Large Semantic Graph Summarization Using Namespaces

Ana Rita Santos Lopes da Costa
Department of Computer Science, Faculty of Science, University of Porto, Portugal

André Santos
CRACS & INESC Tec LA / Faculty of Sciences, University of Porto, Portugal

José Paulo Leal
CRACS & INESC Tec LA / Faculty of Sciences, University of Porto, Portugal

Abstract

We propose an approach to summarize large semantics graphs using namespaces. Semantic graphs based on the Resource Description Framework (RDF) use namespaces on their serializations. Although these namespaces are not part of RDF semantics, they have intrinsic meaning. Based on this insight, we use namespaces to create summary graphs of reduced size, more amenable to be visualized. In the summarization, object literals are also reduced to their data type and the blank nodes to a group of their own. The visualization created for the summary graph aims to give insight of the original large graph. This paper describes the proposed approach and reports on the results obtained with representative large semantic graphs.

2012 ACM Subject Classification Information systems → Summarization; Information systems → Resource Description Framework (RDF)

Keywords and phrases Semantic graph, RDF, namespaces, reification

Digital Object Identifier 10.4230/OASIcs.SLATE.2022.12

Supplementary Material *Software (Python Module)*: `https://pypi.org/project/rdf-summarizer/`
Software (Source Code): `https://github.com/ritasantos11/RDF-Summarizer`
archived at `swh:1:dir:a5f50e27fbce52accaec26736daa124a0d0fe728`

Funding This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020.
André Santos: Ph. D. Grant SFRH/BD/129225/2017 from Fundação para a Ciência e Tecnologia (FCT), Portugal.

1 Introduction

Semantic graphs based on the Resource Description Framework (RDF) are frequently massive, with more than a billion triples. This order of magnitude raises several problems when processing these RDF graphs: they are difficult to load into memory; querying them is time-consuming, and visualization tools are virtually useless.

Graph summaries are an approach to deal with this issue. Summaries may be a smaller sub-graph retaining only some nodes and edges; or a document in another format which summarizes the graphs content, such as statistics on the number of nodes and edges, or its structural features.

A common approach in graph summarization is to group nodes and edges of similar kind. The challenge is to identify a reduced number of groups of nodes and edges in an RDF graph so that the graph summary is both meaningful and understandable. On the one hand, the groups of nodes and edges must retain part of the meaning of their elements. On the other hand, the reduced graph must be small enough to be easily understood.

Most nodes and edges in RDF have an associated Internationalized Resource Identifier (IRI), typically an Uniform Resource Locator (URL). RDF serialization formats, such as RDF-XML and Turtle, shorten these IRIs using namespaces. Although namespaces exist only in serializations and are not part of the semantics of RDF, they have intrinsic meaning.

Namespaces reflect a common purpose or a common origin of IRIs. They are more than just alias to prefixes shared by a large set of IRIs used in a semantic graph. Similar resources described in a semantic graph typically share a common namespace. For instance, book resources in DBpedia have a common namespace[1]. Also, each vocabulary used in semantic graphs has its particular namespace, as is the case of Dublin Core or Friend-of-a-friend.

The semantics of an RDF graph has two equivalents representations: either as a set of triples whose elements are IRIs or strings; or as a labeled multi-graph whose labels are IRIs or strings. In a strict sense, namespaces are not part of RDF semantics. However this does not entail that they are devoid of meaning. Namespaces reflect a commonality of resource identifiers and vocabularies and are assigned by semantic graphs authors, not automatically generated by an algorithm.

Based on this insight, this work explores the use of namespaces and their intrinsic meaning to summarize RDF graphs. Certainly, we must also consider literals and blank nodes since IRIs are not the only kind of element in RDF triples. Thus, the proposed approach maps IRIs, literals, and blank nodes to a reduced set of identifiers to produce summary graphs retaining part of the semantics of the original graph.

Our goal is to generate meaningful summaries from semantic graphs with hundreds of millions of triples and produce them in a few hours. We want to understand the information stored in the graph that otherwise we would not be able to given the volume of the data and to visualize it in an easy way. We applied the proposed summarization method to two large graphs, with more than a million triples: the KBpedia knowledge graph and the Linked Movie Database. The first has over one million triples and the second over three million triples. We were able to produce a summary graph, from which we were able to extract visualizations and statistics.

The remainder of this paper is organized as follows. Section 2 provides background on semantic graphs and RDF. Section 3 surveys related work on semantic graph summarization. Section 4 details the proposed approach to using namespaces for semantic graph summarization. Section 5 reports on the use of this approach to summarize two RDF graphs: KBpedia and LinkedMDB (Linked Movie Database). Finally, Section 6 identifies the main contributions of this research and opportunities for future work.

2 Background

Semantic graphs store information about concepts in the nodes and the semantic relations between them in the edges. They are associated with ontologies, formal and explicit specifications of shared formalizations characterized by high semantic expressiveness required for increased complexity [11].

These type of graphs are expressed in RDF where the knowledge about a given domain is represented by triples *[subject, predicate, object]*. Each triple states that the *subject* is connected to the *object* through a relation described by the *predicate* [7]. RDF itself contains properties for relating subjects to objects, such as the *rdf:type* property, and it is also possible to create other specific properties for the domain in question. RDF graph nodes can be an Internationalized Resource Identifier (IRI), a literal or a blank node (an anonymous resource

[1] `http://purl.org/NET/book/vocab#`

for which an IRI or literal is not given). The literals can only appear in the object position of the triples and have associated a datatype that indicates what is the type of the content of that literal. It can be, for example, strings, integers or dates [16]. Blank nodes can only be used in the subject or object positions.

In some serialization formats the IRIs that identify nodes and edges can be associated with a namespace. Common prefixes of IRIs are given smaller names to identify them, allowing not to use the full IRI when referring to a node or edge. For example, in the RDF Turtle serialization format, a prefix definition would be:
`@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>`.
With this declaration, instead of using the RDF property *type* with the full IRI[2], it can be shortened to `rdf:type`.

These alias definitions are internal to each serialized document and have no impact on the semantics of the graph. Nevertheless, common namespaces (from widely used vocabularies or ontologies) are usually shortened to the same prefixes. The *prefix.cc* website[3] contains a crowd-sourced mapping of namespaces and prefixes [9], and it includes a namespace lookup.

Often in RDF there is the need to make statements about other statements. RDF*, an extension to RDF is being developed to address this issue, but it is currently only a draft [1]. RDF Reification is another approach which provides the ability to make RDF statements to express something in a language using the language, so it becomes treatable by the language [4]. In practice, it allows building a triple in which the subject is another triple. As an example, if we want to represent information about the source of the triple *[ex:Gardener ex:hasKilled ex:butler]*, we need to construct a RDF statement about it, as in Figure 1. This way, the reified triple can be referenced by others triples.

```
ex:statementExample rdf:type rdf:Statement .
ex:statementExample rdf:subject ex:Gardener .
ex:statementExample rdf:predicate ex:hasKilled .
ex:statementExample rdf:object ex:Butler .
ex:statementExample ex:saidBy ex:Nurse
```

Figure 1 Reification example.

KBpedia is a medium-sized open-source knowledge graph that combines leading public knowledge bases (Wikipedia, Wikidata, schema.org, DBpedia, GeoNames, OpenCyc, and the UNSPSC products and services) into an integrated and computable structure [3]. LinkedMDB is a RDF graph that contains linked-data-collection of movies, actors, directors and the relationships between them [10].

SPARQL Protocol and RDF Query Language (SPARQL) is a language for querying RDF graphs. Through this language, it is possible, for example, to return the properties associated with a certain class, query the instances that are part of a certain class and count the numbers of triples [15].

Graphs can be described using the DOT language. DOT is a text based graph representation which can be transformed into a diagram using tools like Graphviz [8]. Graphviz is an open source graph visualization software that allows to create nodes and edges with different sizes and colors, for example. It provides eight layout engines to draw the graph [2].

[2] `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
[3] `http://prefix.cc`

A trie or prefix tree is a tree data structure that stores strings. It allows to retrieve a string from a set in an efficient way. It supports operations of insertion and lookup. For this work, we adapted the definition of trie given in [12].

3 Related Work

Liu et al. [14] present several kinds of graph summarization methods. They distinguish between methods for static graphs that do not consider additional information to their structure and those for static graphs that contain attributes in nodes and arcs.

Static graphs with attributes can use the following summarization kinds of methods: aggregation-based, bit-compression-based, and influence-based. Aggregation methods group several nodes that share properties into a *supernode*, making the summary graph contain specific properties. Others apply clustering to map each densely connected cluster into a supernode. Bit-compression-based methods decrease the number of bits needed to store a graph. Influence-based methods discover the description of the spread of influence by formulating the summarization problem as an optimization process in which some information about the influence is kept [14].

Static graphs without attributes can also use methods based on simplification or sparsification. These methods remove less important nodes or arcs, resulting in a sparsified graph [14].

Bonifati et al. [5] present statistical and goal-oriented method types. The former is based on quantitative measurements and the occurrence count. The latter optimizes the memory footprint by producing a concise representation that fits in memory.

Specifically for semantic graphs, many of the available summarization methods are either structural or statistical. Structural methods consider structural features such as paths, graph patterns, or frequent nodes. Some of these approaches extract the most frequent graph patterns, sub-graphs that share common types and properties. Statistical methods rely on graph statistics, such as node type frequencies. We can combine these method types to obtain better results [7].

A particular type of semantic graph summarization is schema extraction. If the graph itself does not have an associated ontology, it is possible to extract a schema that acts as a summary of the graph [7]. The extracted schema provides information on the content of the graph, that is, what types and properties exist, and helps with its exploration [13].

Kellou-Menouer and Kedad [13] propose a schema with types and edge definitions based on a density clustering algorithm. This approach generates types through grouping and the description of each one of them, where each property will have an associated probability. It then generates the edges between types by analyzing the type descriptions. The resulting summary is a small graph showing the relationships between the different types of the original one. The first phase of this approach is to generate the types and their description, called type profiles, with each property having an associated probability. The following step is to generate semantic and hierarchical edges between types by analyzing type profiles. In the first phase, a density-based clustering algorithm is applied to group similar entities and create a profile for each class. These profiles are used to find semantic and hierarchical edges between types and to generate overlapping types. Type profiles are vectors of properties, where each property has an associated probability that indicates its relative importance.

Bouhamoum et al. [6] adapt the previous approach to massive graphs. They transform the graph into a concise representation that contains all the patterns of combined properties. These patterns are collections of properties that appear together on one or more nodes. The scheme results from a clustering algorithm applied to these patterns. The result is the same if the algorithm is applied to the original graph, but is faster if applied to patterns.

4 Approach

The UML diagram in Figure 2 depicts the three main activities in summarization process. We start by mapping the RDF elements of each triple into groups, producing reduced triples. Then, we count repeated reduced triples and produce a summary graph. We finish with a creation of a visualization of the summary graph.

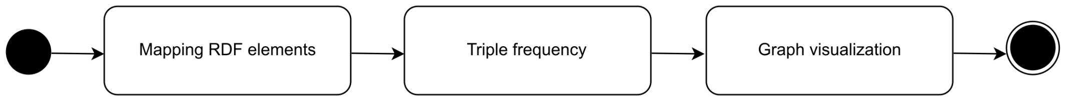


Figure 2 UML activity diagram.

The following subsections detail the graph summarization workflow. Subsection 4.1 describes how RDF triple components are reduced to group identifiers. Subsection 4.2 explains how a summary graph is produced from triples with a reduced number of identifiers. Finally, Subsection 4.3 describes how the graph is visualized.

4.1 Mapping RDF elements into groups

An arc linking two nodes in a semantic graph corresponds to a triple. The subject and object are the source and target nodes and the predicate is the arc. The elements of RDF triples can be IRIs, literals, or blank nodes. We deal with each of these differently:

IRIs: are reduced to their namespace;
Literals: are reduced to their datatype;
Blank nodes: are reduced to a particular group.

Literals occur only as objects in triples and represent unstructured data; for instance, a date or a number. This kind of element is reduced to the literal datatype. Blank nodes occur either as subjects or objects in a triple and correspond to unidentified resources that cannot be referenced from another graph. This kind of element tends to be less frequent and is reduced to a particular group. The most frequent kind of element in a triple is the IRI and reducing them is the core of the proposed approach.

Figure 3 shows a triple from a large graph and how its IRIs are shortened using namespaces. The central column presents the original IRIs with the namespace prefix underlined. The right column presents the short version of the same IRI with the prefix replaced by an alias. The alias is also underlined and separated from the suffix by a colon.

Triple	**Original IRI**	**Short IRI**
Subject	`http://kbpedia.org/kko/rc/Abbey`	`kko:Abbey`
Predicate	`http://www.w3.org/2000/01/rdf-schema#subClassOf`	`rdfs:subCLassOf`
Object	`http://dbpedia.org/ontology/Abbey`	`dbo:Abbey`

Figure 3 Example of a triple and its corresponding triple with namespaces.

IRIs are mapped to their namespaces. This is done by
1. checking explicit prefix declarations in the RDF file;
2. checking the Prefix.cc prefix database;
3. pattern matching.

If the RDF file being processed has any prefix declarations, these are used as the preferred method for finding prefixes: any IRIs matching these prefix declarations are reduced to the corresponding alias defined in the file. IRIs not reduced by the previous step are looked up on a database of prefixes downloaded from the Prefix.cc website[4], which contains a mapping between prefixes and their common alias for almost 3000 prefixes. These prefixes are inserted into a trie where they can be efficiently searched. For all the remaining IRIs we attempt to find namespace prefixes by deleting trailing chunks of the IRIs and searching for common patterns. Namespace prefixes usually end in either `'#'` or `'/'`, so we repeatedly delete the portion of the IRI after the last trailing `'#'` or `'/'`, and add to the prefix list any truncated IRI appearing multiple times.

4.2 Graph summary creation

Triples in the original graph are converted into reduced triples, with each of its elements (subject, predicate, object) reduced to a group identifier, as detailed in the previous subsection. Then, the reduced triples are condensed into a summary graph.

The summary is itself an RDF graph and each statement corresponds to a reduced triple. These triples are reified RDF statements: each is attributed its own identifier, allowing assertions about it to be made. All resources in the summary graph are identified with IRIs on the `ngs` namespace[5]. These include triples identifications and references to groups. The namespace is also used for a predicate that states the number of occurrences of reduced triples, counted during triple conversion.

```
@prefix ngs: <https://www.dcc.fc.up.pt/~up201605706/ngs#> .
ngs:t1 rdf:type rdf:statement
ngs:t1 rdf:subject ngs:kko
ngs:t1 rdf:predicate ngs:rdfs
ngs:t1 rdf:object ngs:dbo
ngs:t1 ngs:num_occurrences 530
```

Figure 4 Example of reduced triples represented in the summary graph.

Listing 4 shows the set of triples in the summary that represent a set of reduced triples. It corresponds to 530 triples in the original graph that are mapped to the single triple *[kko rdfs dbo]*. That is, those 530 original triples had subjects that were reduced to `kko`, predicates reduced to `rdfs` and objects reduced to `dbo`.

4.3 Graph visualization

The graph summary is then processed to be visualized. This requires the conversion of the graph to a DOT file, using the following visual features to represent nodes and edges characteristics:

1. **node size:** nodes appearing more frequently in the graph summary are represented in a larger size;
2. **node color:** groups corresponding to literal data types are represented in a different color;

[4] `http://prefix.cc`
[5] Corresponding to the prefix `https://www.dcc.fc.up.pt/~up201605706/ngs#`.

3. **edge thickness:** triples with a higher number of occurrences in the graph summary are represented with thicker edges;
4. **edge color:** edges are drawn with different colors, each corresponding to a different group.

These visual features highlight the relative frequency of namespaces used in nodes and edges and the connections between them. Additionally, it makes it easy to detect other graph features such as strongly connected components, or disconnected graphs.

5 Validation

To validate the proposed semantic graph summarizing technique we applied it to KBpedia knowledge graph.We ran the program in two ways: converting the object literals to their data types (option 1 in the table of results) and not converting them (option 2).

Table 1 shows the number of triples that the produced smaller graph has, the approximated time that the program took to finish to produce a graph and to write the RDF statements, the percentage of reduction that was obtained like this: $\frac{\#triples - \#reduced\_triples}{\#triples} * 100$ and the number of triples that are processed per second: $\frac{\#triples}{time}$.

Table 1 Table of results.

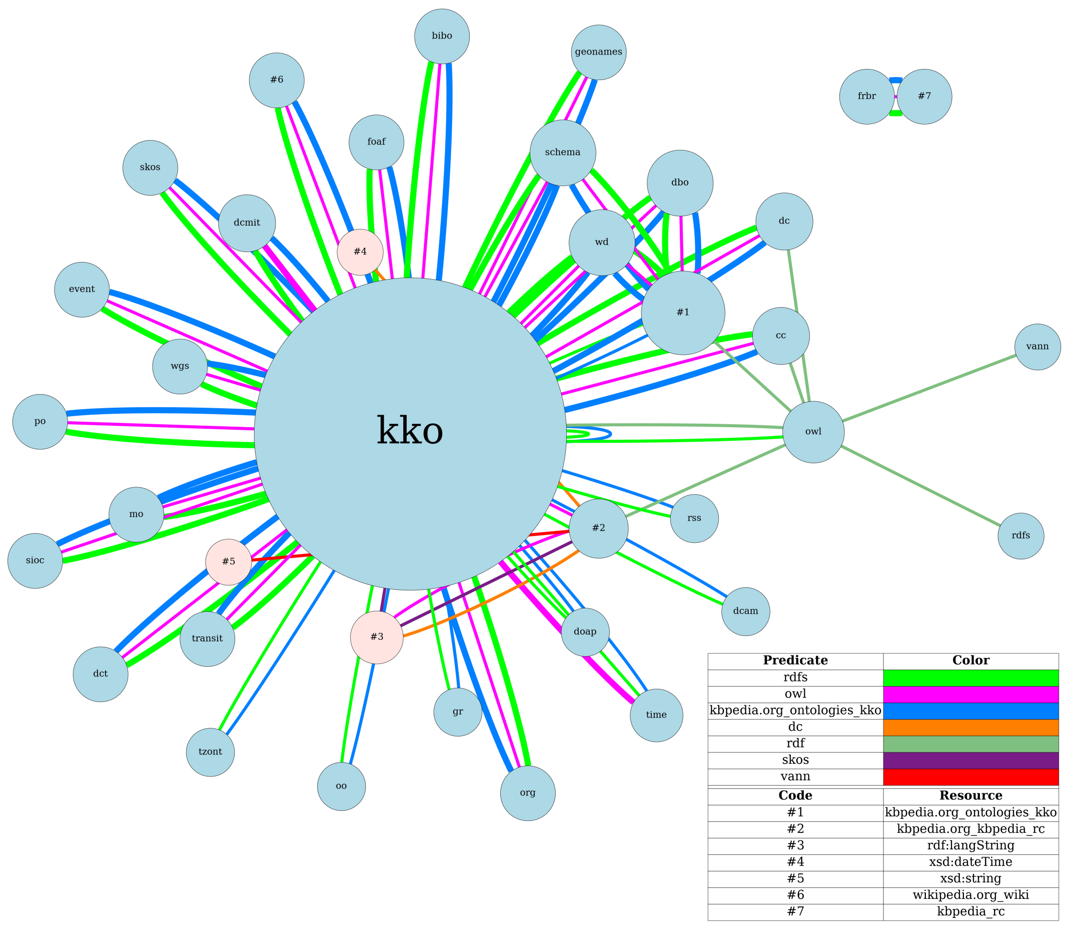
Graph	#Triples	Option	# Reduced triples	% Triple reduction	Time	#Triples/s
KBpedia	1 175 147	1	159	99.99	0h07	2 798
		2	442 290	62.36	01h10	280

Figure 5 shows a visual representation of the KBpedia's summary graph. From this summary we are able to discover several Kbpedia features.

- KBpedia's namespace for individuals, `http://kbpedia.org/kko/rc/`), represented in the figure by the **kko** node, is both the most central node (most of the other nodes are connected to it) and the most frequently mentioned.
- The second largest node (`#1`) corresponds to the `http://kbpedia.org/ontologies/kko#` namespace, used in KBpedia to specify ontology elements.
- KBpedia uses the IRI `http://kbpedia.org/kbpedia/rc#` to identify the graph itself. It is the only resource in its namespace (`#2`), and it connects to **kko** and three literal groups, **rdf:langString**, **xsd:dataTime** and **xsd:string**.
- Most of the predicates in KBpedia come from three different groups. The most frequently used are **rdfs** and KBpedia's **kbpedia.org_ontologies_kk** (corresponding to the `http://kbpedia.org/ontologies/kko#` namespace, followed by **owl**.
- The namespace `http://kbpedia.org/kbpedia/rc/` (ending in '`#`' instead of '`/`', corresponding to group `#7` in the figure) defines a set of classes connected only to the **frbr** group (which corresponds to the `http://purl.org/vocab/frbr/core#` namespace).
- Triples of these groups are disconnected from the core of KBpedia. This is a interesting and unexpected insight provided by the summary.

6 Conclusion and future work

Massive semantic graphs are increasingly hard to understand and visualize. A remedy is to summarize them into smaller graph. In most graph summarizing techniques nodes are grouped into supernodes, and edges between them are grouped in superedges, thus reducing



Predicate	Color
rdfs	🟩
owl	🟪
kbpedia.org_ontologies_kko	🟦
dc	🟧
rdf	🟩
skos	🟪
vann	🟥

Code	Resource
#1	kbpedia.org_ontologies_kko
#2	kbpedia.org_kbpedia_rc
#3	rdf:langString
#4	xsd:dateTime
#5	xsd:string
#6	wikipedia.org_wiki
#7	kbpedia_rc

Figure 5 Visualization of KBpedia graph summary.

graph sizes. Our ongoing research explores the use of namespaces to obtain these supernodes and edges for RDF graphs. The work presented in this paper contributes with a summarizing technique for RDF graphs and a Python package implementing it.

In the proposed approach, RDF triple elements are mapped into groups according to their kind: IRIs are mapped into their namespaces, literals into their data types, and blank nodes into a particular group. The result is a collection of triples of the mapped elements, where most triples are repeated several times. The summary graph is also an RDF graph, where reduced triples are reified, and the nodes representing each triple record the number of repetitions. Finally, the summary graph is converted into the DOT graph description language for visualization.

The proposed approach was applied to large RDF graphs, one with over 1 million triples and another with over 3 million triples. For the smaller one, the resulting summary graphs were produced in a short time and provided meaningful information. For larger graphs, such as LinkedMDB, results could not be obtained in a timely manner. Hence, the current implementation must be optimized to handle larger semantic graphs.

The work presented in this paper is still in progress, and we have planned several improvements. We will explore this approach with even larger graphs to improve the algorithm efficiency. We will formalize graph summaries using RDF Schema, defining the reification of mapped triples. Finally, we will contribute a package for summarizing RDF graphs to PyPI, the repository of software for the Python programming language, which will be available at `https://pypi.org/project/rdf-summarizer/`.

References

1 Dörthe Arndt, Jeen Broekstr, Bob DuCharme, Ora Lassila, Peter F. Patel-Schneider, Eric Prud'hommeaux, Jr. Ted Thibodeau, and Bryan Thompson. RDF-star and SPARQL-star. URL: `https://w3c.github.io/rdf-star/cg-spec/editors_draft.html`.

2 Graphviz authors. Graphviz. URL: `https://graphviz.org`.

3 Mike Bergman and Frédérick Giasson. KBpedia. URL: `https://kbpedia.org`.

4 Tim Berners-Lee. Reifying rdf (properly), and n3. URL: `https://www.w3.org/DesignIssues/Reify.html`.

5 Angela Bonifati, Stefania Dumbrava, and Haridimos Kondylakis. Graph summarization. *arXiv preprint arXiv:2004.14794*, 2020.

6 Redouane Bouhamoum, Kenza Kellou-Menouer, Stephane Lopes, and Zoubida Kedad. Scaling up schema discovery for rdf datasets. In *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*, pages 84–89. IEEE, 2018.

7 Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. Summarizing semantic graphs: a survey. *The VLDB journal*, 28(3):295–327, 2019.

8 Dinis Cruz. Dot language (graph based diagrams). URL: `https://medium.com/@dinis.cruz/dot-language-graph-based-diagrams-c3baf4c0decc`.

9 Richard Cyganiak. prefix.cc. URL: `prefix.cc`.

10 Linked data. Linked movie database. URL: `https://data.world/linked-data/linkedmdb`.

11 Christina Feilmayr and Wolfram Wöß. An analysis of ontologies and their success factors for application to business. *Data & Knowledge Engineering*, 101:1–23, 2016.

12 Elshad Karimov. *Trie Data Structure*, pages 67–75. Apress, Berkeley, CA, 2020. `doi:10.1007/978-1-4842-5769-2_9`.

13 Kenza Kellou-Menouer and Zoubida Kedad. Schema discovery in rdf data sources. In *International Conference on Conceptual Modeling*, pages 481–495. Springer, 2015.

14 Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM computing surveys (CSUR)*, 51(3):1–34, 2018.

15 Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. URL: `https://www.w3.org/TR/rdf-sparql-query/`.

16 David Wood Richard Cyganiak and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. URL: `https://www.w3.org/TR/rdf11-concepts/`.