133–173, doi: 10.1007/978-3-540-77688-8_5.

[2] E. Kharlamov, D. Hovland, M. G. Skjæveland, D. Bilidas, E. Jiménez-Ruiz, G. Xiao, A. Soylu, D. Lanti, M. Rezk, D. Zheleznyakov, M. Giese, H. Lie, Y. Ioannidis, Y. Kotidis, M. Koubarakis, and A. Waaler, ‘‘Ontology based data access in statoil,’’ *J. Web Semantics*, vol. 44, pp. 3–36, May 2017.

[3] G. Fusco and L. Aversano, ‘‘An approach for semantic integration of heterogeneous data sources,’’ *PeerJ Comput. Sci.*, vol. 6, p. e254, Mar. 2020.

[4] S. Das, S. Sundara, and R. Cyganiak. (2012). *R2RML: RDB to RDF Mapping Language*. Accessed: Jan. 31, 2019. [Online]. Available: https://www.w3.org/TR/r2rml/http://www.w3.org/TR/2012/REC-r2rml-20120927/Latestversion:http://www.w3.org/TR/r2rml/

[5] F. Michel, L. Djimenou, C. Faron-Zucker, and J. Montagnat, ‘‘Translation of relational and non-relational databases into RDF with xR2RML,’’ in *Proc. 11th Int. Conf. Web Inf. Syst. Technol. (WEBIST)*, 2015, pp. 443–454.

[6] J. F. Sequeda, A. Garcia-Castro, O. Corcho, S. H. Tirmizi, and D. P. Miranker, ‘‘Overcoming database heterogeneity to facilitate social networks: The Colombian displaced population as a case study,’’ in *Proc. 18th Int. Conf. World Wide Web Ibero-Amer. Track*, 2009.

[7] L. F. de Medeiros, F. Priyatna, and O. Corcho, ‘‘MIRROR: Automatic R2RML mapping generation from relational databases,’’ in *Proc. Int. Conf. Web Eng.*, 2015, pp. 326–343.

[8] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, A. Nikolov, A. Schwarte, C. Heupel, and T. Kraska, ‘‘Incmap: A journey towards ontology-based data integration,’’ *Lect. Notes Informat. Proc. Gesellschaft Inform.*, vol. 265, pp. 145–164, 2017.

[9] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. Skjæveland, E. Thorstensen, and J. Mora, ‘‘BootOX: Practical mapping of RDBs to OWL 2,’’ in *Proc. Int. Semantic Web Conf.*, 2015, pp. 113–132.

[10] Á. Sicilia and G. Nemirovski, ‘‘AutoMap4OBDA: Automated generation of R2RML mappings for OBDA,’’ in *Proc. Eur. Knowl. Acquisition Workshop*, 2016, pp. 577–592.

[11] S. N. Mathur, D. O’Sullivan, and R. Brennan, ‘‘Milan: Automatic generation of R2RML mappings,’’ in *Proc. 26th Irish Conf. Artif. Intell. Cogn. Sci.*, Dublin, Ireland, 2018.

[12] H. Li and Q. Sima, ‘‘Parallel mining of OWL 2 EL ontology from large linked datasets,’’ *Knowl.-Based Syst.*, vol. 84, pp. 10–17, Aug. 2015, doi: 10.1016/j.knosys.2015.03.023.

[13] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo, ‘‘Mapping RDF knowledge bases using exchange samples,’’ *Knowl.-Based Syst.*, vol. 93, pp. 47–66, Feb. 2016, doi: 10.1016/j.knosys.2015.11.001.

[14] D. Oliveira, R. Sahay, and M. d’Aquin, ‘‘Leveraging ontologies for knowledge graph schemas,’’ in *Proc. ESWC Workshop KGB*, 2019.

[15] S. Das and M. Perry, ‘‘A tale of two graphs: Property graphs as RDF in Oracle,’’ in *Proc. EDBT*, 2014, pp. 762–773.

[16] D. Tomaszuk, R. Angles, and H. Thakkar, ‘‘PGO: Describing property graphs in RDF,’’ *IEEE Access*, vol. 8, pp. 118355–118369, 2020.

[17] H. Chiba, R. Yamanaka, and S. Matsumoto, ‘‘G2GML: Graph to graph mapping language for bridging RDF and property graphs,’’ in *Proc. Int. Semantic Web Conf.*, 2020, pp. 160–175.

[18] R. Angles, H. Thakkar, and D. Tomaszuk, ‘‘Mapping RDF databases to property graph databases,’’ *IEEE Access*, vol. 8, pp. 86091–86110, 2020.

[19] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. Newton, MA, USA: O’Reilly Media, Inc., 2013.

[20] G. Klyne and J. Carroll. (2004). *Resource Description Framework (RDF) Concepts and Abstract Syntax*. [Online]. Available: https://www.w3.org/TR/2004/REC-115-concepts-20040210/

[21] E. Harris, S. Seaborne, A. Prud’hommeaux. (2013). *SPARQL 1.1 Query Language*. W3C Recommendation. [Online]. Available: http://www.w3.org/TR/sparql11-query/

[22] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, ‘‘Cypher: An evolving query language for property graphs,’’ in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 1433–1445.

[23] R. Angles, H. Thakkar, and D. Tomaszuk, "RDF and property graphs interoperability: Status and issues," in *Proc. AMW*, Paraguay, 2019.

[24] J. Euzenat and P. Shvaiko, "The matching problem," in *Ontology Matching*. Berlin, Germany: Springer, 2013, pp. 25–54.

[25] (2019). *W3C Workshop Minutes*. W3C Workshop on web Standardization for Graph Data. Berlin, Germany. [Online]. Available: https://www.w3.org/Data/events/data-ws-2019/minutes.html

[26] R. Angles, "The property graph database model," in *Proc. AMW*, Cali, Colombia, 2018.

[27] P. Pauwels, S. Zhang, and Y.-C. Lee, "Semantic web technologies in AEC industry: A literature overview," *Autom. Construct.*, vol. 73, pp. 145–165, Jan. 2017.

[28] B. Motik, I. Horrocks, and U. Sattler, "Bridging the gap between OWL and relational databases," *J. Web Semantics*, vol. 7, no. 2, pp. 74–89, Apr. 2009.

[29] I. Comyn-Wattiau and J. Akoka, "Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 453–458.

[30] N. Noy, A. Rector, P. Hayes, and C. Welty, "Defining n-ary relations on the semantic web," W3C Work. Group Note, Apr. 2006, vol. 12, no. 4. [Online]. Available: http://www.w3.org/TR/swbp-n-aryRelations

[31] F. Cerbah, "Mining the content of relational databases to learn ontologies with deeper taxonomies," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Intell. Agent Technol.*, Dec. 2008, pp. 553–557.

[32] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 2012.

[33] F. Cerbah, "Learning ontologies with deep class hierarchies by mining the content of relational databases," in *Advances in Knowledge Discovery and Management*, F. Guillet, G. Ritschard, D. A. Zighed, and H. Briand, Eds. Berlin, Germany: Springer, 2010, pp. 271–286.

[34] P. Shvaiko and J. Euzenat, "Ontology matching: State of the art and future challenges," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 158–176, Jan. 2011.

[35] P. Ochieng and S. Kyanda, "Large-scale ontology matching: State-of-the-art analysis," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–35, Sep. 2018.

[36] E. Jiménez-Ruiz and B. C. Grau, "Logic-based and scalable ontology matching," in *The Semantic Web*. Berlin, Germany: Springer, 2011, pp. 273–288.

[37] M. Cheatham and P. Hitzler, "String similarity metrics for ontology alignment," in *Proc. Int. semantic Web Conf.*, 2013, pp. 294–309.

[38] M. Cheatham and P. Hitzler, "The properties of property alignment," in *OM*. CEUR-WS.org, 2014, pp. 13–24.

[39] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, W. May, A. Nikolov, A. S. Bastinos, M. G. Skjæveland, A. Solimando, M. Taheriyan, C. Heupel, and I. Horrocks, "RODI: Benchmarking relational-to-ontology mapping generation quality," *Semantic Web*, vol. 9, no. 1, pp. 25–52, Nov. 2017.

[40] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, W. May, D. Ritze, M. G. Skjæveland, A. Solimando, and E. Kharlamov, "RODI: A benchmark for automatic mapping generation in relational-to-ontology data integration," in *Proc. Eur. Semantic Web Conf.*, 2015, pp. 21–37.

[41] (2020). *Schema and Test Queries for the ProGOMap Experimental Datasets*. [Online]. Available: https://drive.google.com/drive/folders/15JQk0yZKucvJ2eadKLz-9Ch0nLyb9LEs?usp=sharing

[42] *Neo4j Graph Platform—The Leader in Graph Databases*. Accessed: Mar. 14, 2019. [Online]. Available: https://neo4j.com/

NAGLAA FATHY received the B.Sc. and M.Sc. degrees in information systems from the Faculty of Computer and Information Sciences (FCIS), Ain Shams University (ASU), Egypt, in 2006 and 2012, respectively, where she is currently pursuing the Ph.D. degree. Her master's was about personalized information retrieval considering users' preferences. She is an Assistant Lecturer with FCIS, ASU. Her research interests include information retrieval, information integration, semantic web, and ontology engineering, and social networking.

WALAA GAD received the B.Sc. and M.Sc. degrees in computers and information sciences from Ain Shams University, Cairo, Egypt, in 2000 and 2005, respectively, and the Ph.D. degree in computers and information sciences, in 2010. Her master's was about designing and planning a network model in the presence of obstacles using clustering around medoids techniques. She was a Ph.D. Student with the Pattern and Machine Intelligence (PAMI) Group, Faculty of Electrical and Computer Engineering, University of Waterloo, Canada. The dissertation title is "Text Clustering Based on Semantic Measures." The work was done jointly between the Faculty of Computers and Information Sciences, Ain Shams University, and the University of Waterloo. She is currently an Associate Professor with the Faculty of Computers and Information Sciences. She is the author of several publications. Her current research interests include data science, semantic web and machine learning, data warehouse, and big data analytics.

NAGWA BADR received the M.S. degree in computer science and the Ph.D. degree in software engineering and distributed systems from Liverpool John Moores University, U.K., in 1996 and 2003, respectively. She had done postdoctoral studies with Glasgow University, U.K. She is currently a Professor and the Dean of the Faculty of Computer and Information Sciences (FCIS), Ain Shams University (ASU). For the last few years, she is the Head of committee that contributed to research projects funded by national and international grants in information systems, bioinformatics, business analytics, and health informatics. Her research interests include software engineering, cloud computing, big data analytics, social networking, Arabic search engines, and bioinformatics.

MOHAMED HASHEM was the Head of the Information Systems Department, FCIS. He was the Vice-Dean of education and student affairs at the Faculty of Computer and Information Sciences (FCIS). He is currently a Professor in information systems with FCIS, Ain Shams University, Egypt. His research interests include modeling and simulation of computer networks, computer security, and data management.