Extracting data models from background knowledge graphs

Daniela Oliveira, Mathieu D'aquin

Extracting Data Models from Background Knowledge Graphs

Daniela Oliveira[b], Mathieu d'Aquin[a,*]

[a]LORIA, Université de Lorraine, Nancy, France
[b]LASIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, Portugal

ARTICLE INFO

Keywords:
Knowledge Graphs
Ontologies
Data Modelling

Abstract

Knowledge Graphs have emerged as a core technology to aggregate and publish knowledge on the Web. However, integrating knowledge from different sources, not specifically designed to be interoperable, is not a trivial task. Finding the right ontologies to model a dataset is a challenge since several valid data models exist and there is no clear agreement between them. In this paper, we propose to facilitate the selection of a data model with the RICDaM (Recommending Interoperable and Consistent Data Models) framework. RICDaM generates and ranks candidates that match entity types and properties in an input dataset. These candidates are obtained by aggregating freely available domain RDF datasets in a knowledge graph and then enriching the relationships between the graph's entities. The entity type and object property candidates are obtained by exploiting the instances and structure of this knowledge graph to compute a score that considers both the accuracy and interoperability of the candidates. Datatype properties are predicted with a random forest model, trained on the knowledge graph properties and their values, so to make predictions on candidate properties and rank them according to different measures. We present experiments using multiple datasets from the library domain as a use case and show that our methodology can produce meaningful candidate data models, adaptable to specific scenarios and needs.

1. Introduction

In recent years, the concept of Knowledge Graph (KG) has become more prominent and has been adopted to describe graphs that aggregate different sources of knowledge [9] (e.g. DBPedia [21]). KGs are enabled by a set of best practices which adhere to the Resource Description Framework (RDF)[1] data model, following the Linked Data[2] principles.

Ontologies, defined as "*an explicit specification of a conceptualization*" [12], are typically used to model KGs, ensuring consistency between data sources by providing a schema that formally represents and precisely defines entities and their relationships [19] (see Suplementary Material 1 for a more detailed conceptual background). A key barrier for a data publisher, however, is to find the ontologies that best fit their data, but also enable integration with existing datasets in the domain.

As an example representing a simplified version of the use case described in this paper, we can consider a library wanting to make their catalogue of books available as a knowledge graph. In choosing what ontologies to use, and what classes and properties from those ontologies to specifically apply, this library will take into consideration how aligned those ontologies are with the data content to be represented, but also how some choices might lead to greater interoperability with other libraries, if the considered elements are commonly used by those libraries, and how consistent the overall result will be. Our objective here is to provide a framework to support such a process by explicitly aligning the content of the data to existing, overlapping knowledge graphs, enabling us to assess content alignment (through similarity), potential for interoperability (through direct or indirect occurrence) and consistency (through verifying the systematic use of classes and properties in different situations). Therefore, if in our example the framework finds that, among overlapping knowledge graphs, two different classes can be used to represent books, the class *Book* from one ontology and the class *LiteraryWork* from another, the user might want to choose the first one for being more semantically aligned with their data. They could however, based on the information provided by the framework, choose the second class for being more commonly used by or linked to existing knowledge graphs and therefore better in supporting interoperabiliy. Once one is chosen, other aspects of the data might be described differently so to keep consistent with this choice.

A survey conducted over several years with Linked Data providers found that the third most common barrier to publishing Linked Data was "*selecting appropriate ontologies to represent our data*" [39, 40]. Similarly, another survey [24] found that data publishers in the libraries, archives and museums domains consider that barriers to publishing Linked Data, included being time-consuming, having a steep learning curve, difficulty in using linked data tools or no adequate tools being available, and difficulty in establishing links. Therefore, when data publishers find the process of publishing Linked Data more demanding than simply maintaining their current publishing process, they end up either not publishing linked data at all or they do not go through all the effort it requires to provide fully and directly interoperable data models with published data sources. In this second case, it can lead to publishers creating local, isolated ontologies or extending upper-level ontologies to meet their requirements, therefore not enabling integration with other existing datasets.

This issue has been described in several domains such as

*Corresponding author
mathieu.daquin@loria.fr (M. d'Aquin)
ORCID(s):

[1] https://www.w3.org/TR/rdf11-concepts/ (Accessed in November 2020)

[2] https://www.w3.org/DesignIssues/LinkedData.html (Accessed in May 2021)

Table 1
Entity types that describe books in different library datasets.

Library	Book entity types
British	`dcterms:BibliographicResource` & `bibo:Book` & `schema:Book`
French	`skos:Concept` | `frbr:Work`
German	`bibo:Document`
Gutenberg	`pgterms:ebook`
University	`bibframe:Work` | `bibfram:Text`
Portuguese	`edm:ProvidedCHO`
Spanish	`bne:C1003`
OpenL	`/type/work`
Institute	`Title`

life sciences [17], education [5], and library data [44]. Since we will focus our use case on the library data domain (see Section 3 for more details), Table 1 illustrates the issue in this domain. It shows how each of the libraries describes an entity corresponding to a physical book in their respective collections. Each of the existing libraries had its data model manually created and curated. Therefore, there is no clear correct choice and no standard is fully adopted in the domain. A librarian wishing to publish their catalogue online, as part of a knowledge graph, would find a hard barrier in deciding which data model to follow.

In this paper, we explore the potential of automating the selection of an ontological model for existing data and describe RICDaM - Recommending Interoperable and Consistent Data Models - a framework to ease the task of selecting the best data model for an input dataset (which can be provided in a structured or semi-structured formats). This framework produces a ranked list of candidate data models that fit the data and are interoperable with a KG of published RDF data sources in the same domain as the input data. This KG is obtained by aggregating freely available RDF datasets, extracting their underlying ontology graph, and then enriching its edges to produce a tightly connected graph. We exploit the content and graph structure of this KG to compute a score that considers the accuracy, interoperability, and consistency of the candidates. The output of the framework is the correspondence between a list of triple patterns (i.e., $\langle \texttt{domain} - \texttt{property} \twoheadrightarrow \texttt{range} \rangle$) extracted from the input dataset and a ranked list of candidate triple patterns from the KG that can be used to model that same information. These rankings are obtained by combining three scores into a single triple score. This score combination is weighted and the user has the choice to decide the weight for each score to best fit their use case or application.

Through the experiments presented here, we show that RICDaM has the potential to support data publishers in expediting the process of selecting an ontological model for their data and transforming the data into this model, by providing an overview of the data models in the domain while suggesting appropriate entity type candidates for the entities in a dataset. Without such a framework, a data publisher would need to study the domain standards, if any exists, by manually analysing existing datasets and choosing the data model that best suits their use case.

A demonstration [29] of the output of the framework for two datasets is available at `http://afel.insight-centre.org/ricdam/`. This demonstration provides some customisation options and intends to emulate a possible implementation of the framework.

We start the article by presenting some preliminary concepts and introducing some related work. Then we describe the datasets used throughout this paper and present an overview of the framework, followed by a detailed explanation of the three stages of the framework including the experiments devised to test the methodology with results and discussion. Finally, we present our conclusions, limitations, and future work.

2. Related Work

Data transformation from heterogeneous data sources to RDF is a complex task and different solutions have been proposed that focus on specific file formats or provide the flexibility to be adapted to several data publishing formats. Blomqvist et al. [20] share a list of requirements for generating RDF from heterogeneous data sources, gathered from experience and their use cases. These requirements include: (1) transform different file formats, including binary, and easily extend with new formats, (2) exploit existing RDF data sources for context, (3) easy to use by Semantic Web experts, (4) integrate well in a data engineering workflow, and (5) be flexible and maintainable. Our framework aligns well with the use cases proposed by Blomqvist et al. It fulfils the requirements or is easily extendable to fulfil them.

In the same article, Blomqvist et al. propose SPARQL-Generate, an extension of SPARQL 1.1[3] to transform data into RDF, while fulfilling the requirements delineated. SPARQL-Generate works by creating an adapted SPARQL query that maps input data to an RDF data source by answering the query.

Similarly, mapping languages are a popular solution that has long been used to map data sources to RDF by creating a set of logical rules that express the relations between entities in a data source. These rules are formalised with a vocabulary that facilitates the translation of data sources to RDF. R2RML[4] [6] was proposed as a language to express mappings between Relational Databases (RDB) and RDF datasets. The RML mapping language [7] expanded the applicability of R2RML by making the language input format-agnostic, while YARRRML [16] was proposed as a human-readable format to write these mappings. Tools and mappings for different formats have been developed (e.g. [25, 26, 13]) and the mapping languages have been applied to convert data sources in different domains (e.g. [8, 18]). Generally, mapping languages aim to create RDF data models that fit heterogeneous data sources.

However, the mappings created are either directly extracted from the data source without following a specific existing ontology or when creating the rules, an ontology or

[3] `https://www.w3.org/TR/sparql11-query`
[4] RDB to RDF Mapping Language

a set of ontologies needs to be manually chosen to map the concepts in the data sources. Our framework extracts these concepts from the existing RDF data sources, facilitating the process of choosing the most appropriate ontology or set of ontologies for a given use case. Furthermore, our solution could be integrated with mapping languages since our framework extracts the specific vocabularies from existing data. Therefore, our framework could facilitate the creation of mapping rules by suggesting a data model or candidate entity types and proprieties to jump-start the creation of the mapping rules.

When looking specifically at entity type matching, approaches have been proposed that exploit relationships between entities to match entity types [32, 38]. A more recent approach [43] uses a fixed group of ontologies (DBPedia, YAGO, and schema.org), chosen due to pre-existing mappings between them, as background knowledge to choose the best entity type to match an input. This approach, like ours, uses a search-based approach to generate candidates and applies a ranking methodology to an input unstructured text based on its context. Our approach does not take into account context since, currently, our framework does not accept unstructured text as an input and, therefore, there is no context to be extracted. Furthermore, while the authors of [43] focus on entity types only, we propose a methodology to match the whole data model of an input dataset that not only ranks candidates based on content but also focuses on maximising interoperability with existing RDF datasets.

When focusing on the schema matching problem in the alignment of object properties, it is common to find approaches that refocus the problem as a task to find relationships between individuals. RelFinder [22] is an example of this approach since they perform a search over entities in an RDF dataset to find their relationships, while other approaches use schema paths to find relationships between entity individuals through their entity types [36, 37]. Hutchison et al. [33] also find relationships between entities but, besides considering the path between entities, also takes into account external resources by measuring the co-occurrence of entities in Web documents. Due to the holistic nature of our framework, we extract object properties from the entity type candidates generated and ranked. Therefore, we do not need the datasets to be aligned *a priori.* Overall, in our framework, we consider a broader view of relationships by taking into account schemas from different datasets and providing candidates that match them. Moreover, our relationship matching strategy works not only for RDF ontologies but also for semi-structured datasets.

Looking at matching datatype properties, the focus is usually on analysing the values of the datatype properties to find matches. Hutchison et al. [34] proposed a method to match datatype property candidates using mutual information at the instance level to generate candidates and applying a genetic algorithm to create mappings between datatype properties. The authors of this work focus on finding the best matches for simple and complex datatype properties. In our work, we aim to find a broad list of candidates from several existing data models and rank them according to different measures.

The problem of generating datatype property candidates is akin to the task of tabular data interpretation, where table headers represent entity types or properties for each row, which represents an entity. The first task in tabular data annotation is identifying the type of data in each column and the relations between columns. Syed et al. [41] explore the use Wikipedia as background knowledge, while other approaches use HTML (Hypertext Markup Language) tables obtained from the Web [23, 45, 1] to interpret tabular data. Numerical values in tables are not easy to label. Therefore, approaches have been proposed to identify them [27, 47] using classification and clustering methods. Commonly, tabular data does not follow any strict data model. Therefore, data publishers provide table headers that, even though they might refer to the same concepts, are provided with different names. For instance, Chen et al. [4] provide the example of a *latitute* column *versus* the abbreviated *lat* header. The authors of this work use a supervised learning method to predict alternative or missing schema labels to integrate tabular data. In our proposed datatype property prediction approach, we also apply a Random Forest model. In addition to similar features proposed by Chen et al. [4], we also extract some features that focus on distinguishing string values, such as the number of letters, letter casing, and the number of non-alphanumeric characters. However, due to the variety of string datatypes, we do not expect the datatype model to obtain a high Precision@1, but we aim for high Precision@5 since the model is used to generate an extensive list of datatype property candidates that is further ranked by content- and interoperability-based scores.

To summarise, while work exists to address parts of the process of recommending a data model to be used for given input data sources, or similar tasks, a complete process aiming to suggest all the elements of such a data model based on user requirements regarding accuracy, interoperability and consistency does not seem to exist at this point. While we take inspiration from some of the works presented above to create components of our process, this means that, to the best of our knowledge, there is no directly comparable approach to ours.

3. Datasets

This research was motivated and guided by a library data domain use case. Several large RDF datasets exist and have been published following linked data principles in this domain and it is common to find semi-structured data in CSV and JSON formats that are related to it. We used two partial ground truths in the experiments to help evaluate the performance of the proposed methodology. The following sections detail the characteristics of each of these datasets including general statistics related to their contents.

Table 2
Summary of the library datasets used. The dashed line separates RDF (top) from non-RDF (bottom) datasets. ET = Entity Types; DP = Datatype Properties; OP = Object properties.

Source name	URL	Handle	# Entities	# ET	# DP	# OP	# Triples
British Library	https://www.bl.uk	British	17 289 195	15	34	17	51
Bibliothèque Nationale de France	https://www.bnf.fr	French	38 100 563	15	93	602	695
Deutsche Nationalbibliothek	https://www.dnb.de	German	50 340 156	20	236	187	423
Project Gutenberg	https://www.gutenberg.org	Gutenberg	856 476	7	40	30	70
James Hardiman Library	http://www.library.nuigalway.ie	University	5 236 482	43	293	74	367
Biblioteca Nacional de Portugal	http://www.bnportugal.gov.pt	Portuguese	2 437 096	2	28	0	28
Biblioteca Nacional de España	http://www.bne.es	Spanish	20 752 087	16	163	38	201
Open Library (JSON)	https://openlibrary.org	OpenL	54 678 367	15	324	5	329
DSI Library (CSV)	https://dsi.nuigalway.ie	Institute	34	N/A	N/A	0	32

3.1. Library Use Case

The library data domain is a prime example of the problem RICDaM aims to tackle. Traditionally, libraries have exposed their catalogues using Machine-Readable Cataloguing (MARC)[5] standards, which have been criticised for their restrictions in the description of relationships between entities [42, 3]. Libraries have, therefore, been shifting towards more open, reusable, and interoperable formats by exposing their catalogues in the RDF format. However, several schemas have been proposed to structure bibliographic data but none are widely adopted [44], with different libraries modelling data in distinct ways [14, 31], hindering data interchange between published datasets. Considering the different solutions, it is also challenging for the library data publisher to select the most appropriate data model when transitioning from traditional standards to RDF models.

In our experiments, we distinguish between RDF and non-RDF (CSV and JSON) datasets. Table 2 presents the datasets chosen from several publicly available online library catalogues. The Gutenberg and University datasets were chosen as the experimental use cases since they follow data model practices not completely or easily linked to the models of the other libraries. The non-RDF datasets include a JSON dataset from the Open Library and a local small-scale CSV example with books and magazines from the library of the Computing and Communications Museum of Ireland located in the Data Science Institute (DSI).

3.2. Evaluation Datasets

The goal of RiCDaM is not to find a single best data model but instead to provide recommendations among proposed data models in a certain domain. These recommendations enable a data publisher to find elements of a data model that best fit their purpose, be it specificity or interoperability. RiCDaM includes the interoperability measure which leads to the system not always proposing the candidate that, for example, is featured in existing schema matching ground truths, such as the OAEI Knowledge Graph track[6]. To the best of our knowledge, no ground truth exists that can reliably evaluate a data model recommendation in these conditions. Furthermore, since our goal is not to find a single best data model but provide recommendations within several viable options, a reliable ground truth to evaluate interoperability and consistency is not trivial to construct, if even possible. Despite these limitations, we used the OAEI Knowledge Graph track and we built a ground truth within the scope of this use case that can be used for future systems and comparisons. Hopefully, this ground truth will help with evaluating future approaches that take a similar approach to the one taken in RiCDaM.

3.2.1. DBkWik Ground Truth

For the evaluation and analyses of the components of the framework developed we use the data extracted for the DBkWik KG [15]. This data was made available for the OAEI 2020 Knowledge Graph track[7] in which the main task was developing systems capable of matching both the instances and the schema of KGs. The data was extracted from pages in the Fandom Wikifarm[8] from three Star Wars-related wikis: Star Wars (SW), Star Wars: The Old Republic (SWTOR), and Star Wars Galaxies (SWG); three Star Trek wikis: Memory Alpha (MA), Memory Beta (MB), and Star Trek Expanded Universe (STE); and two Marvel-related wikis: Marvel Database (MDB) and Marvel Cinematic Universe (MCU).

We use SWTOR and SWG as the Star Wars KG, MB and STE as the Star Trek KG, and MDB as the Marvel KG. SW and MA are used as the input data since they are the most likely to overlap with the resources in the KG data sources and, therefore, better emulate the application of our framework. Since Marvel only has two datasets, we empirically chose MCU as the input dataset. To evaluate the mappings between input and KG, we use the Ground Truths (GTs) provided for the OAEI 2019 Knowledge Graph Track. These GTs include 1-to-1 equivalence mappings of entities, entity types, and properties for each pair of files.

Table 3 shows the number of each one of these elements in the ground truth datasets, split into data that went into the KG and input. It also includes the number of mappings in the GT for each data model element. These statistics show

[5] http://www.loc.gov/marc (Accessed in April 2020)
[6] http://oaei.ontologymatching.org/2020/knowledgegraph
[7] http://oaei.ontologymatching.org/2020/knowledgegraph
[8] https://www.fandom.com

Table 3
DBkWik statistics. ET = Entity Types; DP = Datatype Properties; OP = Object properties.

Dataset	Data	# Entities	# ET	# DP	# OP	# Triples
Star Wars	KG: SWTOR + SWG	37 590	172	393	154	547
	Input: SW	335 666	273	231	483	714
	GT: SW-SWG + SW-SWTOR	2448	17	7	57	-
Star Trek	KG: MB + STE	127 149	527	268	388	656
	Input: MA	143 838	185	181	160	341
	GT: MA-MB + MA-STE	9754	19	16	50	-
Marvel	KG: MDB	1 086 734	190	101	54	155
	Input: MCU	136 500	59	49	114	163
	GT: MCU-Marvel	1641	2	3	8	-

that the ground truth is not very rich in data model elements and most of the mappings considered are between entities.

3.2.2. Gutenberg Ground Truth

In addition to the DBkWik ground truth, we created a partial ground truth for the Gutenberg dataset. This ground truth better aligns with RICDaM's use case since it has several correct options for each data model element and illustrates the need for a framework like RICDaM. We selected the Gutenberg dataset because it has a simple data model with entity types and properties that are ubiquitous to the data included in the library knowledge graph. Therefore, for this ground truth we select both entity types (`pgterms:agents` and `pgterms:ebook`), five datatype properties and `pgterms:creator` as an example of an object property, considering that this is the only object property that is easily mapped between all knowledge graph datasets. For each of these elements, we manually selected possible equivalences in the datasets of the knowledge graph. The ground truth is unranked since we do not consider any candidate better than others but consider instead their context in the graph. The Gutenberg ground truth is available at `https://figshare.com/s/cdfbd624e4e451d7ac25`[9].

4. Framework Overview

The proposed workflow, illustrated in Figure 1, has three stages: (1) **Building Background Knowledge** constructs a KG from RDF data sources, extracts the underlying ontology graph from these sources and trains a Random Forest model to predict datatype properties, (2) **Candidate Generation** produces a list of entity type and property candidates for entities and properties in an input dataset extracted from the background knowledge, and, finally, (3) **Candidate Ranking** ranks the lists of entity type and property candidates using three scoring methodologies (content, interoperability and consistency).

The requirement to apply this framework to obtain candidate ontology elements for specific data is that related and overlapping data can be found already in the form of knowledge graphs. The following sections detail each stage and illustrate their application in the library data domain where such a requirement is easily fulfilled. For the remainder of this article, we will use the prefixes in Table 1s of the Supplementary Material when referring to ontology namespaces.

[9]URL will be replaced with public DOI in a final version.

5. Building Background Knowledge

In this stage, we describe the knowledge structures we extract to support our framework, illustrated in Figure 1. These structures are built to facilitate the generation and ranking of entity type candidates and properties. The framework has the following requirements: (1) efficiently produce ranked data model candidates, (2) enable search over the literals of the RDF data sources to generate entity type candidates, (3) easily traverse the ontology graph to understand the relations of each entity type and property to rank them based on their interoperability, and (4) generate datatype property candidates from literal objects.

We consider two main sources of knowledge that can be extracted from RDF data sources: the data layer and the ontology layer. The data layer contains the entities and their relationships, and the ontology layer contains not only the entity types and properties used in the dataset but also further logical relationships between entity types that might not be featured in the dataset.

Therefore, to answer the requirements of the framework, we define the following tasks:

(1) building the KG: extracts the RDF data layer and stores it in a document store to be indexed;

(2) creating the ontology graph: extracts the ontology layer and connects it via hierarchical and logical relationships and enriches the graph with edges inferred from the data layer;

(3) extracting metadata: pre-computes the metadata maps that facilitate candidate generation and ranking;

(4) fitting the datatype property classification model: training and testing a Random Forest model to predict datatype properties.

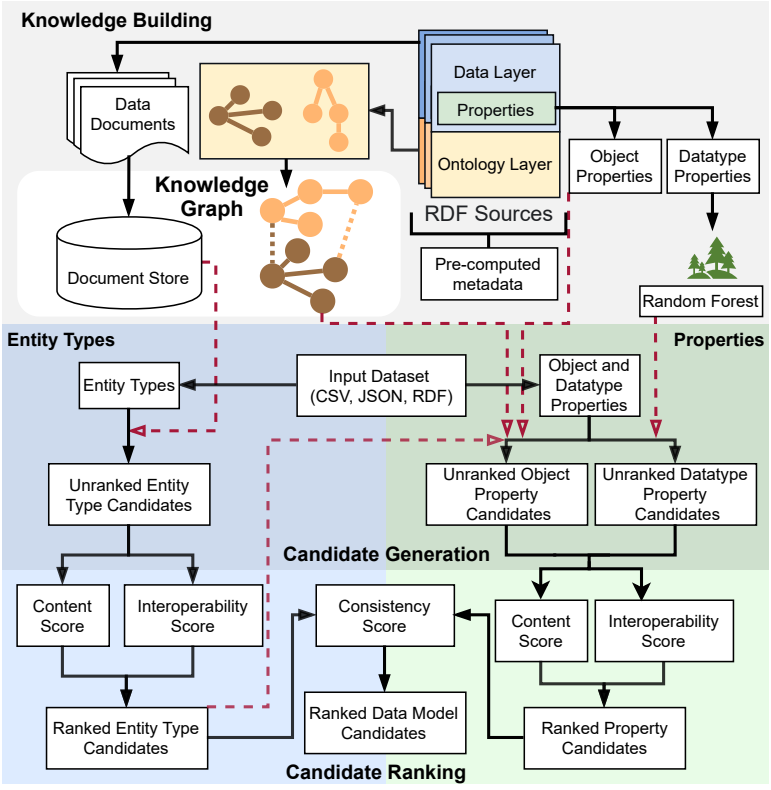

Figure 1: Framework diagram. Red dashed arrows indicate the structures involved in the generation of candidates for each data model elements.

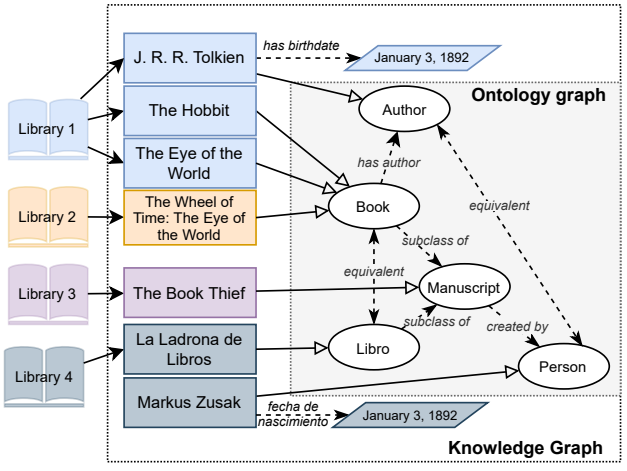

Figure 2: Example of knowledge building from RDF sources.

Figure 2 illustrates the knowledge building process, more specifically tasks (1) and (2). In this figure, we have four RDF datasets from libraries and some entities and their relationships. The books are of types Book, Manuscript, and Libro. The people in the datasets are of type Author and Person. Those represent classes of hypothetical ontologies that were connected in a single graph through enrichment processes. Both people have a datatype property representing their date of birth. As described below, the ontology graph is created by linking the data models from different sources and enriching the relationships between entities.

5.1. Building the Knowledge Graph

The Knowledge Graph is built from several RDF data sources that are assumed to (1) be previously published, (2) follow the RDF data format, (3) cover a similar or related domain to the data that needs to be modelled and (4) include equivalences for entities and properties to be modelled.

The data sources are indexed in a document store[10] with the entities parsed into a structured JSON (JavaScript Object Notation) format, where the attributes are the properties and the values are the objects of those properties. In our implementation, we used the Raptor RDF Syntax Library[11] which converts several RDF file formats into a common JSON structure. We merge blank nodes (i.e., entities without URI, that can be considered independently of other entities) with their referencing entities and exclude documents that do not have a `rdf:type` property.

5.2. Creating the Ontology Graph

The KG is supported by a background ontology graph that provides a wider data model but also includes relationships and entity types that might not be represented in the data. This ontology graph, therefore, allows for broader

[10] Our implementation uses ElasticSearch - https://www.elastic.co/elasticsearch (Accessed in September 2020)

[11] http://librdf.org/raptor (Accessed in September 2020)

searches that consider not only the entity type of a certain entity but also the neighbours and relationships of that entity type. The graph can then be exploited to compute paths between nodes and find more relationships between concepts.

We obtain this graph by automatically extracting and retrieving from the Web the ontologies used by each RDF data source. If the input datasets to be modelled are provided in the RDF format, then their ontologies are also retrieved to be added to the ontology graph.

Then, we construct the first layer of the ontology graph by considering the relationships in and between the ontology classes and properties. A weighted directed ontology multigraph G is defined as $G = (V, E, W)$, where V is the set of vertices, E is the multiset of edges that connect the vertices, and W is the weighting function for the edges. The vertices represent ontology classes or properties that are connected to at least one edge. Edges represent relationships between ontology classes or properties. Edges created from `owl:equivalentClass` or `owl:equivalentProperty` predicates are added in both directions between the pair of vertices. Similarly, for the predicates `rdfs:subClassOf` and `rdfs:subPropertyOf`, we create the inverse edges and name them `superClassOf` and `superPropertyOf`, respectively, to allow graph traversal through the children vertices. These edges are assigned a weight of 1 since they are explicitly stated. Edges obtained through matching as described below, are weighted higher based on the confidence of any extracted mapping.

We have previously shown that enrichment strategies lead to more cohesive ontology graphs with new paths being created between disconnected ontologies [30]. Therefore, the second layer of edges is obtained with an enrichment step that creates new relationships between ontology classes using: (1) co-occurring entity types, (2) `owl:sameAs` links, (3) ontology matching, and (4) extended string matching. The *co-occurring entity type* edges are obtained by extracting entities that have more than one type assigned in the KG. The relationship between the two entity types is usually equivalence or subsumption and, therefore, an edge is added between the two ontology classes with an assigned weight of 1. Next, we extract new relationships from `owl:sameAs` *links* by linking entity types vertices of entities connected via the `owl:sameAs` property and attribute a weight of 1 to these edges.

The final enrichment step adds edges from string matching techniques. First, we use ontology matching techniques to discover new relationships between ontology classes and properties in the ontology graph. We use the AgreementMakerLight (AML) [10] ontology matching system since it is consistently one of the best-performing systems in the Ontology Alignment Evaluation Initiative (OAEI) [11]. We adapted AML's workflow to efficiently process bulk matching requests from the pairwise combinations of ontologies. We use the AML WordNet Matcher to extend the lexicon of the labels in the ontologies and use the Word Matcher to obtain the mappings. This matcher uses a bag-of-words strategy to find word overlaps between two ontology class labels and scores these mappings with a modified Jaccard similarity. The mapping score $score_m$ ranges between 0 and 1, where 1 represents the highest similarity between two mapped classes. A threshold can be set to exclude mappings below a certain score. We weigh each new edge as $W(e) = 1 + (10 - (10 \cdot score_m))$. We apply a linear transformation to the score so that the edge weight ranges from 1 to 11 and lower mapping scores have significantly heavier edge weights than higher scores, making them harder to traverse during graph searches.

The ontology matching enrichment step is evaluated using precision and recall metrics. We evaluate the mappings against varying confidence thresholds to understand how this parameter affects the results. We performed a baseline evaluation of the chosen ontology matching strategy by selecting a subset of two (cmt and conference datasets) of the manually curated reference alignments of the OAEI Conference track [48]. However, since this reference alignment evaluates only equivalent mappings, we added the COMPOSE reference [46], where the author extended some of the conference reference mappings to also include subsumption correspondences.

In addition to ontology matching, if the input datasets are not in RDF format, we perform *extended string matching* step between data sources and ontologies. In this step, we match ontologies extracted from the RDF data sources with the data model inferred from the semi-structured input datasets From RDF input data, we extract entity types and properties labels. For datasets that do not have these resources explicitly described, we use attributes or table headers as input for this matching task. Using the labels previously obtained, we search them in this ontology index and retrieve the top-10 matches. We score the matches using the string similarity score described in Section 7.1.1. We add these matches as new edges and weigh them using the same linear transformation used for ontology matching edges.

The extended string matching was evaluated using the full OAEI Conference track data from the 2019 edition[12]. For this evaluation, we used the last fragment of each URI (i.e., the remaining of the URI after the # symbol which is present in all the URIs of the ground truth) as the label and searched for each label in all other conference ontologies. We then applied the extended similarity methods over this set of labels and ontologies.

5.2.1. Ontology Matching Evaluation Results

Figure 3 shows the precision and recall results of this evaluation. We verify that, as expected, the total number of mappings decreases with the increase of the confidence threshold and the precision increases. Contrary to the expected result, recall is also relatively low, even when the threshold is 0. This is an unexpected result since AML is one of the best-performing systems in the OAEI in the conference track of the competition. This lack of perform-

[12] `http://oaei.ontologymatching.org/2019/conference` (Accessed in September 2020)

ance can be explained by considering that our edge enrichment strategy used only a single matcher, the Word Matcher, as opposed to the combination of matchers that AML uses for the OAEI competition. The main methodology of the chosen matcher is a bag-of-words match, which in the case of this dataset does not obtain a good performance since there is a high variance between equivalent terms in the mappings. For example, `http://cmt#SubjectArea` is equivalent to `http://conf0f#Topic`. The Word Matcher is not designed to pick up on these matches, therefore, the recall for this dataset is low. Nonetheless, we decided that, for the case of our framework, this is a simple matcher that will efficiently obtain relevant edges since matches will need to have at least one overlapping word, increasing the chance that the concepts are related. The additional matchers of AML add to the complexity of the matching algorithm and, in our use cases, there were approximately 40 ontologies to match, with the biomedical use case having large ontologies that make the process a complex task. Therefore, we opted for a less complex solution which ended up resulting in lower performance for the AML matching system. Furthermore, since naturally recall decreases with the increase of the threshold and our goal is to support the generation of extensive lists of candidates, in our case, it is preferable to maximise recall instead of precision. Since we use the confidence value of mappings to put a higher weight on weaker matches in the ontology graph, making them harder to traverse during path computations between vertices, the impact of incorrect mappings should be reduced.

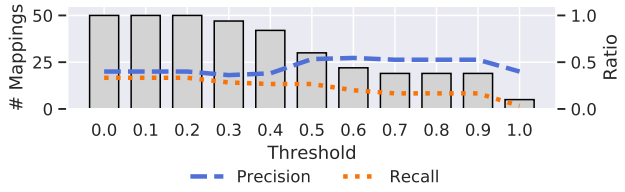

Figure 3: Ontology matching evaluation.

This evaluation considers only equivalence and subsumption relationships. However, other correspondences exist between ontology classes, such as between `bibframe:Status` and `bibo:DocumentStatus`, which are not equivalent or subsumed but are conceptually related. Therefore, in our experiments, we kept every mapping by setting the confidence threshold to 0 and relied on the edge weights to reduce the influence of incorrect mappings without discarding possible relatedness correspondences.

5.2.2. Extended String Matching Evaluation Results

Following the extended string matching methods described, we obtained 1336 class mappings with 129 missing from the reference produced in the previous section, and 144 property mappings with 77 missing from the reference. Figure 4a shows the number of mappings, the precision, and the recall for class mappings and Figure 4b shows the same for property mappings. This extended string matching technique performs well when compared with the adapted AML ontology matching strategy but performs better with class names than property names in this dataset. In the case of the property matching, precision and recall are affected by the high percentage of `rdfs:label` values which are in dromedary case and are not matched against equivalent properties in other ontologies that are stylised with spaces in the label. Furthermore, through manual inspection, we found several cases of false negatives, e.g., in the reference, `http://conf0f#hasTopic` is equivalent to `http://conference#has_a_track-workshop-tutorial_topic` but the extended string matching only finds the match to `http://edas#hasTopic`, which is arguably correct. Finally, the last contributor to the low performance with properties are cases similar to the ones that degraded the performance of the ontology matching system, i.e., words that are synonym but have no overlapping terms. Therefore, for both classes and properties, we keep the threshold at 0 and weight the edges added with a linear transformation to increase the weight of mappings that are less likely to be correct.

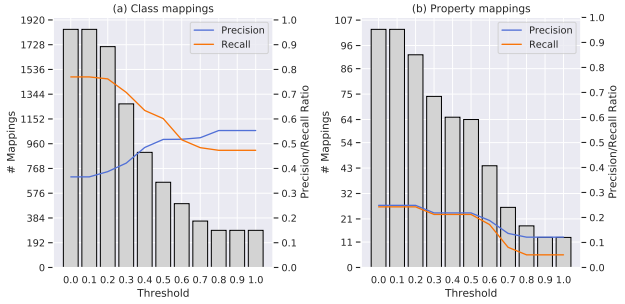

Figure 4: Extended matching evaluation.

5.3. Metadata Extraction

This task focuses on extracting metadata from the RDF data sources that is not easily retrieved either from the document store or the ontology graph. Therefore, we extract and store useful information in intermediary structures that will later be used to more efficiently generate and rank entity types and property candidates.

First, we create an Entity Type Frequency (ETF) map, a map of the number of times t each resource r (e.g., entity type or property) appears in each of the RDF data sources ds, in the form of $ETF[ds][r] \leftarrow t$. Considering that some RDF data sources assign more than one type per entity, the sum of frequencies for each data source is greater than the total number of documents stored.

Then we extract the Knowledge Graph Patterns (KGP) list of tuples, where each tuple contains the triple patterns of the data model of an RDF data source, in the form of $\langle \texttt{domain} - \texttt{property} \rightarrow \texttt{range} \rangle$. Each tuple also includes the frequency of each triple pattern in the data sources.

5.4. Datatype Property Classification Model

In the machine learning field, classification is a supervised learning task where a model is trained to identify the

categories for input data. Similarly to Chen et al. [4], we treat the datatype property generation as a multiclass classification task, where each datatype property in the knowledge graph is a class and its value is processed into a feature vector to use as training data. We use a random forest classifier, which is an ensemble learning method for classification that works by training several decision trees and averaging the predictions to produce a final model. This method helps in correcting the tendency of decision trees to overfit their training set. We use the scikit-learn implementation[13] of the random forest classifier.

We train models for each RDF source in the knowledge graph and the knowledge graph as a whole. The single RDF data source model will tend to overfit since their data model is more homogeneous. However, the goal is to annotate the properties of similar data, so these individual models can provide valuable predictions. When there is variability in the values of datatype properties, the complete knowledge graph model will likely perform better since it provides a broader view of values for datatype properties that are used by multiple RDF data sources.

The model training and test methodology used in our approach (1) defines features to extract from the datatype property values, (2) performs hyperparameter tuning with randomised and exhaustive parameter optimisation, (3) finds an optimal document sample size from the knowledge graph to train the model, (4) fits the models for each RDF source in the knowledge graph and the whole knowledge graph.

5.4.1. Feature Selection

We performed an empirical analysis of the data in the library use cases to better understand the kind of features that had the potential to more accurately distinguish between datatype properties. We concluded that the classification model should be able to clearly distinguish between textual and numerical datatype properties. For example, in the library dataset, it is common to have numerical codes to uniquely identify books and, at the same time, textual properties describe attributes such as title and author. However, the classification model not only has to distinguish between generic datatypes of properties but, among the same datatype, has also to be able to discern more subtle differences between property subtypes. For example, the model has to be able to distinguish between an author's name and a title. In this example, the distinction is not trivial, since it is not uncommon for book titles to overlap with people's names. Nonetheless, in our scoring methodology, the frequency of a candidate datatype property is taken into account to minimise overlapping cases, such as the title and author name. To capture these characteristics, we defined the following features:

(1) Total number of characters in the literal

(2) Number of digits

(3) Number of letters

(4) Number of white spaces

(5) Number of other characters

(6) Number of uppercase letters

(7) Is date? (yes or no)

The Feature 7 was obtained through a rule-based methodology where we use the dateutil[14] Python package to parse the literal. If the literal is recognised as a date, the following rules are checked:

(1) Is Feature 3 > 0?

(2) Is the value decimal?

(3) Is it a negative number? (i.e., starts with a minus sign)

(4) If there are no letters or spaces, does it have more than 3 numbers?

If any of the answers are positive, then the value is not considered a date, otherwise, it is a date.

We evaluated this rule-based methodology by manually creating a ground truth with all the datatype properties in the library knowledge graph that have date datatypes. Then we obtained a random sample of 100 entities from each dataset in the library knowledge graph and extracted all the pairs of datatype property values in each document. The value was parsed by the *Is date* feature methodology and compared with the ground truth to evaluate the precision and recall.

Table 4
Evaluation of date extraction approach.

Approach	Precision	Recall
SpaCy	0.52	0.85
Duckling	0.44	0.16
dateutil + rules	0.96	0.92

Table 4 compares our approach (dateutil + rules) with the Named-Entity Recognition (NER) modules of SpaCy[15] and Facebook's Duckling[16]. Our approach achieves significantly higher performance when identifying dates in our use case. The modules tested falter in this case because they are trained in full-text corpora where dates are used in the context of a sentence. RDF property ranges do not include any context for the date, which leads to several false positives and negatives.

In our approach, false positives are mostly caused by the `bibo:issue` property. Some documents mistakenly have a date as the issue number, despite the property not being considered a date property. When this property is added to the date properties, the precision increases to 0.99.

[13] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (Accessed on May 2021)

[14] https://dateutil.readthedocs.io

[15] https://spacy.io

[16] https://github.com/facebook/duckling

Table 5
Hyperparameter search.

Hyperparameter	Randomised	Exhaustive	Final
n_estimators	$[100, 1000]$	$[100, 500]$	300
criterion	{gini, entropy}	{entropy}	entropy
min_samples_split	$[0.01, 0.40]$ step $= 0.01$	$[0.01, 0.21]$ step $= 0.05$	0.01
min_samples_leaf	$[0.01, 0.40]$ step $= 0.01$	$[0.01, 0.10]$ step $= 0.02$	0.01
max_features	$[2, 7]$	$[2, 4, 5, 7]$	4
min_impurity_decrease	$[0.0, 0.2]$ step $= 0.01$	$[0.0, 0.1]$	0.0
bootstrap	$[True, False]$	$[False]$	False
max_samples	$[0.7, 0.9]$ step $= 0.1$	N/A	N/A

Table 6
Random forest model evaluation for the library use case.

RDF Source	P@1	P@3	P@5
Library British	0.57	0.87	0.97
Library French	0.47	0.77	0.86
Library German	0.43	0.70	0.80
Library Portuguese	0.71	0.9	0.95
Library Spanish	0.28	0.52	0.63
Knowledge Graph	0.21	0.41	0.52

5.4.2. Hyperparameter optimisation

The scikit-learn implementation of the random forest model provides several tuning parameters. We perform two hyperparameter optimisations: randomised and exhaustive. Through empirical observation of the data, we found that some datatype properties were poorly represented in the data, having significantly less overall representation than the average property. Through experimentation, we also found that, due to the size of the data, the hyperparameter optimisation was a slow task. Therefore, together with the empirical observation of the data, we used a 3-fold cross-validation approach to maximise P@5. For each of the optimisations, we extract 100 random value samples of each datatype property in the knowledge graph.

Table 5 presents the two hyperparameter tuning steps (see the link in Footnote 14 for descriptions of the hyperparameters). We start with a broad range of hyperparameters in the randomised grid search. The results reported in the "Exhaustive" column represent the empirically chosen agreement between the 3 repetitions of the random search. The Final column includes the hyperparameters that maximised P@5 in the exhaustive grid search and were used to fit the random forest models.

5.4.3. Model Fitting

Finally, we fit the random forest model to each RDF data source in the knowledge graph and the knowledge graph as a whole. We fit each model using parameters extracted from the hyperparameter optimisation step, using a sample of size s with 40% of the data being separated in the test set. We evaluate the model with Precision@1, Precision@3, and Precision@5. Since the goal is to obtain an exhaustive list of candidates, Precision@5 is favoured over the other measures.

Through experimentation, we found that the features extracted from the datatype property values are relatively stable and even small sample sizes achieve good performance. On the other hand, the time it takes to fit a model increases linearly with the sample size. Therefore, since sample size does not have a significant impact on performance, we kept the sample size value low and chose $s = 2000$.

Library Use Case Results Table 6 shows the precision results over the test set for each RDF source and the whole knowledge graph in the library use case. Overall, the models obtained a high P@5 with low P@1, with the more diverse knowledge graph obtaining lower results than any model individually. The Spanish library obtained a considerably lower performance than the remaining individual libraries because, besides a higher than average total number of properties, it also includes several properties with similar values, making it harder for a random forest model to predict the properties correctly. For example, for dates, there are at least 9 datatype properties in the BNE ontology and it includes others from external ontologies such as `http://rdaregistry.info/Elements/a/P50038`. The same situation is verified in other properties, where values between properties are similar, making it hard for the model to distinguish between datatype properties. Nonetheless, since the aim of the random forest model is to produce an exhaustive list of datatype property candidates, we give preference to P@5. The high performance of the individual models is likely connected to overfitting. However, considering that the datatype properties to be extracted from these models are expected to be in the same domain, following the same structure and semantics, we do not necessarily consider overfitting an issue. The overfitted models are desired in the case of a dataset with data very similar to any of the RDF sources individually. Nonetheless, the whole knowledge graph model is expected to suffer less from overfitting, providing a broader view of the domain to match cases that might not exactly follow any particular data model included in the knowledge graph.

Overall, we consider that the performance is adequate for the task at hand since exact matches are not the goal. The trained model will allow the framework to retrieve the datatype properties in each model in a ranked order as scored by the random forest model.

6. Candidate Generation

In this stage of the framework, we generate an exhaustive list of candidates to match entity types, datatype, and object properties between the input data and the KG. As a preliminary pre-processing step, each input dataset is parsed and converted to the same JSON structure of the KG and is stored and indexed in the document store. For ease of understanding, we assume that the input data D takes the shape of a set of $\langle \mathsf{s} - \mathsf{p} \twoheadrightarrow \mathsf{o} \rangle$ triples, which directly translates to RDF data. For JSON and CSV, we loosely adapt the data to fit this model. For JSON data, we consider the identifier of the document as the subject s (if no identifier is given, one is created), the attributes as predicates p, and values as objects

o. For CSV data, we assign a row identifier if none is given and use it as subject s, the header or column number as the predicate p, and the value of each row/column as object o.

The red arrows in Figure 1 indicate the structures involved in generating each one of the types of candidates. The entity type candidates are extracted by searching matching entities in the document store (see Section. Datatype properties are obtained by training a classification model on the datatype properties of the knowledge graph and using it to predict candidates for datatype properties in input datasets. Finally, object properties are obtained after generating and ranking the entity type candidates. We obtain these properties by extracting the domain and range of the object properties in the input data and then extracting the object property candidates through the pairwise combination of all entity type candidates of the domain and range. Then we extract the object property candidates from the relationships between all pairwise comparisons of the entity type candidates in the domain and range.

6.1. Entity Types

This stage generates an exhaustive list of entity type candidates for entities in the input dataset D. As a first optional step, the user can provide descriptors dsc, i.e, column names, attributes, or predicates that describe entities. These descriptors should contain values that identify the entities, even if with some ambiguity, to facilitate the search in other datasets that might not follow the same standards. For example, considering a dataset with book entities, the user might provide the columns, attributes, or predicates that include the titles of books. If not provided, the search is performed over every column, attribute, or predicate related to an entity in the dataset. Optionally, entity descriptors can also be provided for the datasets in the document store. We found empirically that when entity descriptors for both input dataset and document store are provided, they improve the accuracy and efficiency of the candidate generation and ranking.

For RDF input data, we assume the entity type is the object of the `rdf:type` predicate. In JSON data, we ask the user to supply the attribute which denotes the type of the document. If no type is supplied, then the JSON data is treated like CSV data, where each column is assumed to be a potential entity to have an entity type assigned. For both of these cases, if descriptors are supplied, we find only entity type candidates for the columns or attributes indicated.

The entity type generation algorithm obtains entity type candidates for the entity types found in the input dataset D. See Algorithm 1 in the supplementary material for the full pseudocode algorithm.

The first step of the algorithm is to extract the set of potential unique types from the input dataset. Therefore, the function $unique\_types(D)$ extracts the unique types or type equivalent elements T for entities in D. Then, a function $random\_sample(D, t, n)$ returns a random sample of n documents in D of type t. These documents represent all triples associated with a subject of type t in RDF input data, an individual JSON document of type t, or a row in a CSV file that contains a value for column t. Next, for each input triple, we extract all objects that are literals (i.e., $literal(s) = True$) from all properties in entity document e or for all properties in the descriptors dsc if these were provided. The function $search(S, l, w)$ searches the document store S for label l and returns a maximum of w search results.

In our implementation, we use ElasticSearch's multi-match query[17] to search for full-text matches of label l in the datatype property values indexed in the document store. These fields are replaced by entity descriptors if these are provided for the Knowledge Graph data by the user. This algorithm then returns a mapping between entity type t for each entity e and an extensive list of randomly sampled entity type candidates, including the labels that matched between input and document store entities.

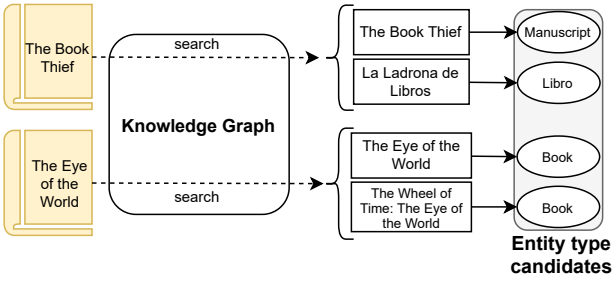

Figure 5: Example of entity type candidate generation

Figure 5 illustrates this process for two books (*The Book Thief* and *The Eye of the World*) from a hypothetical input dataset. In this example, we start by searching the knowledge graph for matching labels and then we extract the entity types from the matched entities. These entity types will represent the unordered list of candidates that will be ranked according to different measures.

6.2. Datatype Properties

The datatype property candidates are generated from the Random Forest models RF trained using the method described in Section 5.4. We train a model for each RDF data source individually and the Knowledge Graph as a whole.

The datatype generation algorithm obtains datatype property candidates for datatype properties in input data D. See Algorithm 2 in the supplementary material for the full pseudocode algorithm.

The algorithm first generates the set DP from the data in input D with unique datatype properties in the input data D. Then we obtain a random sample of entities E_t in the input data that contain each of the datatype properties in DP. After extracting the literal values L of the properties, we use the trained Random Forest models RF to classify the value with a datatype property from the Knowledge Graph. The $predict(rf, l)$ function returns a list ordered by the probability estimates of each datatype property in the classification

[17] https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-match-query.html (Accessed in September 2020)

models matching the literal value. Therefore, we obtain the datatype property candidates C_{dp} by saving the sorted prediction results for each value in each model.

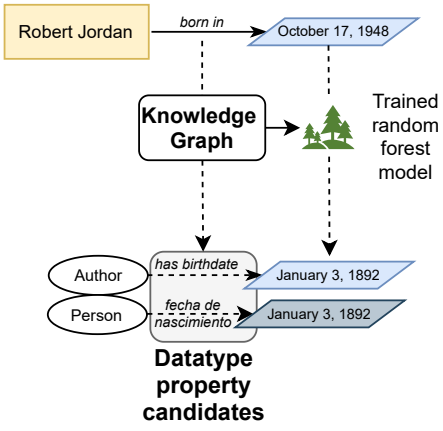

Figure 6: Example of datatype property candidate generation

In Figure 6, the example triple $\langle$`Robert Jordan – born in ⇾ October 17, 1948`$\rangle$ is used to the process of generating datatype property candidates. The literal value (i.e., the date) of the datatype property *born in* is classified by the random forest model trained on literal values in the KG. Therefore, two candidates with literal values with similar features are recognised and proposed as datatype property candidates.

6.3. Object Properties

The object property candidate generation step depends on the generation and ranking of entity type candidates (see Section 7 for more details on the ranking of entity types).

The object property generation algorithm obtains object property candidates for object properties in input data D. See Algorithm 3 in the supplementary material for the full pseudocode algorithm.

First, we obtain all triples which do not have literals as their object. For each property in these triples, we extract their domain and range entity types with the function $type(e)$. The object property candidate map C_{op} is obtained by retrieving the edges in the ontology graph that exist between the pair of subject and object candidates. The function $get\_edges(G, s, t)$ returns a tuple with the object property edges op in graph G between source vertex s and target vertex t and their data provenance $prov$, e.g., $get\_edges(G, s, t) = [\langle op_1, prov1\rangle, \langle op_2, prov2\rangle]$. The ontology provenance refers to the KG data sources that have the candidate edge as an object property. This information is stored as a property of the edge when the edge is introduced in the ontology graph.

Figure 7 shows the process of generating object property candidates with the running hypothetical example. Here we take the triple $\langle$`The Hobbit – written by ⇾ J.R.R Tolkien`$\rangle$, which already has ranked entity type candidates for the entities in the domain and range, and search the knowledge graph for all edges between the entity type candidates. The resulting edges are used as an unordered list of object property candidates.

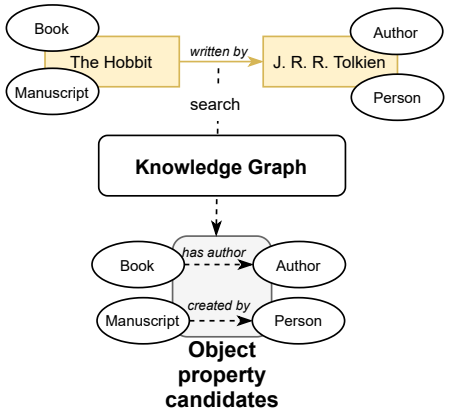

Figure 7: Example of object property candidate generation

Table 7
Candidate generation recall.

Measure	Recall
Entity	0.83
Entity type	0.97
Datatype property	0.95
Object property	0.70

6.4. Experiments & Results

The candidate generation methodology is evaluated in terms of recall since its goal is to generate an extensive list of possible candidates. At this stage, the framework should be able to retrieve the most likely candidates and output an unranked list of entity type, datatype property and object property candidates.

Using the DBkWik dataset and ground truth, we generated candidates using the methodologies previously described. Since this dataset has at most two candidates per input, we evaluate recall by testing if these two candidates are being retrieved. Therefore, candidate generation is evaluated in terms of entity, entity type, datatype property, and object property recall in the DBkWik dataset. Table 7 shows the condensed recall results for all data model elements.

6.4.1. Entity types

Entity recall verifies that the correct entities are being retrieved when searching for the values of the labels in the knowledge graph, while entity type recall checks if, even if the wrong entity was retrieved, the correct entity type was found. The entity type recall follows the intuition that even if similar entities (e.g., different books from the same series) are incorrectly retrieved in some instances, their type is likely the same. Through experimentation (see Figure 3s of the Supplementary Material), we found that the number of search results retrieved stabilised between a search window of 10-15. The input sample size also did not have a large impact on this evaluation dataset (see Figure 4s of the Sup-

plementary Material). On average, each entity type in the ground truth has 55.2 entities, with a median of 6. Only 2 entity types have more than 1000 entities, and 27 have more than 100. Therefore, sampling above a hundred is mostly meaningless for this dataset. Nonetheless, we maintained a sample size of 2000 throughout the experiments to guarantee optimal results.

Therefore, in Table 7, for entity and entity types, we present the results for a search window of 15 results and an input sample size of 2000. With these parameters, the entity type candidate generation methods obtained reasonably high recall for the evaluation dataset. The high entity type recall also confirms the intuition that, in some cases, even when the correct entity is not retrieved, the entity type is still correct.

6.4.2. Datatype properties

Due to the nature of the datatype property generation, all possible knowledge graph datatype properties are retrieved, ordered by their probability of fitting a certain value. Therefore, we would expect a recall of 1. However, Table 7 shows a recall lower than 1 because the ground truth features mappings between object properties and datatype properties. Our framework strictly distinguishes datatype and object properties. Therefore, a property cannot be both. When the framework finds a property that is both a datatype and an object property, it assigns it to the object properties. For example, the property `http://dbkwik.webdatacommons.org/starwars.wikia.com/property/title` in the SW dataset is mapped as equivalent to `http://dbkwik.webdatacommons.org/swg.wikia.com/property/title` in SWG and to `http://dbkwik.webdatacommons.org/swtor.wikia.com/property/title` in SWTOR. However, in SWTOR this property is both a datatype property and an object property, while in SWG it is only a datatype property. Therefore, this property is considered to be an object property. This pattern is found for 345 properties over all the DBkWik datasets.

6.4.3. Object properties

The performance of this generation step is directly correlated with the performance of the entity type generation and content score ranking. This generation step extracts an exhaustive list of all edges between two vertices in the ontology graph. We obtain a reasonably high recall. Similarly to the datatype property generation, the performance of the object property generation is degraded by overlapping mappings between datatype and object properties in the DBkWik ground truth.

6.5. Discussion

In terms of entity type generation, currently, the framework only uses `owl:sameAs` links as a means to enhance the ontology graph to improve scoring metrics that rely on traversing the graph. However, in the future, these links can be directly explored during the entity generation step to produce a more reliable candidate list. For example, similarly to the methods to create the `owl:sameAs` ground truth, when using an RDF dataset as input, if it features `owl:sameAs`, these can be extracted to generate candidates. If the input dataset does not use these links, but they exist in the knowledge graph, they can be explored to provide a more complete list of entity type candidates by extracting the entities that are indicated as values of the `owl:sameAs` property.

When generating property candidates, the first challenge is recognising if a predicate, attribute, or column is supposed to be a datatype or object property. This issue is less critical in RDF input datasets since distinguishing is a matter of finding if the dataset features the URI in the object of the predicate. This is the approach we use in general in this framework. However, for semi-structured CSV and JSON datasets, entities are rarely referenced via a unique identifier. Therefore, distinguishing datatype from object properties is not trivial. For example, in a CSV file, a column might contain book titles, while another contains author names. Looking at the existing data models in the library data domain, it is common that both these columns represent entities with a relationship between them. Although this seems like a trivial example, in the context of an automatic approach, there are several considerations. First, if the input dataset labels overlap with the knowledge graph labels, most columns will find some entity to match, even if incorrectly. For example, a column with page numbers can include a row for a book with 451 pages that can be matched incorrectly to the book title *Fahrenheit 451*. Therefore, robust measures need to be put in place to guarantee that only entities are obtaining entity type candidates. Currently, our generation methodology does not take this into account and the filtering of bad candidates is left to the content scoring. In the future, the framework should integrate more robust methods to distinguish between datatype and object properties, consider all the factors, and improve the candidate generation methodology.

Finally, the last challenge of entity type and property candidate generation is missing concepts, i.e., the input dataset includes data that is not found in the knowledge graph. For example, the University dataset was automatically converted to the BIBFRAME schema. Due to the scope of this schema, it includes several concepts that are not found in other library schemas, such as the entity type `bibframe:ColorContent` or the property `bibframe:baseMaterial`. These concepts do not have a direct overlap in any of the data models adopted by the European libraries in study. Therefore, finding suitable candidates for these concepts in the knowledge graph is not possible. In this case, an investigation would need to be carried out to assess how to best handle the situation, considering that the immediate approach would most likely be to extend the ontologies in the graph and manually produce the annotations to complete the proposed data model. In a future implementation of this framework, ideally, the user would have an integrated feature that would allow the extension of the ontology without leaving the interface that implements the framework.

7. Candidate Ranking

To rank the entity type, datatype, and object property candidates generated, we use three scores: Content, interoperability, and consistency score. This step ranks individually the lists C_{et}, C_{dp}, and C_{op} resulting from the generation step with the content score, returning CS_{et}, CS_{dp}, and CS_{op}, and the interoperability score, returning IS_{et}, IS_{dp}, and IS_{op}. The content score focuses on measuring individual characteristics of the entity type or property candidates, such as the frequency in the search results and distance in the ontology graph. The interoperability score focuses on boosting the score of candidates with higher connectivity in the ontology graph. Finally, the Consistency score transforms the individual lists of ranked candidates into cohesive data model candidates DM and combines the individual scores with a score of the frequency of triple patterns in the input data. It should be noted that when we mention consistency with the data model, we are not referring to logical consistency within the ontologies, e.g., verifying if disjoint axioms are not being ignored. The data model consistency is verified in terms of the frequency of a triple in the KG and consistently suggesting the same candidate for the same input. Therefore, this consistency score takes into account the co-occurrence of the triple and pairwise combinations of $\langle \texttt{domain} - \texttt{property} \twoheadrightarrow \texttt{range} \rangle$ in the KG to score the consistency of the candidate triples.

The next sections detail the metrics used in each type of score. We then describe the specific scoring methodology used for each type of candidate in the data model.

7.1. Content Score

The content score combines metrics based on string similarity and search result frequency into a single score $CS_c \, \forall \, c \in \{C_{et}, C_{dp}, C_{op}\}$. The final candidate ranking produces a set of candidates that are sorted according to their appropriateness in matching entity types of the input data, considering the metrics used. We obtain this result for each of the data model elements by using different combinations of the measures described in the next sections.

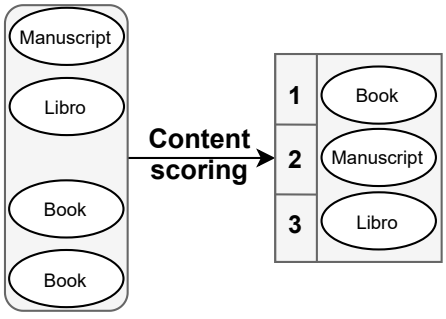

Figure 8: Example of content score ranking

Figure 8 follows the running example where, from the entity type candidates previously generated, we rank them according to different measures. In this figure, we use the example entity type candidate ranking, however, the process is similar for any of the components of the data model by adapting the metrics that are used to rank the candidates (see Section 7.1.2). At the end of the content score ranking, we have an ordered set of candidates.

7.1.1. Content Score Metrics

The following metrics were empirically developed for the framework by studying the needs and characteristics of the use case datasets. Overall, RiCDaM has metrics to (1) improve precision, and (2) improve the cohesiveness of desirable vs. undesirable results. The first proposed methods are aimed at ranking desirable candidates above undesirable and the second provides means to bring the scores of desirable candidates closer together so that the score of a candidate is not artificially inflated by the precision metrics. The intuition behind the development and implementation of each measure is discussed individually in their respective sections. In these measures, we refer to a generic list of candidates C that can represent any of the data model elements (i.e., entity types or properties) in the data model of the input data.

String Similarity When comparing labels of entities we consider that a label is more likely to be correct if it is closer to the input label. Therefore, we applied string similarity measures to find the degree of similarity between input and candidate labels.

We experimented with classical string similarity methods and devised a modified string similarity score *sim* based on the concept of n-grams. In this work, we use a bi-gram ($n = 2$) function $n\text{-}gram(s)$ to decompose a string s into a set of pairs at the letter and word level, e.g. *Harry Potter* is decomposed into {("_h"), ("ha"), ("ar"), ("rr"), ("ry"), ("y "), (" p"), ("po"), ("ot"), ("tt"), ("te"), ("er")} at the letter level and {("_", "harry"), ("harry", "potter")} at the word level. Left padding (_) is added to emphasise prefixes.

We first pre-process the strings by removing punctuation and stop words, then we define the n-gram similarity with a Jaccard index $g\text{-}sim$ between sets A and B as follows in Equation 1:

$$\begin{aligned} union(A, B, n) &= |n\text{-}gram(A) \cup n\text{-}gram(B)| \\ g\text{-}sim(A, B, n) &= \frac{|n\text{-}gram(A) \cap n\text{-}gram(B)|}{union(A, B, n)} \end{aligned} \tag{1}$$

We then define the letter and word level n-gram similarities $l\text{-}sim$ and $w\text{-}sim$ using the letter and word n-grams A_l, B_l and A_w, B_w as follows in Equation 2:

$$\begin{aligned} l\text{-}sim(A_l, B_l, n) &= g\text{-}sim(A_l, B_l, n) \\ w\text{-}sim(A_w, B_w, n) &= g\text{-}sim(A_w, B_w, n) \end{aligned} \tag{2}$$

Lastly, we compute the string similarity metric $s\text{-}sim$ as

follows in Equation 3:

$$inv(A, B, n) = \begin{cases} union(A, B, n), & \text{if } union(A, B, n) \leq 1 \\ \frac{1}{union(A,B,n)-1}, & \text{if } union(A, B, n) > 1 \end{cases}$$
$$s\text{-}sim(A, B, n) = w\text{-}sim(A_w, B_w, n) \cdot (1 - inv(A_w, B_w, n)) + l\text{-}sim(A_l, B_l, n) \cdot inv(A_w, B_w, n) \tag{3}$$

The word-level similarity $w\text{-}sim$ gives a better understanding of the equivalence of two entities based on their labels. However, when the labels have few words, this metric loses its meaning. Therefore, the $inv(A, B, n)$ function balances the weight given to $w\text{-}sim$ versus $l\text{-}sim$. For example, when comparing *Harry Potter* with *Harry's Pottery*, $l\text{-}sim = 0.73$, $w\text{-}sim = 0.0$, and $s\text{-}sim = 0.24$, therefore, representing the level of similarity between the strings but emphasising that, at the word level, the strings do not overlap.

Finally, considering two sets of strings L_e and L_c, we obtain the string similarity score sim by computing the maximum pairwise string similarity between the two sets as seen in Equation 4:

$$sim(L_{e,t}, L_{c,t}) = \max_{\substack{l_e \in L_{e,t} \\ l_c \in L_{c,t}}} s\text{-}sim(l_e, l_c, n) \tag{4}$$

We select candidates that have $sim >= t$, where t is a user-provided threshold.

Search Results Frequency The search results frequency score $freq_s(c)$ follows the intuition that the most common candidate in the list of search results has a higher likelihood of being a good candidate. Considering that $count(c)$ is a function that represents the raw count of the candidate c in a list of candidates C, we apply the following transformation to obtain a final $freq_s(c)$ in Equation 5:

$$freq\_norm(c) = log(1 + count(c))$$
$$freq_s(c) = \frac{freq\_norm(c)}{\max_{c \in C} freq\_norm(c)} \tag{5}$$

KG Frequency This metric measures how frequent a resource is in the knowledge graph. We extract the count of a candidate c from the ETF map (see Section 5.3). We compute the normalised frequency per RDF data source ds as seen in Equation 6:

$$freq_{kg}(ds, c) = \frac{EFT[ds][c]}{\max_{c \in EFT[ds]} freq_{kg}(ds, c)} \tag{6}$$

Borda Score This score is based on the Borda count election method [35], where voters rank candidates in order of preference. In opposition to majority election, the Borda count elects the candidate that is more broadly accepted by the voters.

Borda count works by distributing several points determined by the number of candidates n. Each candidate c receives $n - r_v$ points, where r_v represents the ranking of the candidate in a ballot of voter v, starting from rank 0 for the candidate ranked first. For example, with $n = 3$ candidates per ballot, the candidate ranked first will get 3 points, the second will get 2, and the 3rd will get one point.

The intuition behind this score is that when the initial candidate generation already has a preliminary ranking, as is the case with the entity type and datatype property candidates, instead of just applying flat frequency we consider the ranking in the list of candidates.

With $N = |C|$ and a pool of voters $V = \{v_1, v_2, \dots, v_k\}$, we compute the Borda Score $borda\_score(c)$ for candidate c with Equation 7.

$$borda\_count(c, V) = \sum_{k=1}^{|V|} n - r_{v_k}$$
$$borda\_score(c, V) = \frac{borda\_count(c, V)}{\max_{c \in C} borda\_count(c, V)} \tag{7}$$

Distance to Source Resource Using the edge-enriched ontology graph, we compute the distance between the source resource r and each candidate c in the candidate list C. This score follows the intuition that candidates closer to the source resource in the ontology graph are more likely to be good candidates. We calculate the score $dist_r(r, c)$ between a source resource r and a candidate c as seen in Equation 8.

$$dist_r(r, c, G) = 1 - \frac{log(shortestDistance(r, c) + 1)}{log(\max_{c \in C} dist_r(r, c, G) + 1)} \tag{8}$$

7.1.2. Content Scoring Methodologies

The content scoring methodologies differ between the three types of resources in the RDF data model to accommodate their different characteristics. Below, we present the individual content scoring methodologies for each element type in the data model.

Entity Type Content Scoring Considering a list of candidates C_{et} with all the generated unranked entity type candidates with source entity labels L_e and target candidate labels L_c, this algorithm computes the content score for entity type candidates, obtaining the ranked candidates CS_{et}. See Algorithm 4 in the supplementary material for the full pseudocode algorithm.

For this data model element, we start by calculating the string similarity of each candidate and filtering out the candidates with similarities below threshold α. Then we select the maximum similarity sim for each candidate in the search results of each query. Finally, we obtain the frequency of each candidate c in the search results per entity type query. We then ponder the mean of the average similarity $mean(sim)$ of a candidate and its distance to source resource $dist_r$ by the search result frequency $freq_s$ into a final cs score per candidate per type and select the k elements with the highest values of cs per type t.

Datatype Property Content Scoring Considering the datatype property candidate mapping C_{dp} generated through the processes described in Algorithm 2 in the supplementary material, this content scoring algorithm re-orders the candidates following the process described in Algorithm 5 of the supplementary material.

First, the algorithm computes the Borda score per datatype property dp, per entity e, and per model rf for each datatype property candidate in the ranked prediction p. In step 2, we aggregate the Borda score results per input datatype property and candidate. In step 3, we average the Borda scores that each candidate got in the different entities and models and average this score with the distance to source $dist_t$. Finally, the mapping CS_{dp} returns the top_k candidates per datatype property dp considering the highest content score cs computed.

Object Property Content Scoring Considering the object property candidates mapping C_{op} generated through the processes described in Algorithm 3 in the supplementary material, this content scoring algorithm re-orders the candidates following the process described in Algorithm 6 of the supplementary material.

We first iterate over each candidate c and its provenance ontology o to extract $freq_s(c)$ where the raw count is represented by the number of times a tuple in $edges$ contains candidate c. The frequency $freq_{kg}$ represents the number of documents in the Knowledge Graph that use object property candidate c. Finally, the algorithm computes $dist_r$ and combines all scores into the final content score cs of candidate c. The algorithm returns the top_k object property candidates sorted by content score cs.

7.1.3. Content Scoring Experiments & Results

For the content scoring experiments, we chose a threshold of 0.6 since it provides a good balance between high precision and a reasonable number of search results for both entity and entity type candidate rankings. In Section 3 of the Supplementary Material, we present results and discuss the experiments to find this threshold, together with the experiments that test different string similarity measures and their performances.

We evaluate the content scoring methodology in terms of the effectiveness of each metric. The effectiveness of the metrics in improving the content score is two-fold: (1) improving the overall precision of the candidate retrieval and (2) approximate the scores among the relevant candidates and the non-relevant candidates. This second goal of the score represents the desirability of a candidate. When several candidates are retrieved we want to favour relevant candidates even if they are less performant in terms of, for example, search results retrieved. Therefore, we evaluate the entity type and property methodologies using the Mean Average Precision (mAP) and the standard deviation of the scores among relevant and non-relevant candidates. The mAP allows us to get a precision metric that is more geared towards the ordered nature of our results since RICDaM does not output a single best candidate but, instead, ranks possibilities. The standard deviation tests if the relevant or non-relevant results are getting closer scores with each step. This closeness enables candidates to not dominate over other candidates solely over the frequency in which they appear in the search results. This closeness is desirable especially considering the next step where the interoperability score will be calculated. If a candidate has a content score that is far above all other candidates, the interoperability score will be negligible without a heavy weight over the content score. Ideally, mAP increases with each step while the standard deviations tend to approximate 0.

We evaluate the content scoring methodologies using the DBkWik and Gutenberg ground truths. We compare the results of the systems that participated in the OAEI 2020 Knowledge Graph track [2] with RICDaM's results. Table 8 shows that the precision increases with each step in both the entity types and the properties. In the case of the entity type methodology, $dist_r$ metric does not improve performance but greatly improves the closeness between the scores of the relevant and non-relevant candidates. For the properties, we verify that the second metric increases the dispersion of scores but greatly improves the precision over the base order. Overall, RICDaM also performs better in the Gutenberg ground truth which, even though it is a partial ground truth, showcases the strengths of RiCDaM. The results are improved in all aspects. The entity types in this ground truth are the most complete set of data model elements and Table 8 shows results similar to the DBkWik ground truth. Despite the smaller sample size for the properties, the system is still able to reliably retrieve relevant candidates for the tested cases, showcasing that RICDaM can reliably retrieve a good set of ordered candidates in a relevant amount of cases.

To the best of our knowledge, no system has been published nor exists to directly compare against RiCDaM. Therefore, we further evaluate RICDaM by comparing our results with the closest approaches that are represented by the aggregated OAEI 2020 Knowledge Graph Track results. However, the scope of these approaches is not the same as our framework, neither is the aim of each system. Nonetheless, in Table 9, for the OAEI Systems that participated in the knowledge graph track, we present the top result and, in brackets, the average of all systems that obtained a score above 0. The OAEI track evaluates systems in terms of precision which considers all candidates that are missing from the ground truth as false positives. RICDaM is likely to include non-relevant results since it selects the top-k results for each input, whether this input has one or more relevant matches. Therefore, we also include Precision@1 for RICDaM's evaluations so that we can evaluate the system in the case of only a single candidate being retrieved instead of an ordered list. Furthermore, the OAEI track results do not differentiate between datatype and object properties, therefore, we present the aggregated results for these properties labelled as DP+OP.

Overall, when comparing to OAEI systems in terms of recall, RiCDaM obtains lower performance than the top sys-

Table 8
Entity type and property candidate content scoring ranking evaluation.

Element	Dataset	Metrics	mAP	Standard deviation	
				Relevant	Non-relevant
Entity Types	DBkWik	(1) sim	0.861	0.072	0.124
		(2) sim & $freq_s$	0.828	0.374	0.323
		(3) sim & $freq_s$ & $dist_r$	0.828	0.187	0.161
	Gutenberg	(1) sim	0.644	0.071	0.084
		(2) sim & $freq_s$	0.853	0.318	0.085
		(3) sim & $freq_s$ & $dist_r$	0.853	0.159	0.043
Datatype Properties	DBkWik	$borda\_score$	0.122	0.151	0.231
		$borda\_score$ & $dist_r$	0.326	0.215	0.142
	Gutenberg	$borda\_score$	0.109	0.051	0.262
		$borda\_score$ & $dist_r$	0.695	0.129	0.181
Object Properties	DBkWik	$freq_{kg}$	0.043	0.005	0.011
		$freq_{kg}$ & $dist_r$	0.489	0.180	0.187
	Gutenberg	$freq_{kg}$	0.322	0.006	0.004
		$freq_{kg}$ & $dist_r$	0.745	0.26	0.109

Table 9
Entity type candidate content scoring ranking evaluation.

System	Dataset	Element	Precision	Precision@1	Recall
OAEI Systems	DBkWik	ET	1.0 (0.83)	-	0.98 (0.68)
		DP+OP	0.91 (0.70)	-	0.86 (0.64)
RiCDaM	DBkWik	ET	0.539	0.767	0.889
		DP+OP	0.002	0.429	0.702
	Gutenberg	ET	0.377	1.0	1.0
		DP+OP	0.008	1.0	0.968

tem but is comparable to the average for the DBkWik ground truth. In terms of precision, RICDaM obtains considerably lower performance but, when considering precision@1, the entity type performance is close to the average, while still underperforming in terms of properties.

Overall, our framework obtains lower precision than the maximum and average precision obtained by systems in the OAEI track but recall comparable to the average of the OAEI systems. As expected, RiCDaM performs better when evaluated against the Gutenberg ground truth since this ground truth better showcases the strengths of the RiCDaM methodology. For the evaluation against the Gutenberg ground truth, the recall is less reliable since the ground truth may be incomplete. Nonetheless, results are considerably higher when using this ground truth.

7.1.4. Discussion

To evaluate the content scoring methodologies we compare the performance of RiCDaM against the systems that participated in the OAEI Knowledge Graph task. The main focus of those systems is to improve the performance of their task-specific matchers, therefore, excelling at individual matching tasks of instance and schema of a single knowledge graph. These systems use combinations of string similarity measures, background knowledge exploitation, structural matching between instances in knowledge graphs, or reasoning over ontologies to obtain their results. In contrast, our system focuses on integrating several RDF knowledge graphs, instead of knowing *a priori* which target is more suitable to each input resource. The number of knowledge graphs being integrated increases the complexity of the matching task and, therefore, the complex combination of algorithms these systems use is not directly applicable to the task handled by our framework. In addition, as previously mentioned, our system strictly differentiates between datatype and object properties, which further hinders performance when using the DBkWik dataset.

In addition to the evaluation against the OAEI systems, we also present an evaluation and analysis of the effect of the different metrics in the framework. In general, the methodologies obtain a reasonable performance, with the entity type methodology achieving better results than the property scoring methodologies. Nonetheless, for the library use case, the methodology is successful in re-ordering the candidates that were generated. However, issues with the generation methodology are propagated for the ranking stage. Therefore, if

poor results were achieved at that stage, the scoring methodologies are not going to achieve optimal performance.

Furthermore, due to the modular nature of the framework, for a particular dataset, a new measure might perform exceedingly well and, in that case, the framework can be extended to include the results of that metric.

In terms of the evaluation performed on the content score rankings, we were only able to use the ground truth to evaluate precision and recall-based metrics since, to the best of our knowledge, no ground truth exists to evaluate ranked entities, entity types, or property candidates for data modelling. Furthermore, in the constructed Gutenberg ground truth, it is not possible to provide ranks for the candidates since different existing data models have been manually developed using these candidates, therefore, we cannot claim that one candidate is superior to another, making ranking the candidates a matter of context or expert opinion. Therefore, choosing a candidate over another is not obvious and, in our evaluation, we consider each resource in the ground truth at an equal ranking and evaluate using precision only.

7.2. Interoperability Score

The interoperability score focuses on re-ranking the candidate list obtained via the content score methodology by boosting candidates that are well-connected in the ontology graph and are connected to frequent entity types in the Knowledge Graph. This score ensures that the candidates ranked first are not only good matches but also maximises the integration with frequently used entity types in the Knowledge Graph. Considering that the KG is composed of several potential candidates for each resource, this score improves the integration with related resources by ranking less specific resources higher since they are more likely to contain a broad network of equivalences and subsumptions.

Therefore, the interoperability score IS contains information from the knowledge graph KG and from a subgraph of the ontology graph $G_e = (V, E_s)$ with $E_s \subseteq E$ representing ontology mappings, extended mappings, `owl:equivalentClass`, and `superClassOf` edges only. This subset of edges restricts the graph to the set of equivalent, subsumed, or directly related vertices.

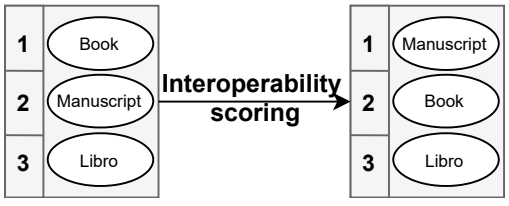

Figure 9: Example of interoperability score ranking

Figure 9 illustrates the impact of re-ranking the candidates according to the interoperability score. In this example, the highest-ranked entity type with content score ends up being ranked second according to the interoperability score. This would be expected since Manuscript is a parent class of Book in the ontology graph. Even though it is not shown in the excerpt of the example KG, it is likely that Manuscript will be more connected than Book in the overall knowledge graph and, therefore, achieve a higher interoperability score than Book.

7.2.1. Interoperability Score Metrics

This score is based on two metrics: *neighbourhood size* and *neighbourhood frequency*. Both scores use the concept of i^{th} neighbourhood $N_G(v, i)$ of vertex v, which is the set of all vertices that lie at the distance i from v in graph G. Neighbourhood size represents the total number of vertices in $N_{G_e}(v, i)$, while the neighbourhood frequency considers the frequency of these vertices in the KG. Higher scores in these metrics translate into a more connected and relevant neighbourhood. In a fully connected graph, the values of these metrics keep increasing until all vertices are included in this score. In contrast to the neighbourhood frequency, the neighbourhood size rewards candidates with neighbours that do not appear in the KG but are still in the path of the candidate vertex in G_e. Entity types with high neighbourhood size scores but low neighbourhood frequency are valuable for integration with datasets not included in the background Knowledge Graph.

Neighbourhood Size Considering V as the set of vertices in G_e corresponding to the candidates in CS obtained from the content score ranking of any data model resource, we calculate the neighbourhood size metric $NS(v, d_{max})$ as the sum of the number of vertices in the i^{th} neighbourhood of vertex $v \in V$ up to a maximum weighted distance d_{max}, i.e., the maximum distance to consider i^{th} neighbourhoods. Therefore, the neighbourhood size $NS(v, d_{max})$ is computed with:

$$NS(v, d_{max}) = \sum_{i=0}^{d_{max}} |N_{G_e}(v, i)| \tag{9}$$

Neighbourhood frequency The neighbourhood frequency is defined as the sum of the frequencies $freq(n)$ of each vertex in the KG over the i^{th} neighbourhood $\forall v \in V : N_{G_e}(v, i)$, where i is the weighted distance from v to a maximum of d_{max}:

$$NF(v, d_{max}) = \sum_{n \in N_{G_e}(v, d_{max})} freq(n) \tag{10}$$

7.2.2. Interoperability Score Methodology

Considering r as a data model resource in $\{et, dp, op\}$, the interoperability score algorithm takes as input the CS_r mapping resulting from the content score algorithms respective to each resource and computes the interoperability score IS_r. See Algorithm 7 in the supplementary material for the full pseudocode algorithm.

This algorithm first finds the vertex in graph G_e that corresponds to each candidate and then computes the neighbourhoods size and interoperability metrics. The mean of these two values is the final interoperability score is.

7.2.3. *Interoperability Score Experiments & Results*

In a first experiment, we assessed the impact of the weights given to content score and interoperability scores. Figure 10 shows the effect of the weights in the MAP evaluation against the Gutenberg ground truth. The figure show that approximately with $w_{cs} = 0.6$ and $w_{cs} = 0.4$ there are not many more gains to precision. Therefore, we set these weights for the remainder of these experiments.

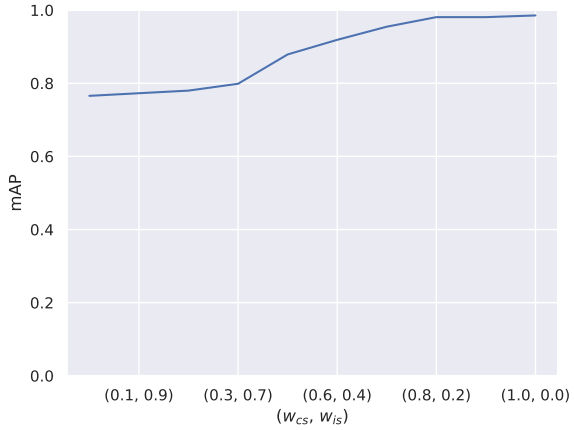

Figure 10: Impact on mAP of content score and interoperability score weights.

Interoperability focuses on maximising the integration between ontology resources selected for the data model and the different schemas in the knowledge graph. To the best of our knowledge, there is currently no ground truth that can reliably evaluate this component of the framework. Since different data publishers have different interpretations of what the best entity type is for their use case, manually creating and curating a ground truth is also not a viable option. For example, in the case of the library data domain, some datasets define the concept of a book as Work, Manifestation, or Bibliographic Resource, while others adopt more specific terms such as Book or Document. One of the selected datasets (British) even uses both general and specific terms to define the concept of a book. Therefore, even though, theoretically, we could compare the top-ranked candidate with approaches that match entity types, similarly to the content score evaluation, to the best of our knowledge, no ground truth contains a ranking of entity type candidates that evaluates their interoperability and it is unlikely to exist since, as mentioned, the best candidate is often use case dependent. Furthermore, interoperability is not focused on improving the ranking based on precision and most ground truths contain evaluations that are focused on that metric. Instead, interoperability is focused on maximising connectivity with the knowledge graph.

Therefore, in this section, we present an analysis and discussion of the effects the interoperability score has on the rankings produced by the framework. We present an experiment that compares the data model element candidates scores at different stages to analyse the impact of the content score (CS) and the interoperability score (IS) in the overall ranking of candidates. We include a final score F_c which is the combination of the content score with the interoperability score, following the function $F_c = pow(cs, w_{cs}) \cdot pow(is, w_{is})$, similarly to the function *scoring* from Algorithm 8, with weights of 1.0 for both scores. For this experiment, we show the rankings up to $d_{max} = 10$ but calculate F_c with the median, M, of the interoperability scores, IS, which is equivalent to $d_{max} = 5$. For each experiment, we present examples representative[18] of the diverse results we obtained that illustrate the points requiring discussion.

Figures 11 through 16 show the rankings for representative input entity type examples in the library use case datasets according to different measures and parameters. The figures show on the Y-axis the top-10 candidates according to F_c. On the X-axis we find CS which is the ranking of the candidate according only to the content score, then neighbourhood distances d from 0 (only the candidate) to 10, then the ranking according to the median M of the scores and, finally, the ranking according to F_c.

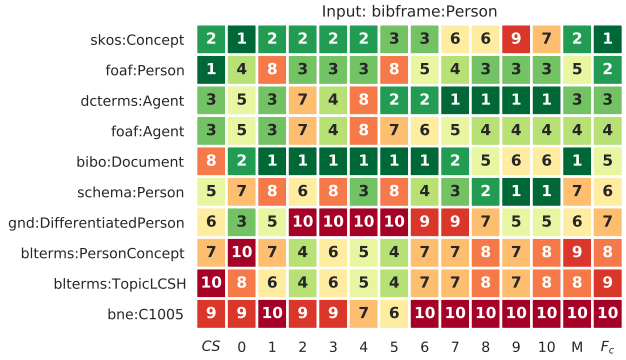

Figure 11: Rankings for entity type in the University dataset.

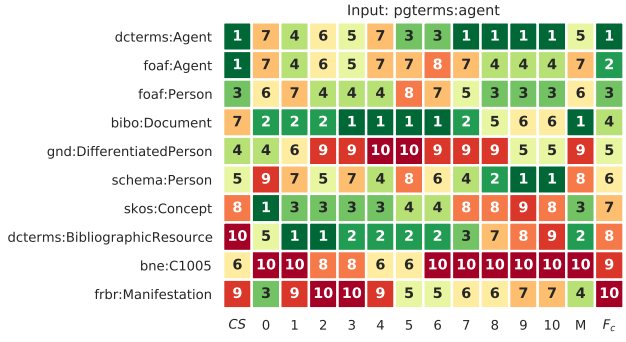

Figure 12: Rankings for entity type in the Gutenberg dataset.

Figures 11 and 12 include entity type candidates for author entities in the University and Gutenberg datasets, respectively. In the University heatmap, we observe that `foaf:Person` was the highest-ranked candidate according to CS in both datasets. However, when the content score is

[18]The extended results are available at https://figshare.com/s/cdfbd624e4e451d7ac25

combined with the interoperability score, the highest-ranked candidate changes to `skos:Concept` in the University dataset and `dcterms:Agent` in the Gutenberg dataset. `dcterms:Agent` is more generic than `foaf:Person`, therefore, in terms of interoperability, it is more desirable since it is more likely to link to more entity types in the ontology graph. In the University dataset, however, it obtained a lower raw content score, CS, which leads to its lower ranking. In this dataset, `skos:Concept` combined a strong content score with high interoperability and, therefore, maintained the top-ranked position.

When looking at the ranking of `bibo:Document` in both datasets, we verify that the interoperability boosted an undesirable candidate since this entity type is well-connected and one of the most frequent in the knowledge graph (used by all books in the German Library). Therefore, this is a case where interoperability is enabling the introduction of noise in the ranking. These cases can be counteracted by balancing the weights given to the content and interoperability scores.

Figures 13 and 14 show the results of the same experiment for two datatype properties in the Institute and OpenL datasets, respectively. In Figure 13, the input property has book titles as values. Despite having similar features as the previous datasets, due to the low amount of samples extracted from this dataset, we observe that none of the candidates ranked in the top-10 is an obvious match. The low sample size leads to poor performance of the classification model, which predicts properties incorrectly and translates to Borda scores that are not in line with the best candidates for the input property. On the other hand, Figure 14 shows that the classification model and the content score can produce good results in the top-10 that are further re-ordered according to the interoperability score. This produces candidate dates that are more interoperable even if less specific. For example, `dcterms:date` is arguably correct, even if `gnd:dateOfBirth` is more specific to the date in question. Nonetheless, once again, it is up to the data publisher to decide between the more specific or more interoperable candidates.

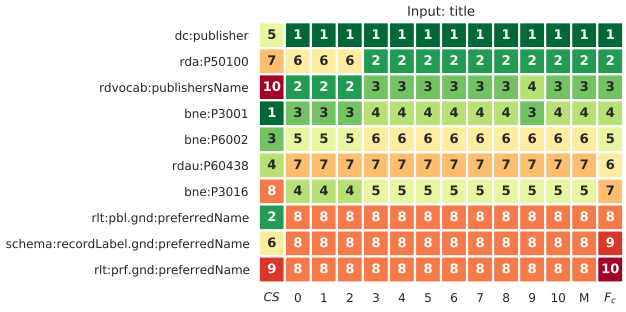

Figure 13: Rankings for datatype property in the Institute dataset.

Figures 15 and 16 show the results of this experiment for two object properties in the University and Gutenberg datasets, respectively. For the `bibframe:subject` property in Figure 15, the top-3 properties are arguably considered correct (`bne:OP3008` - has subject). In this instance, the top-3 candidates have similar interoperability scores and, therefore, there were not many changes regarding the content score ranking. It is also likely that no more correct matches exist since both the German and British libraries use `dcterms:subject`, and the remaining two properties are used by the Spanish and French libraries. The Portuguese library does not use this property, and therefore, the top-3 cover the best candidates in each library.

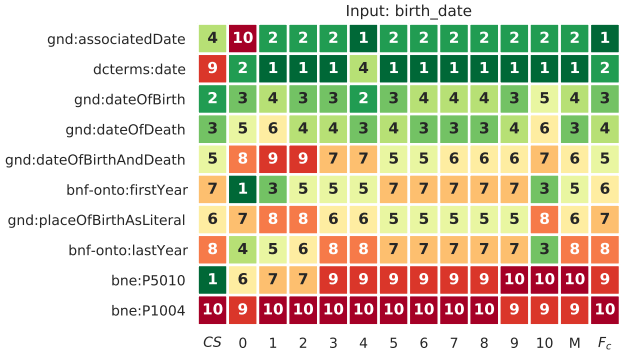

Figure 14: Rankings for datatype property in the OpenL dataset.

In Figure 16, the top ranked candidates are not so fitting since `dcterms:subject` was ranked second, with the same candidate as the input (`dcterms:creator`) ranked third due to its lower interoperability. This is another case of ranking non-relevant candidates higher than more accurate candidates. However, this reduction in precision is tolerable in cases where interoperability with multiple data sources is desirable, the data publisher still has the remaining relevant candidates further in the hierarchy and a fast validation of the output recommendation will yield more accurate and interoperable results.

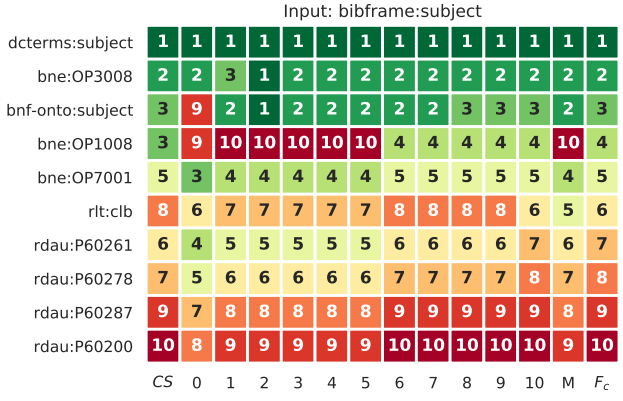

Figure 15: Rankings for object property in the University dataset.

7.2.4. Discussion

For the interoperability score, the choice of d_{max} is the parameter to take into account, which is also use case dependent. At $d_{max} = 1$, the interoperability considers only direct neighbours which include only subclasses, equival-

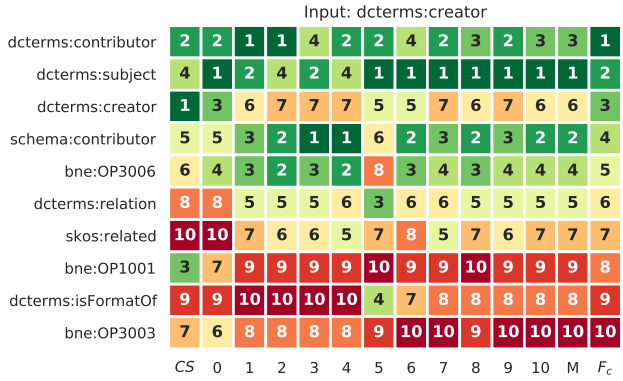

Figure 16: Rankings for object property in the Gutenberg dataset.

ences, and perfect mappings. As d_{max} increases, the neighbourhood of the candidate is expanded to include more related entity types, potentially introducing more noise into the neighbourhood. Therefore, the user's choice is a balance between specificity and maximisation of interoperability. In our experiments, we used a $d_{max} = 10$ to present a broader view of the impact of interoperability but empirically chose to use the median of all IS scores (i.e., maximum weighted distance of 5) to combine with the CS score. This median distance represents a middle point between using the generic values found at distance 10 when the interoperability score stabilises and considering a significant distance from the source to boost candidates that are well-connected in the graph.

7.3. Consistency Score

The consistency score takes as input the results from the individual scoring methodologies of entity types and properties and re-ranks them considering their combined frequency in the KG but also the homogeneity of the candidate data models being proposed. Therefore, the consistency score has two main phases: (1) score aggregation, which combines the individual scores and Knowledge Graph frequencies into a single triple candidate score, and (2) score refinement, which iterates over the candidate triples and boosts triples that improve the consistency of the entity types and properties being suggested considering the whole candidate data model.

Figure 17 shows potential results from the consistency score aggregation and refinement. The aggregation step obtains a ranked set of triples to match input triples. This step takes into account not only the individual scores of the triple elements but also the frequency that the three elements appear together on the knowledge graph. The ranked triples are then refined to boost triples that include elements that are more commonly ranked first. In this example, we see that for the bottom triple, the triples with the class Person are ranked higher than the triples with the class Author. Therefore, in the refinement step, even though the triple ⟨Book – has author → Author⟩ was ranked higher, the triple ⟨Book – has author → Person⟩ is boosted to the top rank because Person is highly ranked in other triples. The final recommended data model is presented to the user in the format presented in this figure.

7.3.1. Score Aggregation

In the score aggregation phase, for each triple pattern in the input data, we obtain the individual content and interoperability scores of each element and combine them with their co-occurrence scores extracted from the KGP map constructed (see Section 5.3). The co-occurrence score corresponds to the frequency in which elements of the triple co-occur in a triple pattern of the KGP. We compute the co-occurrence frequencies of the candidate triples ⟨domain – property → range⟩ dpr and the pairs domain-property dr, domain-range dr, and property-range pr. Therefore, the score aggregation is computed for each input triple pattern ⟨d – p → r⟩ in the input dataset by considering the individual scores CS_e and IS_e of the element e, where e is an element of the triple pattern candidate $\{c_d, c_p, c_r\} \Leftrightarrow c \in C$. The individual scores CS_e and IS_e are obtained with the specific methodologies per resource described in sections 7.1.2 and 7.2.2.

Overall, we normalise all scores using the following:

$$\widetilde{s}(e) = \frac{s(e)}{\max\limits_{e \in E} s(e)} \tag{11}$$

Where E is the set of all elements for which $s(e)$ can be computed. Whenever the tilde character ($\widetilde{}$) is used, it represents a value normalised by the maximum of the set.

We then compute the mean of the co-occurrences of the triple patterns co_r with:

$$co\_mean(d, p, r) = mean(\widetilde{co}_{dpr}, \widetilde{co}_{dp}, \widetilde{co}_{pr}, \widetilde{co}_{dr}) \tag{12}$$

Algorithm 8 in the supplementary material shows the complete methodology to compute the score aggregation phase of the consistency score.

This algorithm first defines the general scoring function that is used to initialise the aggregated score and is also used in the score refinement phase to recompute the aggregated score after refinement.

The aggregation phase begins by extracting the individual scores of each of the candidates that match the triple pattern. The algorithm then computes the Cartesian product between the three sets of candidates (line 19 in Algorithm 8) and obtains the co-occurrence score co, which is used, in addition to the individual score, to compute the final aggregated score agg. The final step uses the function $sortby(C, agg)$ to sort the candidates in descending order of agg score per d, p, r key in the map.

7.3.2. Score Refinement

In the score refinement phase, we iterate over all candidate triples that share elements, i.e., the same input subject, predicate, or object, and adjust the individual score of the overlapping resource by boosting or penalising it depending on its ranking in the different overlapping triples. We use the Borda score (Section 7.1.1) to

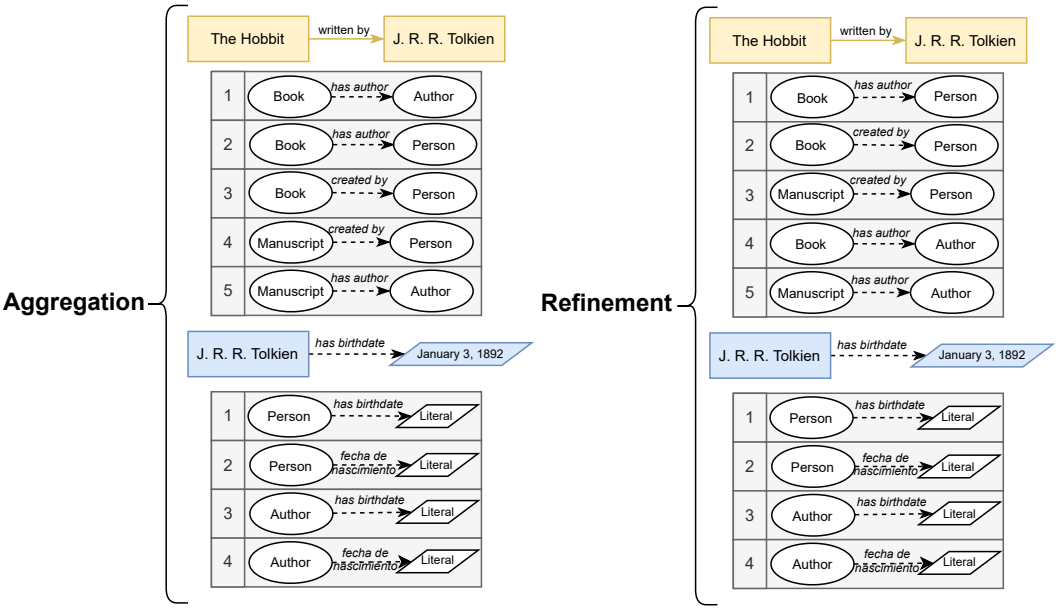

Figure 17: Example of consistency score ranking

consider the rankings of the candidates when refining the score. For example, considering two triple patterns in the input data: [pgterms:ebook, dcterms:creator, pgterms:agent] and [pgterms:ebook, dcterms:publisher, literal], the refinement phase would look at the rankings of the candidates for pgterms:ebook in both triples and adjust the score according to the rankings of the entity types in both triples. This is an iterative process that runs until the data model converges or up to a maximum of n iterations.

Algorithm 9 in the supplementary material shows the complete methodology to compute the score refinement phase of the consistency score.

First, the algorithm computes the Borda score for each candidate of each element. For that, it uses the function $groupby(DM, e)$ to group key-value pairs in DM, where the element e is a specific component of the $\langle d, p, r\rangle$ key. For example, $groupby(DM,$ pgterms:ebook$)$ gets all the triple patterns and their candidate lists that have the subject pgterms:ebook. The Borda score is then used to refine the individual scores of each candidate and the function $scoring()$ computes an updated aggregated score agg. Finally, we reorder the candidates in descending order of the new agg score. This process is repeated until the refinement does not change the order of the DM mapping anymore or until m iterations are reached.

7.3.3. Consistency Experiments & Results

Figure 18 shows a representative example[19] for the Gutenberg library of the effects of the consistency score and the weights of each score in the candidate rankings for the input triple $\langle$pgterms:ebook – dcterms:creator ↠ pgterms:agent$\rangle$.

[19]The extended results are available at https://figshare.com/s/cdfbd624e4e451d7ac25

The F_c rankings represent the weighted average of the content score CS and the interoperability score IS of the elements of the triple. Agg represents the score obtained after the aggregation step of the consistency score algorithm. Finally, Ref represents the rankings after the refinement step of the consistency score.

In Figure 18a, we show weights that we empirically found to work well for this use case, presenting triple recommendations that seemed potentially good candidates. In this example, the top-ranked candidate triple is not changed in the refinement step, however, the remaining triples are significantly changed by the consistency steps. Generally triples with bibo:Document as domain and dcterms:Agent as range are ranked higher since these were the triples that obtained higher scores and, therefore, are more frequently ranked first. The predicate dcterms:subject ranked above other more desirable candidates due to its high interoperability. This is another case where a different balance between the weights given to interoperability and content score could positively affect the final results. The remaining candidates as boosted by co-occurrence frequency and the score refinement to appear in the top-20 recommended triples for the input triple pattern. The remaining candidates follow similar patterns, where despite obtaining lower scores when combining content and interoperability metrics, the triple score is boosted by co-occurrence in the Agg step, and further refined in the last step. This pattern is also verified for all remaining triple patterns of the datasets of both use cases. Previous issues are not resolved in this step, but when the rankings follow the expected outcomes, the resulting rankings follow a similar pattern to the example in Figure 18.

Figure 18b shows the impact of giving equal weight to content and interoperability scores. Here we see that the Aggregation step changes the ranking based on F_c and the Re-

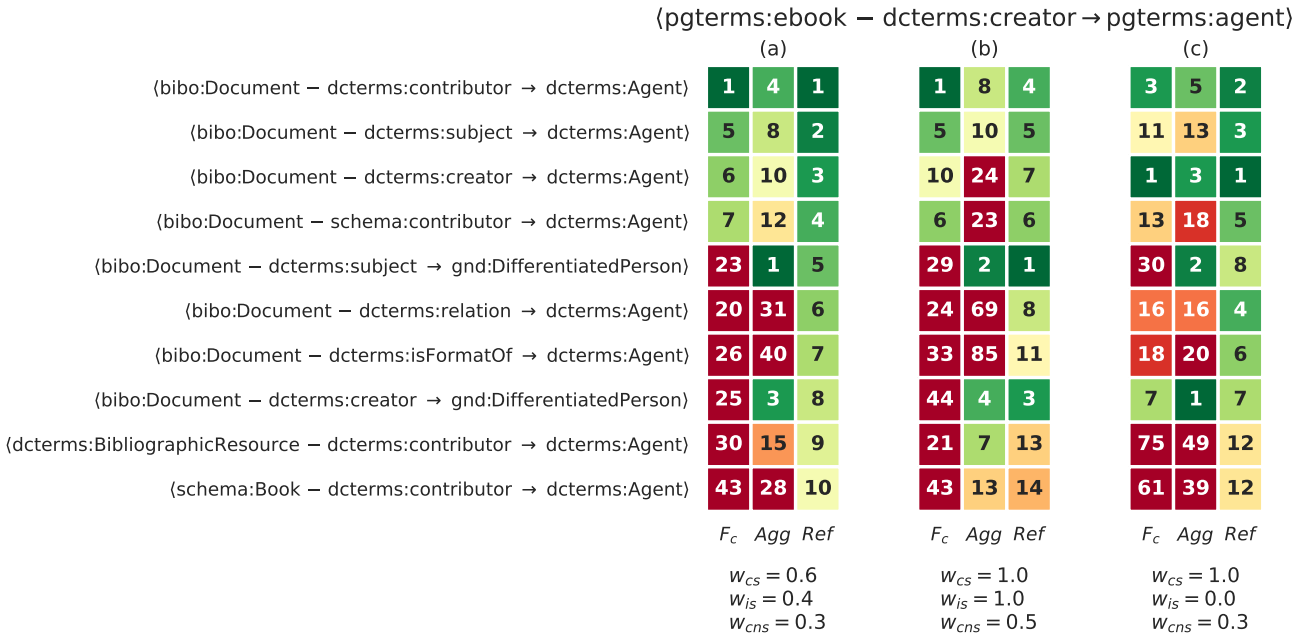

Figure 18: Ranking of triple candidates according to final score F_c, score aggregation Agg, and score refinement Ref.

finement step leads to the wrong triple being ranked higher than others that are potentially more correct. Figure 18c shows the result of the recommendation if the interoperability is given no weight. In this case, a more specific triple is ranked higher after aggregation but the refinement steps leads the recommendation to the more general triples.

7.3.4. Discussion

For the chosen use cases, we observed that the consistency methodology fulfilled its purpose and led to more consistent data models in terms of frequency in the knowledge graph and the candidates being suggested within the data model.

A feature of this score is that it combines the previous scores with the co-occurrence scores to obtain a final triple score. This combination is achieved by balancing parameters that should align with the preferences of the user. In our experiments, we found the weights 0.6 for w_{cs}, 0.4 for w_{is} and 0.3 for w_{cns} to be the most balanced, however, these might not be optimal for different use cases. In the case of the library data, we obtained reasonable results for the final rankings, but it is not guaranteed that it will be similar for other datasets. Besides, these parameters are meant to be adjusted to the preferences of the users. Therefore, the user can focus on the framework by boosting the content, interoperability, or consistency scores. However, the user should also consider that the strength of this framework lies not in achieving the highest precision possible for each candidate but, instead is focused on producing a ranking of candidate triples that will be accurate, consistent, and interoperable. Maximising one of these scores, while ignoring the others will produce results that might be less optimal.

7.4. Evaluating Distance to Source

The final experiment focuses on comparing the ranking obtained by our framework with the original entity types from library KG datasets. This evaluation was obtained by taking each dataset, excluding their entities from the document store, and running the framework to produce a set of entity type candidates.

We use the term *lenient* to refer to a metric that does not use a binary classification but considers the distance in the graph to make an assessment. Therefore, we distinguish two types of evaluation: (1) *strict* where the correctness of the rank is binary, and (2) *lenient* which considers the distance from the source type $t \in T$ to the candidate type $c \in C$.

This evaluation uses Precision@k (P@k), where $k \in \{1, 3, 5\}$. The lenient evaluation instead of binary correctness considers the distance in the ontology graph G_e. We compute lenient precision by considering correct candidates at three distances $d \in \{1, 2, 3\}$, i.e., any candidate at a distance less or equal to d is considered correct. We calculate these metrics for every input entity type in each library, average the results, and present the aggregated result in Table 10.

The strict evaluation obtained poor performance in all datasets. This result was expected since the removal of data from the document store has a high impact on both CS and IS due to frequency counts being lowered in the search results and document store for the original entity types. Therefore, the likelihood of our framework suggesting the same entity type as the original is low. In the lenient evaluation, we verify that the entity type candidates suggested are close to the original types. Overall, the interoperability score does not seem to drive the candidates away from the entity type originally chosen to model the data. We observe slight vari-

Table 10
Results of the strict and lenient evaluation of the distance from source.

Library	Score	Strict			Lenient								
		$P@1$	$P@3$	$P@5$	$P@1$			$P@3$			$P@5$		
		d_0	d_0	d_0	d_1	d_2	d_3	d_1	d_2	d_3	d_1	d_2	d_3
British	CS	**0.037**	0.025	0.022	0.370	0.852	0.926	0.272	0.753	0.926	0.200	0.704	0.904
	IS	0.000	0.037	0.022	0.370	0.889	0.963	0.284	0.728	0.926	0.230	0.674	0.881
French	CS	0.000	0.028	0.033	0.417	0.833	0.917	0.306	0.694	0.972	0.333	0.717	0.983
	IS	0.000	0.028	0.033	0.417	0.833	1.000	**0.472**	0.722	0.972	**0.433**	0.700	0.950
German	CS	0.000	0.000	0.007	0.152	0.848	**1.000**	0.151	0.742	0.989	0.167	0.727	0.980
	IS	0.000	0.000	0.007	0.182	**0.909**	**1.000**	0.237	0.785	0.978	0.213	**0.747**	0.980
Portuguese	CS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	IS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Spanish	CS	0.000	**0.062**	**0.038**	**0.500**	0.833	**1.000**	0.312	**0.812**	**1.000**	0.231	0.731	**1.000**
	IS	0.000	**0.062**	**0.038**	**0.500**	0.833	**1.000**	0.312	0.625	**1.000**	0.231	0.731	**1.000**

ations in the IS score in comparison to the CS score, however, in most cases, this variation is small enough to claim that the interoperability score does not degrade the recommendation of the framework. The German library has one of the lowest performances because they adopted a data model that is structurally distant from the ones used in the other libraries. However, the Portuguese library obtains a score of 0 in every metric. This is due to the simple nature of their data model and the lack of general interoperability with the remaining models. Despite the low performance in this experiment, however, the Portuguese library data was still part of the knowledge graph and might use a desirable data model for other users.

This evaluation, together with the previous experiments, demonstrates that our framework proposes candidates that are potentially more interoperable with existing domain datasets while maintaining the original meaning intended in the library datasets by proposing similar entity types.

8. Conclusions

RICDaM is a framework composed of several methods and algorithms to produce a ranked set of candidates to match the data of an input dataset. This framework is divided into (1) building the background knowledge graph from multiple data sources and the ontology graph extracted from those sources, and (2) generating and ranking entity type and property candidates by exploiting information obtained from the knowledge graph and the ontology graph.

We experimented and evaluated the framework with diverse methods to examine its performance and analyse its impact on potential applications. Through the evaluations and experiments, we concluded that, within the set parameters, the framework achieves the goal of facilitating the process of creating a data model for an input dataset that can be adjusted to be accurate, interoperable, and consistent. This process is facilitated by proposing a ranked list of candidates that orders candidates based on the relevancy to the input and their interoperability with other concepts in the knowledge graphs. The caveats of each methodology proposed were discussed and should be taken into consideration when implementing the framework and associated evaluations. Overall, one of the goals of the framework was to lower the entry-level barrier to publish linked data. In our proposed framework, most of the underlying processes of identifying the best class or property to model the data are automatised and present the user with a recommendation of a data model for their data. Nonetheless, the user is presented with a ranked set of options and can customise to model to fit a certain use case. This customisation can require some level of understanding of the data model and the underlying concepts of the RDF data model. Therefore, despite being able to streamline the process of creating a data model, the publishing barriers are not eliminated since some level of knowledge is still required for an optimal experience. On the other hand, the framework is more successful in aiding in selecting and linking the ontologies used to model data in a domain. The framework takes existing datasets and provides interoperable and consistent recommendations for data models and eliminate the need to manually analyse the data domain and empirically decide between existing data models.

In the future, we will apply this to more domains with potentially more complex data, such as the biomedical domain. This domain is characterised by very large data with complexities specific to the domain. Preliminary work in this domain was conducted [28]. However, further testing is necessary to accurately analyse the effectiveness of the methodology in this domain.

The consistency score can consider the consistency in terms of independent data sources in the knowledge graph. Currently, the knowledge graph is treated by the framework as a single source. However, if it distinguishes between data sources, it is possible to boost the patterns that exist within the same source data model, ensuring that logical coherence is maintained from the original data source. In this sense, depending on which data source is being mostly ranked first, the framework would boost the triple candidates that belong to that source. For example, if candidate triples from the British library are frequently ranked first, the framework

would boost candidates from that library to improve the consistency of the model at the data source level.

Another consideration in terms of the weighting scheme for the scores is that, in the future, these should not be taken globally for each entity in the triple candidate. Data publishers can have different requirements for the data model elements, considering, for example, that entity types should focus more on interoperability, while properties should be as accurate as possible. Therefore, in the future, these weights should be expanded to provide more fine-grained control over the weights given to the scoring methodologies.

A demonstration of an implementation is available at http://afel.insight-centre.org/ricdam/ [29]. The demonstration includes two pre-loaded outputs: one for the Gutenberg dataset and the other for the Open Library, and presents the top candidate data model to the user. The interface gives an overview of the best-ranked candidates for each triple but also allows the user to adapt the data model to their preference and use case. The demonstration presents an overview of the output of the framework, allows the customisation of the data model and tuning of the parameters to produce different candidate rankings. In the future, a complete tool using this framework would allow the user to search the indexed ontologies or add new ontologies to the graph to complete the model when the data model fails to find the desired class. For JSON or CSV input datasets, it would also be possible to generate an RML mapping file to facilitate the process of translating the data to RDF.

Acknowledgements

This research was supported by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289 and SFI/12/RC/2289_P2, co-funded by the European Regional Development Fund and the Fundação para a Ciência e a Tecnologia through the LASIGE Research Unit, UIDB/00408/2020 and UIDP/00408/2020.

References

[1] Adelfio, M.D., Samet, H., 2013. Schema Extraction for Tabular Data on the Web. Proceedings of the VLDB Endowment 6, 421–432. URL: http://dl.acm.org/doi/10.14778/2536336.2536343, doi:10.14778/2536336.2536343.

[2] Algergawy, A., Faria, D., Ferrara, A., Fundulaki, I., Harrow, I., Hertling, S., Jimenez-Ruiz, E., Karam, N., Khiat, A., Lambrix, P., Li, H., Montanelli, S., Paulheim, H., Pesquita, C., Saveta, T., Shvaiko, P., Splendiani, A., Thieblin, E., Trojahn, C., Vataščinova, J., Zamazal, O., Zhou, L., 2019. Results of the Ontology Alignment Evaluation Initiative 2019, in: Proceedings of the 14th International Workshop on Ontology Matching co-located with ISWC 2019, CEUR Workshop Proceedings, Auckland, New Zealand. p. 40.

[3] Andresen, L., 2004. After MARC – what then? Library Hi Tech 22, 40–51. URL: https://doi.org/10.1108/07378830410524486, doi:10.1108/07378830410524486.

[4] Chen, Z., Jia, H., Heflin, J., Davison, B.D., 2018. Generating Schema Labels through Dataset Content Analysis, in: Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18, ACM Press, Lyon, France. pp. 1515–1522. URL: http://dl.acm.org/citation.cfm?doid=3184558.3191601, doi:10.1145/3184558.3191601.

[5] d'Aquin, M., Adamou, A., Dietze, S., 2013. Assessing the educational linked data landscape, in: Proceedings of the 5th Annual ACM Web Science Conference, Association for Computing Machinery, Paris, France. pp. 43–46. URL: https://doi.org/10.1145/2464464.2464487, doi:10.1145/2464464.2464487.

[6] Das, S., Sundara, S., Cyganiak, R., 2012. R2RML: RDB to RDF Mapping Language. URL: https://www.w3.org/TR/r2rml/.

[7] Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R., 2014. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data., in: LDOW. URL: https://www.researchgate.net/profile/Ruben_Verborgh/publication/264274087_RML_A_Generic_Language_for_Integrated_RDF_Mappings_of_Heterogeneous_Data/links/53d8fd2b0cf2631430c38a7b.pdf.

[8] Do, B.L., Aryan, P.R., Trinh, T.D., Wetz, P., Kiesling, E., Tjoa, A.M., 2015. Toward a framework for statistical data integration, in: Proceedings of the 3rd International Workshop on Semantic Statistics, CEUR Workshop Proceedings, Bethlehem, U.S.A.. p. 12.

[9] Ehrlinger, L., Wöß, W., 2016. Towards a Definition of Knowledge Graphs, in: Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems and the 1st International Workshop on Semantic Change & Evolving Semantics, CEUR-WS, Leipzig, Germany.

[10] Faria, D., Pesquita, C., Santos, E., Palmonari, M., Cruz, I.F., Couto, F.M., 2013. The AgreementMakerLight Ontology Matching System, in: On the Move to Meaningful Internet Systems: OTM 2013 Conferences, Springer, Berlin, Heidelberg. pp. 527–541. URL: https://link.springer.com/chapter/10.1007/978-3-642-41030-7_38, doi:10.1007/978-3-642-41030-7_38.

[11] Faria, D., Pesquita, C., Tervo, T., Couto, F.M., Cruz, I.F., 2019. AML and AMLC Results for OAEI 2019, in: Proceedings of the 14th International Workshop on Ontology Matching co-located with ISWC 2019, CEUR-WS. p. 6.

[12] Gruber, T.R., 1993. A translation approach to portable ontology specifications. Knowledge Acquisition 5, 199–220. URL: http://www.sciencedirect.com/science/article/pii/S1042814383710083, doi:10.1006/knac.1993.1008.

[13] Haesendonck, G., Maroy, W., Heyvaert, P., Verborgh, R., Dimou, A., 2019. Parallel RDF generation from heterogeneous big data, in: Proceedings of the International Workshop on Semantic Big Data - SBD '19, ACM Press, Amsterdam, Netherlands. pp. 1–6. URL: http://dl.acm.org/citation.cfm?doid=3323878.3325802, doi:10.1145/3323878.3325802.

[14] Hallo, M., Luján-Mora, S., Maté, A., Trujillo, J., 2016. Current state of Linked Data in digital libraries. Journal of Information Science 42, 117–127. URL: https://doi.org/10.1177/0165551515594729, doi:10.1177/0165551515594729.

[15] Hertling, S., Paulheim, H., 2020. DBkWik: extracting and integrating knowledge from thousands of Wikis. Knowledge and Information Systems 62, 2169–2190. URL: https://doi.org/10.1007/s10115-019-01415-5, doi:10.1007/s10115-019-01415-5.

[16] Heyvaert, P., De Meester, B., Dimou, A., Verborgh, R., 2018. Declarative Rules for Linked Data Generation at Your Fingertips!, in: Gangemi, A., Gentile, A.L., Nuzzolese, A.G., Rudolph, S., Maleshkova, M., Paulheim, H., Pan, J.Z., Alam, M. (Eds.), The Semantic Web: ESWC 2018 Satellite Events, Springer International Publishing, Cham. pp. 213–217. doi:10.1007/978-3-319-98192-5_40.

[17] Hu, W., Qiu, H., Dumontier, M., 2015. Link Analysis of Life Science Linked Data, in: Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (Eds.), International Semantic Web Conference, Springer International Publishing, Cham. pp. 446–462. doi:10.1007/978-3-319-25010-6_29.

[18] Iglesias-Molina, A., Chaves-Fraga, D., Priyatna, F., Corcho, O., 2019. Enhancing the Maintainability of the Bio2RDF Project Using Declarative Mappings, Edinburgh, Scotland. p. 11.

[19] Krötzsch, M., Thost, V., 2016. Ontologies for Knowledge Graphs: Breaking the Rules, in: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (Eds.), International Semantic Web Conference, Springer, Cham. pp. 376–392. URL: http://link.springer.com/10.1007/978-3-319-46523-4_23,

doi:10.1007/978-3-319-46523-4_23.

[20] Lefrançois, M., Zimmermann, A., Bakerally, N., 2017. A SPARQL Extension for Generating RDF from Heterogeneous Formats, in: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (Eds.), The Semantic Web, Springer International Publishing, Cham. pp. 35–50. URL: http://link.springer.com/10.1007/978-3-319-58068-5_3, doi:10.1007/978-3-319-58068-5_3. series Title: Lecture Notes in Computer Science.

[21] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C., 2015. DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web 6, 167–195. URL: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-140134, doi:10.3233/SW-140134.

[22] Lehmann, J., Schuppel, J., Auer, S., 2007. Discovering Unknown Connections – the DBpedia Relationship Finder, in: Proceedings of the 1st Conference on Social Semantic Web (CSSW), Leipzig, Germany. p. 11.

[23] Limaye, G., Sarawagi, S., Chakrabarti, S., 2010. Annotating and searching web tables using entities, types and relationships. Proceedings of the VLDB Endowment 3, 1338–1347. URL: http://dl.acm.org/doi/10.14778/1920841.1921005, doi:10.14778/1920841.1921005.

[24] McKenna, L., Debruyne, C., O'Sullivan, D., 2018. Understanding the Position of Information Professionals with regards to Linked Data: A Survey of Libraries, Archives and Museums, in: Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries, Association for Computing Machinery, New York, NY, USA. pp. 7–16. URL: https://doi.org/10.1145/3197026.3197041, doi:10.1145/3197026.3197041.

[25] de Medeiros, L.F., Priyatna, F., Corcho, O., 2015. MIRROR: Automatic R2RML Mapping Generation from Relational Databases, in: Cimiano, P., Frasincar, F., Houben, G.J., Schwabe, D. (Eds.), Engineering the Web in the Big Data Era. Springer International Publishing, Cham. volume 9114, pp. 326–343. URL: http://link.springer.com/10.1007/978-3-319-19890-3_21, doi:10.1007/978-3-319-19890-3_21. series Title: Lecture Notes in Computer Science.

[26] Michel, F., Djimenou, L., Faron Zucker, C., Montagnat, J., 2015. Translation of Relational and Non-Relational Databases into RDF with xR2RML, in: 11th International Confenrence on Web Information Systems and Technologies (WEBIST'15), Lisbon, Portugal. pp. 443–454. URL: https://hal.archives-ouvertes.fr/hal-01207828, doi:10.5220/0005448304430454.

[27] Neumaier, S., Umbrich, J., Parreira, J.X., Polleres, A., 2016. Multi-level Semantic Labelling of Numerical Values, in: Proceedings of the 15th International Semantic Web Conference, Springer International Publishing, Kobe, Japan. pp. 428–445. URL: http://link.springer.com/10.1007/978-3-319-46523-4_26, doi:10.1007/978-3-319-46523-4_26. series Title: Lecture Notes in Computer Science.

[28] Oliveira, D., 2021. Generating and ranking candidate data models from background knowledge. Thesis. NUI Galway. URL: https://aran.library.nuigalway.ie/handle/10379/16394. accepted: 2021-01-04T11:09:07Z.

[29] Oliveira, D., d'Aquin, M., 2020. RICDaM: Recommending Interoperable and Consistent Data Models, in: Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020), CEUR-WS.org. p. 5. URL: http://ceur-ws.org/Vol-2721/paper535.pdf.

[30] Oliveira, D., Sahay, R., d'Aquin, M., 2019. Leveraging Ontologies for Knowledge Graph Schemas, in: Proceedings of the 1st Workshop on Knowledge Graph Building co-located with ESWC 2019, CEUR-WS.org, Portoroz, Slovenia. pp. 24–36. URL: http://ceur-ws.org/Vol-2489/paper3.pdf.

[31] Park, H., Kipp, M., 2019. Library Linked Data Models: Library Data in the Semantic Web. Cataloging & Classification Quarterly 57, 261–277. URL: https://doi.org/10.1080/01639374.2019.1641171, doi:10.1080/01639374.2019.1641171.

[32] Paulheim, H., Bizer, C., 2013. Type Inference on Noisy RDF Data, in: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Salinesi, C., Norrie, M.C., Pastor, O. (Eds.), Advanced Information Systems Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg. volume 7908, pp. 510–525. URL: http://link.springer.com/10.1007/978-3-642-41335-3_32, doi:10.1007/978-3-642-41335-3_32. series Title: Lecture Notes in Computer Science.

[33] Pereira Nunes, B., Dietze, S., Casanova, M.A., Kawase, R., Fetahu, B., Nejdl, W., 2013a. Combining a Co-occurrence-Based and a Semantic Measure for Entity Linking, in: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (Eds.), The Semantic Web: Semantics and Big Data. Springer Berlin Heidelberg, Berlin, Heidelberg. volume 7882, pp. 548–562. URL: http://link.springer.com/10.1007/978-3-642-38288-8_37, doi:10.1007/978-3-642-38288-8_37.

[34] Pereira Nunes, B., Mera, A., Casanova, M.A., Fetahu, B., P. Paes Leme, L.A., Dietze, S., 2013b. Complex Matching of RDF Datatype Properties, in: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (Eds.), Database and Expert Systems Applications, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 195–208. URL: http://link.springer.com/10.1007/978-3-642-40285-2_18, doi:10.1007/978-3-642-40285-2_18. series Title: Lecture Notes in Computer Science.

[35] Saari, D.G., 1994. Geometry of Voting. Springer Berlin Heidelberg, Berlin, Heidelberg. OCLC: 903196450.

[36] Sabou, M., d'Aquin, M., Motta, E., 2008. SCARLET: SemantiC RelAtion DiscoveRy by Harvesting OnLinE OnTologies, in: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (Eds.), The Semantic Web: Research and Applications, Springer, Berlin, Heidelberg. pp. 854–858. doi:10.1007/978-3-540-68234-9_72.

[37] Seo, D., Koo, H.K., Lee, S., Kim, P., Jung, H., Sung, W.K., 2011. Efficient Finding Relationship between Individuals in a Mass Ontology Database, in: Kim, T.h., Adeli, H., Ma, J., Fang, W.c., Kang, B.H., Park, B., Sandnes, F.E., Lee, K.C. (Eds.), U- and E-Service, Science and Technology, Springer, Berlin, Heidelberg. pp. 281–286. doi:10.1007/978-3-642-27210-3_37.

[38] Sleeman, J., Finin, T., Joshi, A., 2015. Entity Type Recognition for Heterogeneous Semantic Graphs. AI Magazine 36, 75–86. URL: https://www.aaai.org/ojs/index.php/aimagazine/article/view/2569, doi:10.1609/aimag.v36i1.2569. number: 1.

[39] Smith-Yoshimura, K., 2016. Analysis of International Linked Data Survey for Implementers. D-Lib Magazine 22. URL: http://www.dlib.org/dlib/july16/smith-yoshimura/07smith-yoshimura.html, doi:10.1045/july2016-smith-yoshimura.

[40] Smith-Yoshimura, K., 2018. Analysis of 2018 International Linked Data Survey for Implementers. The Code4Lib Journal URL: https://journal.code4lib.org/articles/13867.

[41] Syed, Z., Finin, T., Mulwad, V., Joshi, A., 2010. Exploiting a Web of Semantic Data for Interpreting Tables, in: Proceedings of the Second Web Science Conference, Raleigh, NC, USA.

[42] Tennant, R., 2004. A bibliographic metadata infrastructure for the twenty-first century. Library Hi Tech 22, 175–181. URL: https://www.emerald.com/insight/content/doi/10.1108/07378830410524602/full/html, doi:10.1108/07378830410524602.

[43] Tonon, A., Catasta, M., Prokofyev, R., Demartini, G., Aberer, K., Cudré-Mauroux, P., 2016. Contextualized ranking of entity types based on knowledge graphs. Journal of Web Semantics 37-38, 170–183. URL: https://linkinghub.elsevier.com/retrieve/pii/S1570826815001468, doi:10.1016/j.websem.2015.12.005.

[44] Ullah, I., Khusro, S., Ullah, A., Naeem, M., 2018. An Overview of the Current State of Linked and Open Data in Cataloging. Information Technology and Libraries 37, 47–80. URL: https:

//ejournals.bc.edu/index.php/ital/article/view/10432, doi:10.6017/ital.v37i4.10432.

[45] Venetis, P., Halevy, A., Madhavan, J., Paşca, M., Shen, W., Wu, F., Miao, G., Wu, C., 2011. Recovering semantics of tables on the web. Proceedings of the VLDB Endowment 4, 528–538. URL: http://dl.acm.org/doi/10.14778/2002938.2002939, doi:10.14778/2002938.2002939.

[46] Vennesland, A., 2017. Matcher composition for identification of subsumption relations in ontology matching, in: Proceedings of the International Conference on Web Intelligence - WI '17, ACM Press, Leipzig, Germany. pp. 154–161. URL: http://dl.acm.org/citation.cfm?doid=3106426.3106503, doi:10.1145/3106426.3106503.

[47] Yi, Y., Chen, Z., Heflin, J., Davison, B.D., 2018. Recognizing Quantity Names for Tabular Data, in: Joint Proceedings of the First International Workshop on Professional Search (ProfS2018); the Second Workshop on Knowledge Graphs and Semantics for Text Retrieval, Analysis, and Understanding (KG4IR); and the International Workshop on Data Search (DATA:SEARCH'18), CEUR-WS.org, Ann Arbor, Michigan, USA. p. 6.

[48] Zamazal, O., Svátek, V., 2017. The Ten-Year OntoFarm and its Fertilization within the Onto-Sphere. Journal of Web Semantics 43, 46–53. URL: http://www.sciencedirect.com/science/article/pii/S1570826817300100, doi:10.1016/j.websem.2017.01.001.