Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

Exploiting Context-Dependent Quality Metadata for Linked Data Source Selection

by

Beyza Yaman

Theses Series **DIBRIS-TH-2018-?**

DIBRIS, Università di Genova

Via Opera Pia, 13 16145 Genova, Italy http://www.dibris.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria,

Robotica ed Ingegneria dei Sistemi

Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum

Exploiting Context-Dependent Quality Metadata for Linked Data Source Selection

by

Beyza Yaman

April, 2018

Dottorato di Ricerca in Informatica ed Ingegneria dei Sistemi
Indirizzo Informatica
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università degli Studi di Genova

DIBRIS, Univ. di Genova
Via Dodecaneso, 35
I-16146 Genova, Italy
http://www.dibris.unige.it/

Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum
(S.S.D. INF/01)

Submitted by Beyza Yaman
DIBRIS, Univ. di Genova
beyza.yaman@dibris.unige.it

Date of submission: February 2018

Title: Exploiting Context-Dependent Quality Metadata for Linked Data Source Selection

Advisors: Giovanna Guerrini
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università di Genova
`giovanna.guerrini@unige.it`

Barbara Catania
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università di Genova
`barbara.catania@unige.it`

Ext. Reviewers:

Abstract

The traditional Web is evolving into the Web of Data which consists of huge collections of structured data over poorly controlled distributed data sources. Live queries are needed to get current information out of this global data space. In live query processing, source selection deserves attention since it allows us to identify the sources which might likely contain the relevant data. The thesis proposes a source selection technique in the context of live query processing on Linked Open Data, which takes into account the context of the request and the quality of data contained in the sources to enhance the relevance (since the context enables a better interpretation of the request) and the quality of the answers (which will be obtained by processing the request on the selected sources). Specifically, the thesis proposes an extension of the QTree indexing structure that had been proposed as a data summary to support source selection based on source content, to take into account quality and contextual information. With reference to a specific case study, the thesis also contributes an approach, relying on the Luzzu framework, to assess the quality of a source with respect to for a given context (according to different quality dimensions). An experimental evaluation of the proposed techniques is also provided.

Contents

Chapter 1

Introduction

Background and Motivations. In the last decade, a huge amount of Linked Data has been published to construct a global data space [46] based on open standards - the Web of Data. The Semantic Web vision [17] of enhancing usability and usefulness of interconnected resources, builds on the idea of 'marking up', often automatically, data with machine-understandable semantic information about the human-understandable content. Servers expose existing data using the so called Semantic Web standards, such as RDF, and semantically enriched data can be queried through the SPARQL language. The Semantic Web view is further enhanced by *Linked Data* [16, 22], providing a publishing paradigm in which not only documents, but also data, can be a first class citizen of the Web.

Even though significant efforts have been devoted to its development, it is still difficult to benefit from data in the Web of Data in an effective way. Typical systems operating on Linked Data collect (crawl) and pre-process (index) large amounts of data, and evaluate queries against a centralised repository. Given that crawling and indexing are time-consuming operations, the data in the centralised index may be out of date at query execution time. An ideal query answering system for *live querying* Linked Data should return current answers in a reasonable amount of time, even on corpora as large as the Web.

In such a live query system, *source selection*, determining which sources contribute answers to a query, is a crucial step of query processing [43] that requires knowledge of the contents of data sources. Source selection is the process that, given a query, identifies a set of relevant data sources i.e., the sources containing the most relevant data items to answer it. Identifying sources containing possible results for a given search query over the Web is a non-trivial task. Lightweight data summaries for determining relevant sources during query evaluation have been proposed and are exploited for this task [85]. Such data summaries are approximate multidimensional indexing structure that store descriptions of the content of data sources. They form the basis for sophisticated query optimisation and help the query processor decide on which sources to route a query or a subquery. In the Linked Data context, indeed, there is a large number of sources, which, in contrast to classic data integration scenarios, are of small size in the range of a few kilobytes to megabytes. The main problems of processing queries hence become *i)* finding the right sources to contain possible answers that can contribute to the overall query and *ii)* efficiently fetching content from these sources, possibly in a parallel way.

Identifying the sources containing the most relevant results for a query is a non-trivial and very challenging task in the Linked Data environment. While the Linked Data cloud contains a high volume of data, indeed, the connection and integration among data is not adequate to meet the expectations of delivering a global searchable data space. The diversity of publishers means that the data fulfilling an information need may be stored in many different sources and described in many different ways. Though the semantic enrichment aimed at facilitating the integration of information from mixed sources, dissolving ambiguities in terminology, improving information retrieval thereby reducing information overload, and identifying relevant information with respect to a given domain, it shift interoperability and data retrieval issues from the syntactic to the semantic level. Taking into account a higher level notion of *context*, as suggested in [38], can help in devising the most relevant sources for the user's information needs, among the ones containing possible answers to the user's query.

Another relevant aspect is that many data sources are unreliable, in that their data is typically dirty. The need to keep into account the quality of data to cope with its increasing quantity decisively emerges. We claim, thus, that sources should be selected not only on the base of relevance but taking into account also the quality of contained data. Quality can be assessed according to several dimensions such as provenance, availability, consistency etc. This allows queries to return not only relevant but also accurate answers, by preferring trustable sources.

Although it is possible to talk about the quality of a data source by itself, quality of the source is evaluated with regard to the circumstances in which data is created or consumed. Quality of a data source, in terms, e.g., of trust, completeness, and freshness may indeed be different depending on the geographical area, historical period, or type of content. As a result, it is not only impossible to assess quality in an absolute way, but it is difficult as well to assess a single quality dimension independently from the context [18] [62]. For example, IMDB is a highly comprehensive data source and provides a high amount of data for the movie domain while DBpedia, which provides a huge amount of data over many domains, is not comparably effective for the movie domain. Similarly, locally maintained data sources may contain high quality data for some specific geographical locations/area, while they can be less accurate, complete and timely for data referring to other geographical areas.

Problem Statement. This thesis faces the problem of efficient source selection techniques for Linked Data. We aim at selecting the most relevant and highest quality sources for further query processing among distributed data sources. In the context of live query processing over distributed sources, we aim at proposing a source selection approach with the following features:

- sources providing most valuable answers (in terms of relevance and quality) are automatically discovered;
- user can specify quality and cost trade-off in source selection;
- content and quality of the sources are analyzed according to a domain-dependent context hierarchy.

To achieve these objectives two main issues have been investigated.

What are the criteria to select best sources? The live query system must be aware of which datasets or portion of the dataset might potentially contain the requested data. Since more data being published on the web, selecting the appropriate source for the task at hand is a non-trivial task. Especially cross-domain data sources (e.g., DBpedia, Wikidata) contain contents from different contexts with various quality issues. This issue is coped with by performing a pre-processing of the data, targeting two different goals:

> ***How can we improve the evaluation of source relevance with respect to the query?*** Data source context is organized according to higher-level domain-dependent *context taxonomies*, i.e., hierarchical clusters of concepts which allow to get a more abstract semantic grasp of the content available at a given source.
>
> ***How can we improve the evaluation of source quality with respect to the query?*** The quality of the subset of the data sources associated with a given context is assessed according to different dimensions. Even though there is a sufficient amount of work dedicated for the quality assessment of the Linked Data, we focus on how to exploit quality indicators, computed in a context-dependent way, to inform (and improve) the source selection process.

How can we select the best sources efficiently in a distributed live query processing context? We would like to employ an indexing structure for the management of data and data sources that enables an efficient selection of *best* sources in a live query processing context. This will result in a system which selects the sources looking at the most relevant and highest quality of sources, by assessing the quality of the sources and computing relevance to the given query. The key to get such a system is an extension of data summaries to take context and quality indicators into account.

Approach Overview. The resulting approach consists of an offline part and an online part, where the offline part is used to extract, analyse and store data. Data are extracted using the underlying semantic structure which allows to interpret data according to their underlying higher-level semantics (concept taxonomy). Data are then analysed according to their quality and content to get quality indicators and metadata of the source (in a context-dependent way). Data are then indexed in a way consistent with the way of accessing data, such that, data summaries also refer to context and quality metadata. The online part consists in the execution of live queries on the data, once the most relevant sources for the user queries are selected.

A framework for context-dependent quality-based source selection, i.e., to cope with live queries over Linked Data taking into account quality in a context-dependent way, has been proposed in [28]. The main focus of the framework is a lightweight data summary approach to select the relevant sources among a set of heterogeneous known data sources using context dependent quality indicators. Figure 1.1 provides a graphical overview of the source selection process, and steps are briefly described in what follows. First, the user query is submitted to the system (Step 1). User query is a conjunctive Sparql query associated with context which is either extracted from the application environment or query itself and associated with quality thresholds which is defined by the user. Given this query, a look-up is performed on an indexing structure returning to the user a ranked list of high quality relevant sources (Step 2). The extended QTree allows the retrieval of potentially relevant sources with quality indicators, for given quality dimensions in a given context, above a given threshold. Once query results are returned to the user, her feedback (if any, Step 3) on the obtained results is considered, resulting in an update and refinement of the quality metadata associated with the sources according to the context (Step 4). Such context-dependent metadata are maintained making use of Named Graphs. Named Graphs mantain detailed quality indicators and metadata, so that this knowledge is shared and easily accessible. Such detail quality information is the basis on which the numeric indicators in the extended QTree are computed. Finally, context-dependent numerical quality indicators are incrementally updated in the extended QTree according to the new metadata (Step 5).

This framework foresees a two steps approach: pre-processing phase and live query phase. In the first phase, the RDF triples are extracted and associated with contexts according to the context model (Section 2), quality scores of the triples are computed in a context-dependent way (Section 3), the data and the sources are summarized into the index structure (Section 4). In the second phase, this indexing structure is used to select the high quality sources for a given query (Section 5). In this thesis work, we will mainly focus on Steps 1 and 2 described above.

Given a user query, the approach consists of four tasks: *i)* context-dependent quality aware source selection, to devise the most relevant sources according to the query and its context; *ii)* queries are than executed on the selected sources and results are returned to the user; *iii)* feedback from the user on the results obtained by the query evaluation is gathered; *iv)* the data quality indicators are refined and updated according to such feedback; *v)* the auxiliary structures employed for source selection are updated according to the refinement in *iv)*.

Contributions. In this thesis, we propose a specific instantiation of this framework focusing on the initial set up of context structure, context-dependent quality assessment and extended data summaries to support source selection, leaving out issues related to feedback gathering and

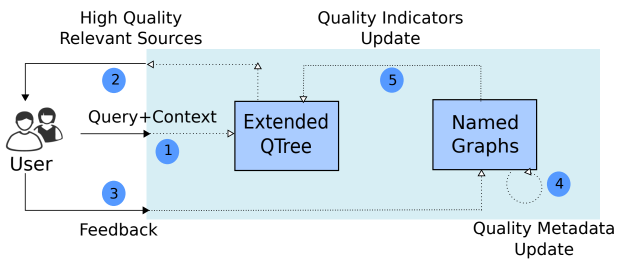


Figure 1.1: Overview of the Approach

incremental refinement of indicators and summaries.

The main technical contribution of the thesis is the extension of the QTree indexing structure by associating contexts and quality indicators to triples and sources. Two different extensions are proposed: the first one, referred to as *extended 3 dimensional QTree*, associates quality information only with QTree buckets and exploits context only in the ranking phase. The second one (referred to as extended 4 dimensional QTree), by contrast, extends the QTree with the context as a fourth dimension (in addition to subject, predicate, and object). This leads to an increase of the dimensionality of the data summary, but increases the pruning potential and thus, potentially, the effectiveness of the indexing structure.

The two data summaries are experimentally evaluated with reference to a real dataset and a corresponding concept taxonomy, in terms of quality and efficiency of the source selection process.

The specific novel contributions of this thesis are: *i)* definition of a context taxonomy, based on SKOS concepts [67], and association of contexts to data and queries; *ii)* context dependent quality assessment of the sources, relying on the Luzzu framework [33] ; *iii)* definition of the extended 3 dimensional QTree and extended 4 dimensional QTree data summaries for context-dependent quality-based source selection; *iv)* source selection based on the context-dependent quality measures; *v)* implementation of the structures and their experimental evaluation.

Outline. The thesis is organized as follows. In Chapter 2, we discuss the role of context for improving the understanding of and retrieval from sources and the specific SKOS based context taxonomy we adopt. Chapter 3 is focused on data quality and discusses: *i)* the possible approaches to assess the quality of sources; *ii)* the possible ways to represent the quality meta data or quality scores in the data management system; *ii)* dimensions/metrics which are relevant for the Linked Data domain; *iv)* how the quality is assessed in a context-dependent way through the Luzzu framework. The original QTree, as well as the two extensions we are proposing, together with their use in source selection, are presented in Chapter 4. Chapter 5 contains the experimental evaluation, including a description of the dataset and experimental setting. Finally, Chapter 6 concludes the thesis and outlines future work directions.

Chapter 2

Context

When one needs to integrate data from multiple data sources, it is challenging to identify the sources that are truly useful for the application domain. This can be done by associating with each data source relevant information describing its context and relying on such information during source selection, with the aim of improving relevance of the query execution system. In this chapter, after introducing context related notions and discussing context modelling in the Semantic Web (Section 2.1), we outline the proposed approach (Section 2.2). A specific model for context modelling relying on the SKOS standard is then presented in Section 2.3. Issues related to how contexts can be associated with RDF triples and sources, as well as SPARQL queries are then addressed in Section 2.4.

2.1 An Introduction to Contexts

This section introduces the basic notions about contexts and discuss the context usage in Semantic Web.

2.1.1 Basic notions

A *context* is any information that can be used to characterize the situation of an entity. An entity is a person, place, fact, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [35]. Characterization of an entity allows machines to discover the underlying latent structure of the content and decide the most relevant portion of the data for user applications. Goal of such applications should be to make interacting with computers easier with the so called *context-aware computing* [5].

A system is *context-aware* if it relies on contexts to provide relevant information and/or services to the user, where relevancy depends on the user's task [35]. This definition allows to customise the context according to the application needs and user environment. In Information Management, context-aware systems are mainly devoted to determining what portion of the entire information is relevant with respect to the ambient conditions [26]. Therefore, context aware management systems require data context modeling and underlying structures for an efficient machine discovery.

A *context model* is a representation of context information, relationships between contexts, and associations between contexts and data entities [89]. A formal representation of context data within a model is required for consistency checking, as well as for performing reasoning [19]. In the general high-level architecture of a context-aware system, context design is carried out according to the application domain, by modeling the elements that affect the knowledge/services/actions that have to be made available to the user at run-time, when a context becomes active [26]. Several models have been proposed to characterize context information (e.g., mark-up based, object oriented, spatial modelling, etc.) Bettini *et al.* review these approaches and discover the most prominent as ontology-based models for knowledge representation [19]. An ontology is a highly expressive mechanism to define concepts and relations which allows a formal language representation. Contextual knowledge is interpreted and evaluated by use of ontology reasoning; this allows computers to determine the contextual compatibility, to compare contextual facts, and to infer new and more complex contexts from core ones, also counteracting the common problem of incompleteness and ambiguity [56].

One important issue related to context modelling is whether contexts should be identified as a universal knowledge or they should be defined with respect to a given application domain. Many works have been proposed which try to identify context information that can significantly enhance the functionalities of domain-specific context-aware applications [19]. There is a general agreement that domain-dependent context modelling provide more specific modelling and reasoning capabilities.

A context model should provide formal elements for representing *context concepts*, semantically related through relationships, leading to the representation of a *context graph*. When the context graph is indeed a tree, we call it *context taxonomy* [92]. Concepts organised according to a hierarchical structure are widely used since they guarantee an easy design and interpretation of the contexts, providing an immediate way for moving from more general concepts to more specialised ones (or vice versa).

Context concepts may refer to various dimensions describing environmental information related to a data item: **i) who*: the user providing the item (e.g., a user profile); *ii) what*: the type of the item *iii) where*: the spatial location and type of the environment in which the item is located; *iv) when*: the temporal location of the item; *v) why*: motivation for they item generation [6].

Understanding context concepts alone is not sufficient for interpreting the situation of a piece of data item but also it requires comprehending semantic similarity among them. For instance when we question if “Mona Lisa" painting and “The Fortune Teller" painting similar to each other, one should ask in which way s/he should consider similarity. Because they can be clustered as in the same location or draw by Italian painters but they don’t depict the same subject. Thus, context similarity allows to distinguish information from different sources by adding semantic relations.

2.1.2 Contexts in the Semantic Web

In Web searching, context is considered as the set of topics potentially related to the search term [7]. In Semantic Web environment context representation has been used in several approaches. Guha *et al.* focus on the logical representation and aggregation of the context information [41]. Stoermer also introduces a Context Relations Ontology (CRO) to make explicit assertions in the related contexts and overcome the problem of inconsistent aggregations. Giunchiglia discusses an epistemologically adequate theory of reasoning with contexts for multi context systems [40]. Bizer *et al.* propose the usage of the Named Graphs for contexts for trust and provenance information [27]. Delbru *et al.* discuss the employment of the context reasoning for semantic web search to perform context dependent RDFS and partial OWL inference based on a persistent TBox composed of a network of web ontologies [34]. Bao *et al.* provides formal and explicit representations for the usually implicit contextual assumptions of data and knowledge on the Web using logical statements [13]. These works propose the representation and identification of the contexts with IRIs/URLs.

2.2 The Proposed Approach

We propose a model driven approach, with domain information captured in various dimensions reflecting different domain elements. The proposed approach is summarized as follows:

- Domain selection: Most applications consist of various distinct sub-elements describing the domain, however, not all of them are relevant from the application point of view. Thus in this thesis, we focused on the tourist attraction domain focusing on specific aspects such as what, where, when context elements.
- Analysing the domain: The requirements of the domain is analysed by use cases and possible examples selecting key features and essential parts to describe the main elements.
- Selecting Reference Contexts: A subset of the well-organized collection of elements are chosen which are going to represent the data portions in the domain. In particlular DBpedia SKOS Thesaurus is used for this specific step as reference contexts.

- Creating Context Taxonomy: Structured representation of topics are extracted from the knowledge base corpus dependent on the domain requirements. Elements representing the domain are structured using hierarchical relations. In addition, the various labels are associated with taxonomy nodes to ensure that the hierarchy is represented for the task at hand.
- Computing Context Similarity : Context Taxonomy is used for inference as the topics are regularized by hierarchical knowledge. The inference on large taxonomies can be costly, thus to ensure efficiency context filtering is applied.

2.3 Context Model

We propose an approach for modelling contexts for semantic web data and sources using a reference knowledge base and organizing contexts as a part of the Context Taxonomy to define and discover the similarity among them.

2.3.1 Reference Contexts

Context modeling requires knowledge of the domain for the scenario which can be retrieved via domain experts. Moreover, it is also possible to exploit from knowledge bases, ontologies and thesauri from sources when they are convenient for the application domain. In the scope of this study, we have chosen SKOS [67] categories to represent our context model since SKOS categories express the context information required for our scenario in a fine way.

SKOS (Simple Knowledge Organization System) [1] is a W3C recommendation for defining, publishing, and sharing concepts over the World Wide Web and it is used for knowledge organization. Concepts are entities, identified by URIs, instances of class (`skos:Concept`). SKOS concepts are then semantically connected by simple and predefined predicates, like (`skos:broader`) and (`skos:narrower`) . Such predicates are one the inverse of the other and can be used to assert a generalization/specialization relationship between two concepts [67].where concepts and their relationships can be stated as RDF triples. For example, For instance if we use the expression (`:Rome skos:broader :Italy`), then, this RDF triple can be inferred as Italy concept is broder than Rome concept. SKOS concepts can be associated with RDF resources through predicate (`dc:subject`). For instance (`:SistineChapel dct:subject :Rome`) (`:Rome skos:broader :Italy`) makes it possible to infer (`:SistineChapel dct:subject :Italy`). That is, if Rome is a subject of Sistine Chapel, and Italy is broader than Rome, then Italy is also a subject of Sistine Chapel. This rule therefore allows a basic type of query expansion [67].

[1]https://www.w3.org/TR/skos-reference/

2.3.2 Context Taxonomy

In this section, we discuss how we obtained the context taxonomy starting with SKOS thesaurus. Since SKOS can be visualised as a directed labelled graph as it is represented in RDF form [2], we define directed graph as follows:

Definition 2.1. *(Directed Graph) A directed graph (digraph) is an ordered pair G = (V, E) is a set of vertices V and a set of ordered pairs of vertices called edges E where* $e = (x, y) \subseteq E$ *and* $x, y \subseteq V$.

Therefore, size of this graph requires the extraction of a subgraph for two reasons: *i)* to provide an efficient computation of the context similarity *ii)* to extract only domain dependent portion of this big graph and leave out the unnecessary portions.

Definition 2.2. *(Directed Subgraph) Given a digraph G=(V,E) and a digraph G'=(V',E'), G' is a directed subgraph of G if and only if* $V' \subseteq V$ *and* $E' \subseteq E$ *where* $a = (x, y) \subseteq E'$ *and* $x, y \subseteq V'$

Following extraction of the subgraph, a minimum spanning tree is created representing our context domain model which reflect the hierarchical structure of the category classes. This structure will be further used to compute the similarity between context which will be discussed in the next subsection. Thus, we model contexts in terms of Context Taxonomy defined as follows:

Definition 2.3. *(Context Taxonomy) A Context Taxonomy, denoted with CT, is a tree where a set of nodes (N) represents contexts, and a set of edges (E) which are unordered pairs of distinct vertices representing relations between contexts defined as* $E = \{e_{subclass}\}$ *where* $e_{subclass} \subseteq C \times C$.

The taxonomies can be created either manually, automatically or semi-automatically but it is a complex process and even it is created automatically it is not free from errors, human expertise is required during taxonomy construction for error-pruning. So taxonomies should be built with less human involvement and less error. In this thesis, we extract a domain-specific taxonomy from a general purpose thesaurus which is represented as graph to automate the process. Context Taxonomy is created following these steps:

- Obtain directed subgraph G' from directed graph G by reversing relevant vertices and edges for the domain
- Find a minimum spanning tree T for G'
- Assign prefix labels to the nodes of T (used to compute context similarity)

[2]`https://www.w3.org/2004/02/skos/core/guide/2004-11-25.html`

The Context Taxonomy allows one to classify and manage the content and can be employed for several operations, such as, describing relations between concepts,inference on the structure and content etc. We describe the nodes of Context Taxonomy as *contexts* which we define as follows:

Definition 2.4. *(Context) A context, denoted with c, is a resource identified with an URI which is represented by a node in a Context Taxonomy.*

2.3.3 Context Similarity

In this subsection, we discuss how Context Taxonomy is used to compute relevant information for the user needs by using context similarity. In order to compute context similarity one can depend on ontological concepts. Context distance is defined as the path between query context and source context which belong to the Context Taxonomy where it is computed by counting the number of edges between the context nodes. Choosing the shortest path means selecting the lowest distance between two contexts. Resnik proposes a well-known metric that simply uses the probability of a concept which relies on the shortest distance between two concepts by using the lowest common subsumer [76].

Context Taxonomy allows to categorize the class of the entity and a comparison is performed taking advantage of the hierarchical structure of the taxonomy based on the notion of information content. A Context Distance function is defined between *context components* to find the similarity of the contexts. This function defines the distance between two elements as the minimum path connecting first component to the second [77]. This relation is defined as follows:

Definition 2.5. *(Context Component Distance) Let assume c_1, c_2 are context components ranging over CT, a Context Distance is a function $d(c_1, c_2) = \min(c_1, c_2)$ where $d(c_1, c_2)$ is the length of the shortest path connecting c_1 with c_2.*

Computing distance between context nodes in a tree has an advantage that it would be possible to assign a weight to each edge on the tree and distance can be computed as the shortest length of paths between contexts which connects them. The length of a walk is the number of edges in it. The shortest walks consist of just a single node and have length zero. Thereupon, for each element of the context this computation is affirmed and the nodes with highest similarity measures are selected from the ontology, thus, final context tuple composed of elements from each dimension is produced according to the Definition 2.4.

In the scope of this work, Context Distance is defined to compute *query-source similarity* to discover more similar sources during query execution. Either source or query might consist of one or more context information with different positions in an ontology. These contexts might have either sibling relation where they are siblings of a parent (Figure 2.1.a), or they may be cousins from other generations (Figure 2.1.b), or they might be ancestor-descendant of each other (Figure 2.1.c). Above all these relations query and source context might be equal where the distance

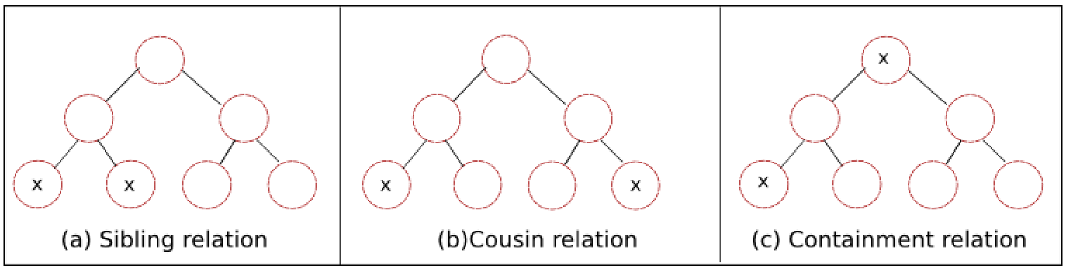


Figure 2.1: Context positions with combination possibilities on a tree

between nodes is 0 (trivial walk). These relations might come across in both ways, such that, while source might have a more general context; in other case query might have a more general context as well.

2.4 Enriching Data with Context Information

This section discusses improvement of the data by enrichment of the characteristic specifications. The objective of defining a source and data in a context is three-fold: *i)* to perceive the characteristics of sources and clustering them w.r.t. these characteristics in structured and generic way *ii)* to understand the hierarchical relation between sources among different possible elements of context *iii)* to use these contexts when searching for a piece of information.

2.4.1 Associating Contexts with Triples

The triples gathered with `skos:concept` information from the knowledge base can be viewed as a content-based clustering of the available data sources. As a result, each source can be viewed as a collection of triples, where each triple is associated with context nodes. To achieve this representation, we propose to associate the content of each source with classes or instances from the context ontology, which also results as the semantic enrichment of the data sources. We first recall the definition of an RDF triple which is the core of Linked Data that connects a subject to an object via some special relations called properties in a statement.

Definition 2.6. *(RDF Triple-RDF Graph) An RDF triple is a tuple* $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ *where* U *denote the set of IRIs representing web resources,* B *denotes a set of blank nodes representing resources without URI,* L *denotes a set of literals representing non URI values such as strings, numbers and dates. A set of RDF triples is named an RDF Graph.*

Based on this definition we also define the triple with context which is extended to quadruple:

Definition 2.7. *(Triple with Context) A triple with context is a quadruple (s, p, o, c) which represents a triple in context c where (s, p, o) is a triple and c is a context.*

Each resource in the dataset -if it exists- has a property of `dct:subject` as we discussed in Subsection 2.3.1. In this approach, we associate contexts to triples w.r.t. the concepts of subject and object elements by discovering their concepts in the dataset. Following the `dct:subject` property concepts we first associate subjects and objects with concepts in the dataset and then they are used to associate for the triple association. At the end each triple is associated with all the concepts related with both elements in the dataset. However, since this is an iterative approach and each subject and object might continously be associated with concepts, we adopt a close-world assumption by limiting this information to the extracted dataset.

This approach allows us to extend triples with the ability to refer to a context. Moreover, the model allows us to reason on classes and instances of the context ontology, thus providing a unique schema for data collections instead of their heterogeneous schemas. Contexts with different granularities increase the possibility of finding most relevant portion of the data for a query. Moreover, data can be represented with different granularities of the same context dimension (e.g. museum, art-museum). Moreover, the context of the content can be represented also in the source level which we will explain in the next subsection.

2.4.2 Associating Contexts with Sources

Source selection is the identification process of a relevant set of data sources for the query. RDF source refers to a persistent yet mutable source or container of RDF graphs. An RDF source is a resource that may be said to have a state that can change over time. A snapshot of the state can be expressed as an RDF graph [3].

RDF resources may have multiple representations that are made available via content negotiation. A representation may be returned in an RDF serialization format that supports the expression of both RDF datasets and RDF graphs where the consumer is expected to use the RDF dataset's default graph [1]. RDF content of the RDF source can be expressed using RDF datasets where a dataset should be understood to have at least the same content as its default graph [2]. We define RDF source and Linked Dataset as follows [84] in the Linked Data environment:

Definition 2.8. *(RDF Source and Linked Dataset) We define the http-download function get : $U \rightarrow 2^G$ as the mapping from URIs to RDF graphs provided by means of HTTP lookups that directly return status code 200 OK and data in a suitable RDF format. We define the set of (RDF) data sources $S \subset U$ as the set of URIs $S := s \in U : get(s) = \phi$. We define a Linked Dataset as $\Gamma \subset get$; i.e., a finite set of pairs $(s, get(s))$ such that $s \in S$. The "global" RDF graph presented by a Linked Dataset is denoted as*

$$merge(\Gamma) := \biguplus_{(u,G)\in\Gamma} G$$

where the operator "$\uplus$" denotes the RDF merge of RDF graphs: a set union where blank nodes are rewritten to ensure that no two input graphs contain the same blank node label.

RDF Dataset of the source is retrieved via content negotiation by dereferencing the RDF resource. We define RDF dereferencing as follows:

Definition 2.9. *(RDF Dereferencing) A URI may issue a HTTP redirect to another URI with a 30x response code, with the target URI listed in the Location: field of the HTTP header. We model this redirection function as redir* $: U \to U$*, which first strips the fragment identifier of a URI (if present) and would then map a URI to its redirect target or to itself in the case of failure (e.g., where no redirect exists). We denote the fix-point of redir as redirs, denoting traversal of a number of redirects (a limit may be imposed to avoid cycles). We denote dereferencing by the composition* $deref := get \circ redirs$*, which maps a URI to an RDF graph retrieved with status code 200OK after following redirects, or which maps a URI to the empty set in the case of failure. We denote the set of dereferencable URIs as* $D := d \in U : deref(d)6 = \phi$*; note that* $S \subset D$ *and we place no expectations on what deref(d) returns, other than returning some valid RDF. As a shortcut, we denote by* $derefs : 2^U \to U \times 2^G; U \mapsto (redirs(u), deref(u)) | u \in U \cap D)$ *the mapping from a set of URIs to the Linked Dataset it represents by dereferencing all URIs (only including those in D which return some RDF).*

Multiple sources might consist of the same information where one source might provide a better or more specific content than the other. Context can be employed as disambiguation or clarification for the content type for further knowledge deduction. Such that, we define source in a context as follows:

Definition 2.10. *(Source with Context) A source with context is a pair SC=(S,c) where S is an RDF source where it is represented with URI and* $c = (e_1, e_2, .., e_n)$ *is a set of Contexts.*

This representation allows us to represent the contexts in the higher level and find the relevant sources for a given query with a context. Different approaches can be considered to associate sources with contexts. We applied the first one in the scope of this thesis from the following approaches:

- associate with a data source ds the set of contexts related to a resource r such that $deref(r) = ds$
- associate with a data source ds the set of contexts related to all the resources appearing as subjects or objects in ds, maintaining for each of them their related frequency in ds

- associate with a data source *ds* the top-*k* contexts (selected among those associated with resources in *ds*), taking into account their frequency.

2.4.3 Associating Contexts with Queries

A query which is specified with a context reveals more information about the user intentions. Context information which is acquired from context-aware applications is used as a filter to improve the service by allowing query extension with a domain of discourse, e.g. location in mobile applications. Contexts which are arranged in an semantic hierarchy can be inherited to its child contexts such that query results provide either the context of the query or most similar ones. Queries can be augmented and executed by using Context Taxonomy elements which is represented in the same query language. Therefore, it improves the recall and precision of the results by increasing relevance.

Context aware approaches requires the query expressed in a context to manage the data in a contextual way. This process augments the semantic of the query by characterizing the user intention. The query context proposed in this paper is obtained by the *fusion* of two different context components: *i)* an *external* context component, which comes from the environment from which the query is being asked as well as a *ii)* an *internal* context component, related to what is being asked in the query.

The external component is in turn composed by different information, mainly related with the user profile and the location in time and space from where the query is being asked. For what concerns the user profile, we assume that user explicitly provides her interests in the application environment. For instance she might like post-modern paintings or Jazz music etc. On the other hand, current location and current time of the user are sensed through the mobile app directly. Of course, profile information are more static while location information are highly dynamic. Availability of these dimensions depends on the user and environment. The user might or might not provide her interest on the application, besides her GPS information might be closed, thus, these context dimensions cannot be always available. Moreover, a mobile app can always get the time information.

The internal context component, by contrast, comes from the query itself. It is related to what the query is being asked and to the fact that the user also can specify some interests in the query characterized by space and time. In this case, we consider them as contexts which are explicitly declared in the query [55]. Availability of these dimensions depend on the query executed by the user. The user interest asked through query might or might not exist in the context ontology. Moreover, she might ask for an explicit entity with a time or space, thus, these context dimensions cannot be always available.

Note that because of the external component of the query, the same query asked in different context might lead to the selection of different sources, and thus, ultimately, to returning different results. Therefore, these two types of contexts are compared to have a unique context element for each dimension.

We assume that there is a reasoner component which is responsible for matching the suitable query contexts based on internal and external context. Using such reasoner, each query can be mapped to one or more components in the Context Tree describing its contexts. Thus, we aim at executing queries corresponding to various contexts which might mean executing queries multiple times with various contexts. Since the execution costs will be very high, it is required to discover the contexts which might be likely related to the personal contexts. So given a set of query context components how can we arrive to the query contexts which is represented in the context graph tree:

- Given a context tree represented with different context dimensions, an instance might be related either one or more Context Dimension
- Context Dimensions rely on the Context Taxonomy model where data from a data domain described by a pre-specified set of attributes
- Once context components are found for each entity of the query, we represent them as a node and then search for it in the tree hierarchy

This study focuses conjunctive SPARQL queries which consist of basic graph patterns (BGPs) which combine triples patterns. A triple pattern is a tuple with 3 components containing one or more variable in any position of the triple. We define these notions as follows:

Definition 2.11. *(Triple Pattern, Basic Graph Pattern) A triple pattern is a triple with variables in the subject, predicate and/or object position. A query variable $v \in V$ where V is infinite and disjoint from RDF-Triple. A triple pattern is represented as $(RDF-Triple \cup V)x(I \cup V)x(RDF-Triple \cup V)$. Variable in the triple pattern is prefixed by question mark (?). A Basic Graph Pattern is a set of triple patterns.*

Moreover, we associate a Sparql query with context to be able to match with triples represented in context dimensions. We define query with context as follows:

Definition 2.12. *(Query with Context) A query with context is a pair QC=(Q,c) where Q is a Sparql query consisting of a set of basic graph patterns and $c = (e_1, e_2, .., e_n)$ is a context.*

As a result of execution of the *Query with Context* we obtain a list of Query Relevant Sources which is defined as follows [84]

Definition 2.13. ***Query Relevant Sources**: Let $uris(\mu) := \{u \in U | \exists s.t.(v,u) \in \mu\}$ denote the set of URIs in a solution mapping μ. Given a query Q and an intermediate dataset Γ, we define the* *function qrel, which extracts from* Γ *a set of URIs that can (potentially) be dereferenced to find further sources deemed relevant for Q:*

$$qrel(Q,\Gamma) := \bigcup_{tp \in Q} \bigcup_{\mu \in [\![\{tp\}]\!]_\Gamma} uris(\mu)_\Gamma$$

2.5 Use Case

In this work we adopt a data centric context model which allows us to integrate the data with related topic information. Therefore, a portion of the DBpedia English SKOS thesaurus is obtained in a controlled mechanism by using explicit mappings to create local domain dependent context model. We will discuss further how we adopted SKOS ontology in Section 2.5.

In this subsection, we will further exemplify the steps we followed respectively in Section 2.2 by applying on the reference scenario. In the scope of this work, we used SKOS to construct our domain ontology where SKOS allows the mapping of semantic relations between terms, using semantic reasoning methods to suggest links from one terminology to another. Once a satisfactory context model is obtained, we need: *i)* to identify the sources containing data relevant for the devised concepts to gather associated data, and *ii)* to extract data from the sources w.r.t. the determined concepts. This subsection presents the design of a system to adapt SKOS model for the scenario, identifying and extracting the relevant part of DBpedia for a given domain.

DBpedia: The DBpedia project [58] builds a large-scale, multilingual knowledge base by extracting structured data from Wikipedia editions in 125 langugages. All these versions together describe 38.3 million things, out of which 23.8 million are localized descriptions of things that also exist in the English version of DBpedia [3]. DBpedia is one of the main cores of Linked Open Data (LOD) and widely used in the semantic web research community for a numerous applications such as NLP, entity disambiguation, question answering, knowledge exploration, and many more. It is developed by the collaboration of academic research and industry. these collaborations makes it more evident that outgoing and incoming links both in the instance and schema level increases every day which allows to infer information in several knowledge layers. Beside many reports confirm the increase in the usage of DBpedia progressively [4]. Data sources like DBpedia or Wikidata use SKOS categories to model the topics of the included data domains. Beside that each day new datasets takes the advantage of using existing SKOS concepts from DBpedia to enrich their data sources with additional topic information [5].

[3]http://wiki.dbpedia.org/about

[4]https://goo.gl/DD78sE

[5]https://pro.europeana.eu/resources/apis/entity

SKOS concepts in DBpedia are extracted from the Wikipedia categories which systemize the Wikipedia pages with the goal of classifying them according to their topics. Each Wikipedia page has one or more categories representing them in different perspectives, e.g., a painting might have a location category beside painter category. These categories are associated to the pages with most specific classes, such that, further inference rules can be applied to extract superclasses of category classes hierarchically. DBpedia adopts SKOS concept categories with the same goal of Wikipedia for classifying the instances by using SKOS standard properties, such as, `(skos:broader)`, `(skos:Concept)` and `(dct:subject)`. DBpedia is a cross-domain knowledge base, thus, resources in DBpedia may be associated to the concepts from different domains which might be either instances or classes in the domain, e.g., Rome instance or City class. DBpedia SKOS thesaurus is an extremely large directed graph which contains almost 1,5 million nodes and 3 million hierarchical relations. We define directed graph as follows:

Adopting SKOS Model: In this thesis, we take advantage of the underlying semantic structure of the Linked Data to extract domain specific data collections exploiting SKOS categories. Context modeling entails: *i)* identifying a number of relevant context dimensions, *ii)* identifying the classes and instances relevant for the application domain. In the scope of the reference scenario, we first determine the context dimensions required for the use case and we find the corresponding `skos:Concept` classes in the knowledge base that we take as reference. We adopt English DBpedia as a reference knowledge base for the SKOS categories to use as a mediated schema since it provides the defined classes and external links to several data sets in the instance and schema level. In DBpedia language chapters, SKOS thesaurus are created as a seperate ontology in its own language (DBpedia Italian, French etc) since each language edition is extracted seperately from the related Wikipedia language editions. We have chosen English DBpedia for three reasons: Usage of English DBpedia *i)* allows us to overcome the language problems *ii)* mapping more then one datasets to the one specific schema *iii)* overcome the lack of SKOS thesaurus for some other datasets except DBpedia.

We defined context model as members of a root domain concept of *Tourist Attractions in Europe* . We recursively explored sub-concepts and instances as possible domain concepts for the scenario starting from the seed node. To avoid having irrelevant concepts, we restricted the recursion with a depth limit of 3. We represent the context dimensions and selected concepts for each dimension as a combination of the following elements: (*what*): Point of interest represents the specific places that user might find useful or interesting (the natural parks, historical buildings, hotels, restaurants etc.). (*where*): Location represents a physical location, which can be seen according to different detail levels, such as country, region, city etc. (*when*): Time represents the position in time, again according to different detail levels.

Our aim is to represent the aggregation of different context information into the Context Dimensions within a structured Context Taxonomy, which enables context matching, context reasoning and search. Domain concepts are then identified according to case study scenario and considered context dimensions based on the reference source. Based on the context model we adopted from the English DBpedia we further identify the sources containing relevant data and context dimensions:

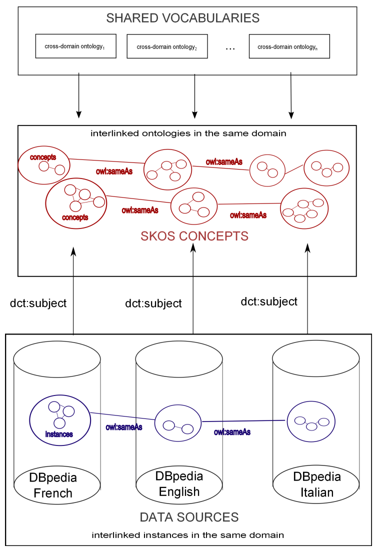


Figure 2.2: Underlying Structure

Identifying sources: We decide the relevancy of the data sets to the scenario depending on the inclusion of the domain data associated with SKOS concepts in the data structure. While some data sources provide data from a specific domain (e.g., IMDB) others provide cross domain data (e.g., DBpedia). This step consists of the manual analysis of the available sources in the LOD cloud to for the possibility of usage for the reference scenario. We utilize English DBpedia as a reference for the SKOS categories and we discovered that following the (`owl:sameAs`) links to other sources we can take advantage of the ontology alignment existing in the semantic data structure. Thus, we follow the (`owl:sameAs`) links to find the mappings between other sources for the same categories used in the scenario. Traversing these links we retrieved the data from French DBpedia and Italian DBpedia for our preliminary evaluation. Moreover, we also obtained Museums in Italy data set from LOD cloud which coincides with our reference scenario. We selected these sources because they provide a SKOS hierarchy linked with each other such that it is possible to find the same concepts in the different sources which allows us to overcome the problem of getting data with several languages. In this way, also we are able to organize the same instances from different sources according to the context dimensions. For each data source we retrieve the data based on the context dimensions.

Data Extraction: Context dimensions and classes are extracted from the data sources via `skos:Concept` classes based on the reference knowledge base (English DBpedia). Relying on the reference source concept (Tourist Attraction in Europe class) in English DBpedia, we recursively retrieved the sub-concepts of the category. The topics of an instance is connected by (`dct:subject`) property to the concept classes which are part of the SKOS hierarchical class organization (`skos:broader`) associated with instances in data sources. Using these relations the sources in English DBpedia are gathered for further context association. For each source, we traversed the *(owl:sameAs)* links to the other data sets and extracted the same instance representation in other data sets to associate it with the same concept in English DBpedia. Figure 2.2 shows the underlying structure that we reveal to extract the data collections. Data sets have the instances which are linked with `owl:sameAs` categories but also they are linked in the higher schema level by SKOS categories. Moreover, the instances have the topics which are described by SKOS categories which are linked via (`dct:subject`) metadata. The triples that refer to our domain concepts are extracted via SPARQL queries which were posed to endpoints. By using Sparql queries the instances of these concepts are gathered to be labeled by the context dimensions existing in the data structure.

Datasets: A real-world set of RDF data sources has been collected from the Web via LDSlicer [6]. The set has been gathered through a breadth-first crawl starting from a specific concept node (Tourist Attractions in Europe) in DBpedia by using content negotiation technique to retrieve the content of RDF sources. As a result, a dataset consisting of 1.086.874 RDF statements has been retrieved from 13.466 RDF sources. We then associated a set of contexts (i.e., SKOS concepts) with each triple.

The whole dataset contains RDF data sources collected from twelve endpoints, including many DBpedia language chapters. In particular, the collected data sources come from English [7], French [8], Italian [9], Dutch [10], German [11], Basque [12], Portuguese [13] DBpedia endpoints and each chapter is extracted from language-specific Wikipedia versions. Besides, we used Yago [14] which is a huge semantic knowledge base, derived from Wikipedia WordNet and GeoNames [60]. Another huge knowledge base is Wikidata [15] which aims to provide a free knowledge base that anyone can edit and contribute directly to the structured knowledge base [37]. DBpedia-Wikidata is an extension of the DBpedia knowledge base to enrich the DBpedia resources via extracting knowledge from Wikidata entities. Geonames [16] is geographical database covers all countries and millions of place

[6]`https://github.com/keme686/LDSlicer`
[7]http://dbpedia.org/
[8]http://fr.dbpedia.org/
[9]http://it.dbpedia.org/
[10]http://nl.dbpedia.org/
[11]http://de.dbpedia.org/
[12]http://eu.dbpedia.org/
[13]http://pt.dbpedia.org/
[14]http://www.yago-knowledge.org/
[15]http://wikidata.org/
[16]http://www.geonames.org/

names in different data formats and finally, German National Library (DNB) [17] is a knowledge base of national bibliographic data, including all authority data.

An overview of the characteristics of the collected data sources is shown in Table 2.1. In particular, Table 2.1 shows the overall number of triples and the overall number of sources used in the experiments. The table also shows the number of four dimensional tuples, obtained by combining each triple with an associated context, and number of sources obtained by combining each source with an associated context. As we discussed in Chapter 4, we index sources as (source,context) pair in the 4D Qtree, number of sources are multiplied by the number of contexts they are associated. Thus, we have more number of sources indexed in the 4D Qtree structure.

As it can be seen from the table, different endpoints contributed with a different number of sources and triples. There are several reasons for this: 1) English DBpedia do provides higher number of resources than other endpoints; 2) other sources may not be connected with owl:sameAs links to English data sources; 3) some sources are focused on more specialized domains than English DBpedia.

Data Endpoints	#Triples	# Sources	#Triples with context	#Sources with context
English DBpedia	137171	1802	616169	7571
French DBpedia	106886	668	682365	3541
Italian DBpedia	49515	746	302778	4232
Dutch DBpedia	45665	429	281421	2354
German DBpedia	99540	851	564403	4317
Basque DBpedia	1372	142	11345	801
Portuguese DBpedia	39116	412	239830	2137
Yago	304672	2184	1795355	10489
Wikidata	188741	2262	1025594	10829
DBpedia-Wikidata	70145	2247	377856	10737
Geonames	42332	1594	212702	7809
DNB	1719	129	10748	797
Total	1.086.874	13.466	6.120.566	65.619

Table 2.1: Description of the considered dataset

Creating Context Taxonomy: Following the extraction of the sources from endpoints, next step is creating a Context Taxonomy from the existing contexts in the extracted data. Although we extract the arbitrary data sources w.r.t. the SKOS categories with a depth of 3 to represent the topics of the sources; each source is associated with multiple context information other than the

[17] `http://www.dnb.de/EN/Service/DigitaleDienste/LinkedData/linkeddata_node.html`

Tourist Attraction class. We follow the given steps below to create a Context Taxonomy whic is aminimum spanning tree:

- A set of contexts associated with sources are extracted from the dataset. Let N is a set contexts extracted from the sources in the data set where $N = \{c_1, c_2, ..., c_n\}$ and let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ is a Directed Graph which represents DBpedia English SKOS Thesaurus; $N \subseteq \mathcal{N}$

  We note that those sources are part of the Tourist Attraction domain as the initial Context Dimension, but we have also seen that all the concepts contained by the source in the DBpedia should be represented as contexts since they are part of the topic description of the sources. Thus, each source domain concept should be directly related to the concepts identified in its metadata definitions and, we include also those novel context dimensions in their respective domain. In total, we obtained 1632 context nodes associated with sources in the existing data set.

- We create a subgraph with extracted context nodes by enriching the relations between these nodes by computing shortest paths for each node pairs. Let $G = (N_{enriched}, A)$ is a graph where $N \subseteq N_{enriched} \subseteq \mathcal{N}$ and $A \subseteq \mathcal{A}$ and G is a Directed Subgraph of $\mathcal{G}$. For instance, given context nodes $(c_1, c_2) \in N$ are connected in $\mathcal{N}$ with a path $c_1 \rightarrow c_x \rightarrow c_y \rightarrow c_2$ where $c_1, c_x, c_y, c_2 \in \mathcal{N}$

  Using these context nodes, we computed the shortest paths between each node pairs in the DBpedia English SKOS Thesaurus and discovered the relations between these nodes to create a subgraph. We did not exclude intermediate nodes in the path since they are part of the domain discourse, thus, we took all the nodes in the walk between (c_1, c_2). At the end, we produced a subgraph of the English DBpedia SKOS graph which represent domain with 5917 context nodes. Comparing with a volume of 1.5 million SKOS concept nodes, the difference can be noted explicitly.

- A Minimum Spanning Forest is created starting with this Directed Subgraph, such that,let $MSF_{enriched}$ is a minimum spanning forest $MSF_{enriched} = \{\mathcal{T}_1 = (\mathcal{N}_1, \mathcal{A}_1), .., \mathcal{T}_k = (\mathcal{N}_k, \mathcal{A}_k)\}$ are minimum spanning trees and where $\mathcal{N}_1 \cup, .., \cup \mathcal{N}_k = N_{enriched}$ and $k \in \mathbb{N}$.

  Next step is the creation of minimum spanning trees from this graph to build our Context Taxonomy. Since we have other domains beside Tourist Attraction domain we create more than one minimum spanning tree representing different Context Dimensions. In the scenario, we are looking for three context dimensions which can be extracted for both internal and external information. We assume that location dimension is retrieved by the mobile application. Figure 2.3 shows a portion of the tree with different Context Dimensions. Black nodes in this tree are the nodes which are found as a result of the subgraph extraction and we keep them for further inference operations (for the sake of simplicity some black nodes are omitted). For example, initially we have public park entities called "Park im

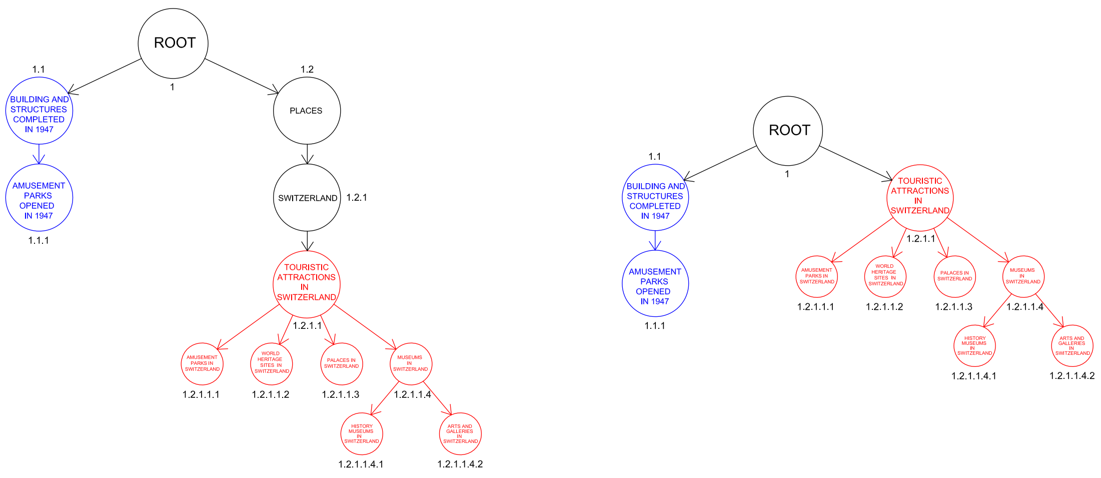
Grüene and Jungfrau Park" which are members of the “Amusement Park in Switzerland" context. Even though these entities have a common context, they might have some other concepts shared with other entities, such that, “Park im Grüene" has another context in another Context Dimension called “Buildings and structures completed in 1947". These two Context Dimensions cluster the same source from two different perspectives. In the first concept, *what* and *where* information are combined to represent the entity, on the other hand, *what* and *when* information are represented in the second concept. Using these combined contexts we achieved to create a structure presented in [75] where different context clusters are aggregated for a meaningful content labeling.

- Labeling of tree is performed according to Breadth First Search and Depth First Search Algorithm.

  Having created the Context Taxonomy, we label the context nodes for further computations of the Context Distance. Each vertex is labeled twice for *i)* traversing the tree using Breadth First Search algorithm and discovering the neighbor nodes first and giving them similar labels, e.g., starting from the root as 1 children of the root have labels “1.1" and “1.2" (Figure 2.3). This representation allows us to discover similarity by computing least common subsumer approach *ii)* traversing the tree using Depth First Search algorithm and labeling them with the number of search order to use them during indexing which will be further discussed in Chapter 4. Using this new subgraph as a Context Taxonomy, we enrich our triples and sources extracted from the endpoints.

- Intermediate nodes are eliminated from the minimum spanning forest to create a new minimum spanning forest with the initial nodes. Let MSF is a minimum spanning forest $MSF = \{T_1,..,T_k\}$ where $T_1 = (N_1,A_1)$ and $T_k = (N_k,A_k)$ are minimum spanning trees and $N_i \cap N_j = \phi$ and $N_1 \cup, ..., \cup N_k = N$. In this case new relation are created, e.g., $c_1 \rightarrow c_x \rightarrow c_y \rightarrow c_2$ is a path where $c_1, c_2 \in N$, $c_x \ and c_y$ are eliminated and a new relation created between $c_1 \ and c_2$ keeping the description of the previous positions.

  Finally, we filter constructed minimum spanning forest to avoid unnecessary nodes included in the tree. This kind of tree is particularly useful for representing hierarchies about the relations among all nodes in a graph. However, as the tree grows it is possible to have some performance problems, such that, tree nodes are filtered to include only initial context nodes of the dataset in the tree. This process is performed by seperating the connections between context nodes and the nodes which were intermediate results of the graph computation. In this new tree, breadth first search labeling is kept as it was computed in the primary tree to preserve the semantic hierarchical relations of the tree. On the other hand, depth first search numbering is re-performed again since there are less number of context nodes in the tree. This new tree can be seen in the Figure 2.3.


ROOT
1
1.1
BUILDING AND STRUCTURES COMPLETED IN 1947
AMUSEMENT PARKS OPENED IN 1947
1.1.1
1.2
PLACES
SWITZERLAND
1.2.1
TOURISTIC ATTRACTIONS IN SWITZERLAND
1.2.1.1
AMUSEMENT PARKS IN SWITZERLAND
1.2.1.1.1
WORLD HERITAGE SITES IN SWITZERLAND
1.2.1.1.2
PALACES IN SWITZERLAND
1.2.1.1.3
MUSEUMS IN SWITZERLAND
1.2.1.1.4
HISTORY MUSEUMS IN SWITZERLAND
1.2.1.1.4.1
ARTS AND GALLERIES IN SWITZERLAND
1.2.1.1.4.2

(a) Context Taxonomy

(b) Filtered Context Taxonomy

Figure 2.3: Context Taxonomy before and after filtering

Chapter 3

Quality

Joseph Juran defines *quality* as “fitness for use"[51], Philip Crosby defines *quality* as “conformance to requirements" [29] and Armand V. Feigenbaum defines *quality* from the business point of view saying “Quality is what the user, the customer, says it is" [90]. The common view of all of these quality gurus is that quality is dependent on meeting the needs of the clients and information seekers. Taking into account literature, we describe data quality by analyzing main elements of quality assessment. Moreover, we discuss quality assessment techniques for Linked Data with a focus on problems due to poor quality data and solutions proposed to address such problems.

Poor data quality isn’t always apparent but it is originated as a result of different motives. Data source diversity causes heterogeneous schemas, plenty of data types and diverse data structures which make difficult for the user to get a homogeneous unified data view. Data dynamicity increases the problem of versioning, freshness and correctness of results w.r.t. the updated data sources, cost and performance of query execution.

Data quality is extremely relevant in different research areas to measure the value of the used data. Information retrieval is adversely affected by poor quality data even in ordinary procedures. Especially in the web environment, quality problems are encountered in different measurement levels of the quality criteria. In the scope of this study, we are interested in assessing the quality of the data for source selection process. In particular, we are looking for the answers of questions *i)* how can we assess the quality of the Linked Datasets using available tools and techniques, *ii)* which subset of the data is more suitable for source selection. We applied several quality metrics implemented via Luzzu framework and measured the quality of the sources in a context dependent way.

This chapter is structured as follows: Section 3.1 discuss the existing quality problems in Linked Data and Section 3.2 gives an overview of the implemented frameworks for the Linked Data quality assessments. Later, Section 3.3 presents the formalizations on quality assessment. Section 3.4 present the used procedure to assess the context-dependent quality and employed framework named Luzzu. Finally, Section 3.5 presents the assessment exemplar from the Luzzu framework.

3.1 Common Linked Data Quality Problems

In this section, we discuss the common problems occurring in the Linked Data context which is evaluated on a portion of data which is gathered by crawling methods. Studying data quality problems in the Linked Data allows us to notice the possible issues in the usa case specific problems.

Auer *et al.* [10, 11] classify the Linked Data life cycle as follows: *i)* extraction of the data from different structured sources, *ii)* authoring to edit RDF documents by revisioning manually, *iii)* interlinking to connect the data sets by associating them externally either manually or semi-/automatically *iv)* enriching knowledge base to provide powerful reasoning, querying or quality checking, *v)* repairing knowledge base to avoid several modeling errors and inconsistencies for all the previous steps. During this cycle, each step may cause the data quality problems as a result of creation, copying, deleting, updating of the data/metadata, or integration processes. Thus, a user developing applications with Linked Data may apply data quality assessment in three stages: during extraction, pre-processing (cleansing, refining etc.), and/or after the data is linked. As a consequence, these steps require a continuous quality profiling of the data.

Hogan *et al.* [48] analyzes the common errors and the condition of the web of data depending on the 12,5 million triples crawled from the web. The data gathered in the study demonstrates that beside having the corrupt data, even the data is not used depending on to the standards most of the time, and in the following the complications are summarized due to this study:

- **URI/HTTP accessibility and dereferenceability:** This category discusses 3 sub-categories: *i) Dereferencability issues* where URIs returns 4xx client error or 5xx server error. Beside, they discuss the existence of the URIs resulting in a 30x redirect code which is encouraged to identify non-information resources. *ii) Availability of structured data* where URIs return a 200 OK response (excluding redirects) and how much these URIs have the text/html content. *iii) Misreported content-types :* where the documents contain the content type information and ratio of correct content type.
- **Syntax error:** Misuse of the characters or namespaces usually caused by simple errors such as unescaped special characters, misuse of RDF/XML shortcuts, and omission of namespace. causes invalid rdf/xml documents. The summary of the analysis shows that only half of the URIs which are used to describe the resources have valid rdf/xml data.
- **Reasoning:** Reasoning use the semantics of classes and properties to interpret the data, and to infer new knowledge. *i) Use of undefined classes and properties :* are defined either because of the lack of knowledge or syntactic mistakes but it causes the wrong inferences

meaningless data. *ii) Misuse of owl:DatatypeProperty/owl:ObjectProperty* where these types are used vice versa. *iii) Members of deprecated classes/properties* are the ones which are deprecated and are not reccomended to use but they still exist in the dataset,e.g. *wordmap:subCategory iv) Malformed datatype literals* where typed literals were described as literals but sometimes they were lexically invalid and sometimes the they were not compatible with datatype range.

- **Non-authoritative contributions:** This category points out the redefinition by third parties of external classes/properties such that reasoning over data using those external terms is affected, e.g. foaf:Image authoritatively defined as a class but it is a property.

Hogan *et al.* [48] analyzes the common errors and the condition of the web of data depending on the 12,5 million triples crawled from the web. The data gathered in the study demonstrates that beside having the corrupt data, even the data is not used depending on to the standards most of the time. In contrast to the Pedantic Web study, Hogan *et al.* [49] extend the work by experimenting the 1.1 billion quadruples collected by crawling from 4 million RDF documents spanning through 778 PLDs on the web to quantify the conformance of the data w.r.t. the Linked Data publishing guidelines. The authors studies the best practises of the Linked data and then analyze the level of conformance for each sub-categories. Since one work is the continuance of the other, we summarize the errors and percentages w.r.t the more recent work:

- **Naming Issues** discusses the best practices for URI descriptions, such as, avoidance of black nodes, usage of HTTP URIs, dereferencable URIs, host stable URIs and short URIs which are encouraged by the W3C and provides the ratio of each metric.
- **Linking Issues** discusses the usage of external URIs and existence of őwl:sameAs links to increase the discoverability in different sources.
- **Resource Describing Problems** discusses the avoidance of prolix RDF features which increase the amount of redundant information, re-use of existing terms to increase reusability and usage of several vocabularies by cherry picking them.
- **Resource Dereferencing Problems** discusses the existence of human readable metadata in the dataset, dereferencability of the forward links and existence of metadata about the document and existence of licensing information.

Kafer *et al.* [52] monitor the Linked data quality for a long-term period taking into account the change frequency of data sources and resources. The authors used the LOD Cloud and BTC data set to initialize the crawl and they reached 95,737 dereferenceable URIs spanning 652 pay-level domains. The sum of snapshots contains around 464 million quadruples. The authors had monitored the data for six months, and then analysed the content changes and stability of these documents w.r.t. the 29 weekly snapshots.

- **Document-Level Dynamics** discusses the avaiability of the documents by monitoring them 29 weeks and computes the death rate and change rate of the document within this period.
- **RDF-Level Dynamics** monitors the documents in the triple level by tracking the changes where they involve additions, updates and deletions. Beside they provide term level changes such as subjects, predicates and objects and discovers the predicates which are indicative of dynamic statements. Finally they analyze RDF Link Structure of documents over time. It is checked if the links tends to increase or decrease over time, and what is the rate of fresh links which are added

3.2 Quality Assessment Frameworks

The multi-schema structure of the Linked Data Cloud requires users to examine and understand the underlying structure of the data where different terms are employed from various vocabularies. At the same time, distributed and heterogeneous structure of the sources complicates the process of selecting sources efficiently. Therefore, several data quality assessment tools have been proposed to evaluate data quality and to produce statistics or quality scores on datasets.

Evaluation of the data is performed via both subjective and objective assessment techniques depending on the application. Since there is a wide range of quality aspects, relevant ones should be selected according to the use case analysis. This will allow user to find the relevant methods which serve the goals of the application according to the availability of the data indicators. Later, assessment tools are employed to evaluate quality of the data either using actual content, structure or exploring relationships that exist between value collections both within and across data sets. As a result of these evaluations, detailed statistics are presented for the datasets in an effort to ease of understanding and usage of data.

In the remainder of this section, we will present several existing quality assessment frameworks and tools which are developed with the aim of assessing Linked Data quality. We are going to discuss them under two main criteria: *i)* frameworks which provide summaries *ii)* frameworks which provide quantifiable results.

	ExpLOD	TripleCheckMate	ABSTAT	ROOMBA	LODstat
Metadata Profiling				✓	
Statistical Profiling	✓	✓	✓		✓
Topical Profiling					

Table 3.1: Comparison of several profiling techniques

	ProLOD++	RDFUnit	Sieve	Loupe	Luzzu	WIQA	Aether
Metadata Profiling			✓		✓		
Statistical Profiling	✓		✓	✓	✓	✓	✓
Topical Profiling	✓						

Table 3.2: Comparison of several profiling techniques (cont.)

3.2.1 Frameworks for Summaries

Even though Linked Data has been developed as a descriptive language for the sources, lack of these descriptions affect adversely the consumption of these data. Thus, we further divide the framework types according the usage objectives.

Topical Profiling Contents of the sources in Linked data cloud covers a wide area of knowledge domains. While some sources store data for specific domains (e.g. geoLinkedData) others might store data from different domains (e.g. DBpedia). In the distributed data management systems, it is generally not easy to find the data source which is relevant for the query because of the domain heterogeneity. Thus, understanding and analyzing the characteristics of a data set w.r.t. the topics creates a good basis to provide a qualified result to the user w.r.t. the context they are looking for. Clustering step is performed as a pre-processing phase in advance and provides the topical analysis and labeling of the data for further quality analysis of the data.

There are several studies which profile the data according to their domain which handles the topical clustering of data sets to analyze and increase the data quality. Ritze *et al.* [78] profile the Web tables to improve the content of the DBpedia by using similarity metrics based. They match 33 million Web tables to the DBpedia knowledge base and publish statistics about classes, properties, and instances to fill the missing values in DBpedia from Web tables.Meusal *et al.* [65] investigate the extent of the automatic topical classification of the LOD datasets by applying several data mining techniques (k-nearest neighbor, J48 decision trees, naive bayes) to assign the topical categories to the LOD data sets. The authors follow a supervised clustering approach taking into account the schema structure of the data sets and reach 80.62% accuracy by the naive bayes algorithm. Instead Böhm *et al.* [24] applied an unsupervised approach for discovering the latent topics by exploiting the underlying sub-graph patterns inside graph structured data and then compared the mined result to topics extracted from Wikipedia.

However, there are no frameworks which provides a tool for the topic profiling. ProLOD++ framework which is an extension of ProLOD [25] clusters data using data mining methods (k-means algorithm) to further partition the data into subsets to provide meaningful quality assessment for the data and to compute the labels for collection of data. Beside topic profiling, profiling tasks include statistical information with frequency of the RDF elements and data types. Mining tasks includes discovering inconsistencies in the ontology, synonyms, inverse predicates etc. Cleansing tasks include auto completion, ontology alignment etc. A web user interface is provided for the visualization of the dataset statistics. Thanks to data mining understructure, user is also able to add semi-automatically generated triples to the datasets based on tool suggestions.

Statistical Profiling Second group of profiling tools keep statistical information about the dataset. These information exploit the fine grained exploitation of the data by discovering the type of resources, properties and literals, as well as, frequency of these elements. The aim of this approach is to analyze the consistency and comprehension of the data, as well as, discovering the structure of the data. Using such statistics also allows to identify datasets for specific applications, to improve the selection process or query optimization for the federated queries [61].

ExpLOD [53] framework describe datasets by discovering *the owl:sameAs* links and grouping equivalent RDF resources. Also the framework provides summaries and frequencies of the grouped data by representing class membership, predicate membership, as well as, showing the class and property types of an instance for additional descriptive information. The statistics are computed for the selected datasets from LOD cloud. Although, the framework can be downloaded a web interface is provided by the developers.

LODstat [36] provides statistical summaries including class, property, language, datatypes and vocabulary usage for each individual dataset in the LOD cloud, as well as, general statistics are provided for the whole cloud for 32 criteria. The statistics are published with a combination of VoID statistical vocabulary for general information and RDF Data Cube vocabulary for the aggregated datawarehouse-like information, which can be exported to local computer. Other important feature is that statistics can be queried via Sparql endpoint and REST interface provided by the web front-end. The statistics are produced regularly, such that, it would be possible to see the differences on the datasets over time.

Aether framework [61] provides statistics for general class, property, triple and distinct RDF node information existing in the VoID vocabulary, and they further describe 18 more criteria including namespace analysis, subdivisions by triple property, object or subject, as well as, divisions with most referenced IRIs, classes or namespaces. Similar to LODstat approach also Aether is created to provide the drill-down operations based on aggregated statistics, however, they decided to provide only fully functional VoID vocabulary to overcome the problems of combined representation [61]. The selected datasets are assessed and visualized via web user interface and graph charts. The important contribution of this work is the possibility of querying online VoID description of a Sparql endpoint and possibility of comparing two different endpoint statistics.

ABSTAT Framework [72] provides ontology driven RDF data summaries to support query and navigation. These statistics help to find concepts, data types, and properties of resources described/linked in the dataset. The data is abstracted to provide the triple based common patterns and types to the user. On the other hand, the frequency of the co-occurrences of patterns are summarized to help the consumer to discover the datasets/schemas or link the resources into the other datasets. The main contribution is finding the connection between types, which reveal relations of resources which might have different type information independent from each other. The summaries are represented via web user interface for the selected datasets.

Loupe [66] provides a summary of explicit vocabulary information about the dataset such as the classes and properties used in the dataset and a comprehensive analysis of them by inspecting class, property, triples and named graphs. By using these explicit information further implicit information is analyzed to discover the relationship between classes and properties from different vocabularies. Usually, it requires for a user to understand the content and schema to be able to query the dataset. However, this specific contribution distinguishes Loupe from other related works which allows user deduce the most important features from a dataset. The statistics can be discovered via web user interface.

While primary frameworks (ExpLOD, LODstats) provide statistics about the general facts of the datasets, it can be seen that most recent tools provide more complex statistics of the datasets which are inferred using the general facts. Therefore, they all aim at solving different problems and one who is in need of a statistical tool can select one of the existing tools to use or might further develop it for their requirements. However, the common problem of these tools is the lack of automation for the datasets which is presumably caused by the high evaluation times of the datasets. Unfortunately, they are not able to provide statistics for an on-the-fly dataset assessment but they provide statistics for the previously selected ones by themselves.

3.2.2 Frameworks for Quantifiable Results

These group of tools provide profiling of the datasets based on the metadata attached to the datasets by the publishers or specific metadata inside the content of the data. Particular feature of these tools is that they provide quality scores for the assessed datasets which can be used to quantify and rank them. The data sets are accessed from the data portals such as CKAN [1], DataHub [2], European Open Data Portal for Linked Data [3].

Sieve [64] assesses the quality of datasets based on provenance information to discover completeness, conciseness, and consistency dimensions. Since framework is developed as a part of integration tool, thus, it is also utilized for the fusion of the data by discarding the conflicting values over multiple datasets. User might customize the scoring functions and metrics from the configuration file for the quality dimensions. The approach is vocabulary-agnostic, however, it is restricted to the availability of the provenance information. As a result, the computed quality scores are stored as quality metadata as quads.

WIQA [21] framework exploits a Named Graph associating provenance and trust metadata with

[1] http://ckan.org/

[2] https://datahub.io/

[3] https://data.europa.eu/euodp/en/linked-data

data for quality assessment, and it allows information consumers to filter out information of uncertain quality using various quality-based information filtering policies.

Luzzu [33] assesses dataset quality using a library to describe the dataset quality in a generic way depending on the user-provided quality metrics. The produced results are represented via ontology driven metadata which can be queryable through Sparql queries and quality reports on the assessed datasets. The framework provides a set of built-in libraries for quality dimensions and metrics to assess data quality with flexible usage. Additionally, it also allows to define add-in libraries for user provided domain metrics. These metrics allow users to filter datasets by multiple quality dimensions.

TripleCheckMate framework [54] provides crowd-sourced quality assessment and data quality improvement of resources in a dataset. The framework provides assessment for the subjective quality metrics such as relevancy, accuracy etc. The content of the resources can be evaluated in the triple level by the people who are registered to the website in a fine-granular way. This evaluation is measures by the inter-rater agreements which provides multiple evaluations for a resource. These features allow also evaluation of the voter performance and better evaluation for the source. Later, the votes are used to compute aggregated statistics for the user and resource. A web based user interface is provided by the framework for the evaluation. The assessment evaluation is realized only for DBpedia but it is noted that the framework can be used for any dataset.

ROOMBA framework [8] assesses the quality (license, completeness, availability, provenance etc.) based on the metadata descriptions of the LOD data sets from the data portals. In an extended version of the Roomba framework, the authors provide also a quality assessment module to assess the objective quality measures for the datasets [9]. Instead of checking the content of the data for syntactic problems, the authors assess the general dataset information provided by the publishers. They provide a Command Line Interface application for their framework which evaluates the dataset and produces a quality report including quality and error percentages. On the other hand, it would be possible to add additional modules for the metrics which is needed by the user.

3.3 Quality Assessment

Quality assessment is the process of evaluating if a piece of information meets the information consumer's needs in a specific situation [20]. The goal of assessment is to provide a precise evaluation and diagnosis of the state of the information system with regard to DQ issues. Therefore, the principal outputs of assessment methodologies are *i)* measurements of the quality of data bases and data flows, *ii)*costs to the organization due to the present low quality, and *iii)* a comparison with DQ levels considered acceptable from experience, or else a benchmarking with best practices, together with suggestions for improvements [15].

Our focus is to analyze the quality of the huge amount of data automatically to provide data source descriptions to support quality oriented source selection. In the scope of this thesis we aim at *i)* Discovering the existing metadata of the data sources to evaluate, *ii)* Assigning metrics to the data sources to evaluate the data quality to prove that data corresponding to the particular standards or requirements, *iii)* Improving the quality of the search results by assigning context-dependent quality scores to the data sources. Since we defined a data source as an RDF graph (Definition 3.1) where its content is represented as RDF dataset we provide a dataset quality assessment for the RDF Source [30]:

Definition 3.1. *(Dataset Quality Assessment) A Dataset Quality Assessment is a pair (D, M), where D is an Linked Dataset that will be assessed for quality using a set of metrics M (Definition 3.2) where M is a set of defined metrics on which the dataset D is assessed upon.*

The quality assessment process involves measuring the quality dimensions that are relevant to the user and comparing the assessment results with the user's quality requirements [20]. In the following subsections we will discuss the fundamental concepts of the quality assessment process, in particular, quality dimensions in Subsection 3.3.1 and quality metrics in Subsection 3.3.2.

3.3.1 Quality Dimensions

Fitness for use can be computed depending on the several factors which captures a specific aspect of the data. These factors are called quality dimensions and they are subject to the discussions since relational database era [88, 73, 14]. A Data Quality (DQ) Dimension is a characteristics of data that can be measured or assessed against defined standards in order to determine the quality of data.

Quality is perceived as an aggregation of multiple criteria [69], though all the criteria cannot be applied at the same time. It is challenging and costly to satisfy all the conditions (because there is a trade-off between different dimensions e.g. accuracy and freshness) the selection of the data is realized differently from the usage point of view. Thus, each quality process has its own aspect from the evaluation point of view.

Neumann classifies the quality criteria for the web information systems into four categories,namely, *i)* content related criteria, which is related to the data itself, intrinsic dimensions *ii)* technical criteria, measures the environment conditions such as network access, attached by the software and hardware of the source *iii)* intellectual criteria measures the subjective aspects perspectives of the data, iv) instantiation related criteria determines the representational aspects of the data. Bizer *et al.* [23, 21] classify assessment metrics into three categories, according to the type of information exploited to obtain quality indicators: *i) content-based metrics*, which use the content itself, e.g., the content subject; *ii) context-based metrics*, which rely on the conditions in which data was created/accessed/last modified; *iii) rating-based metrics* , which rely on explicit ratings or feedbacks about data itself, data sources, or data providers.

In the Linked Data environment, Zaveri *et al.* [91] discuss the quality assessment methodologies in their systematic survey consisting also several existing frameworks. Metrics and quality dimensions are identified, as well as classified referencing to the Wang *et al.* [88], namely, *i)* Accessibility *ii)* Intrinsic *iii)* Contextual and *iv)* Representational categories where each category captures the same essence for the underlying dimensions that belong to that category [79]. For this work, 69 metrics are provided for 18 dimensions. Also, these metrics are labelled as subjective and objective.

3.3.2 Quality Metrics

A data quality assessment metric, measure or indicator is a procedure for measuring a data quality dimension [21]. Since the dimensions are rather abstract concepts, the assessment metrics rely on quality metrics that allow for the assessment of the quality of a data source w.r.t the criteria [21]. We define a metric as follows [30]:

Definition 3.2. *(Atomic Quality Metric) An Atomic Quality Metric (AQM) is a function that takes an RDF triple and returns a single value of a simple data type.*

$$q : RDF\ Triple \in RDFSource \rightarrow \{\mathbb{R} \cup \mathbb{Z} \cup \mathbb{B} ...\}$$

where simple data types include real numbers, integers, and booleans.

This atomic metric is later aggregated to find a dataset value. This metric can be applied for each triple of the dataset in order to calculate the quality of the overall elements in a dataset which guides us to find an aggregated quality metric:

Definition 3.3. *(Aggregated Quality Metric) An Aggregated Quality Metric is a function $\bar{\cdot}$ over an AQM which takes n arguments of the same simple data type in a dataset and returns value typically has the same type T as each of its arguments applied on each triple of the dataset:*

$$\bar{q} : T^n \rightarrow T, T \in \{\mathbb{R} \cup \mathbb{Z} \cup \mathbb{B} \cup ...\}$$

Aggregate functions include count, median, maximum, sum operations where they are typically formed by counting the frequency of specific values in the dataset, such as the number of booleans of value true, representing the number of triples that match a certain pattern, or by normalising an aggregate value over the count of all triples in the dataset [30]. For example, average mean can be found by dividing found specific pattern in a triple to the overall number of triples in the dataset.

Quality metrics are the elements used for identifying the quality of the data which assess the adequacy of data items and data sources for a specific goal. Relevance of quality metrics depends on the application domain and on the quality dimensions to be assessed and they can be grouped diversely according to the user and publisher point of view. For an end-point user perspective, the quality metrics might be the statistical information about the data and data source, nontheless, a publisher may use different data elements to calculate the data quality, such as, the data collection itself, metadata about the data or provider, crowdsource and expert rankings.

As a result this metric function occasionally, the score of quality dimension can be calculated as an aggregation of several metrics defined for the dimension. These aggregation metrics are defined by the user to produce singular quality score for the quality dimension taking into account different metric types. For instance Availability of the dataset can be assessed using more than one metric, e.g. dereferencability of the triples and availability of the endpoint in the Linked Data applications. Moreover, these metrics can be defined in the form of sum, average, min, max, which is a different notion of the Aggregation Quality Metric which aggregates the same type of metric into the unique value for a metric.

3.4 Assessing Context Dependent Data Quality

In this section, we will discuss the necessity of context aware quality assessment, applied procedure and employed framework for the task at hand.

3.4.1 Context Dependent Data Quality

Context information is represented as a form of metadata that can be formalized as a semantic layer and represented as an ontology [18]. These metadata includes the descriptions about data instances, collections or sources to characterize data and add the additional definitions for the area of usage. Since the data cannot be properly interpreted without the environment of the creation and access, these definitions becomes essential elements for the analysis of data quality problems.

In an environment where there are a lot of unreliable information, data consumers expect the answers to their queries to be relevant, trustable and to have high quality. However, most of the time queries exposed to the web information systems are ambigious which requires domain information and even if they are not ambigious they don't consist every aspect of the question which might be relevant to answer the query, e.g., “restaurants in Rome" might make more sense for the kind of restaurant required by the consumer or the time of search might indicate more information to access the open restaurant information for the needed time period or type of food (lunch, dinner etc).

Bertossi *et al.* states that data quality is context dependent and the data cannot be assessed seperately from the environment it is produced or used [18, 62]. Semantic contradiction arises because the resources with same URIs might exist in different sources and the descriptions of the data depends on the context. e.g. Milano has different type of information in DBpedia and Geonames, however, Geonames is a data source focused on the location,thus, it is expected that

Geonames has a higher rate of accuracy for the location context. Thus, in the scope of this work, data quality notion is adopted to be context-dependent in Linked Data environment.

We adopt this idea in the Linked Data environment and the assessment of a data collection is realized by mapping them into the appropriate contexts. The quality of data is assessed taking into account the context information and during source selection the distance of the query context is measured w.r.t. the source context to avoid semantic contradiction. This process allows to investigate the data quality from different point of views and granularities.

However, our idea of context quality subtly differs from these studies. While these studies consider the context information as a restriction for interpreting the data, we adopt the idea of context as a characterization of the data for semantic enrichment. To account for context-dependence of data quality, we thus propose a flexible approach exploiting quality profiles to associate various metadata and quality information with data, taking into account context-dependence.

Even though, there are a number of generic frameworks assessing quality in Linked data environment which is discussed in Section 3.2 and some others taking into account the domain dependency for query answering [81], to the best of our knowledge there is no such study applying context depending quality measures for the source selection process in Linked Data. Thus, in the following section we discuss our assessment procedure.

3.4.2 Quality Assessment Procedure

In this subsection we discuss the quality assessment steps by preprocessing the data sources and their contents before indexing them. At the end of the procedure steps, the quality scores with related quality metrics are kept in the quality profiles.

- *Data Input and Parsing*: The data sets can be imported in different ways: *i)* Triple/ Quad dump import *ii)* Crawler import *iii)* Sparql import. During the data access two types of information can be extracted:
  - *Set of triples (data)*: Information about each entity is represented in the RDF form.
  - *Set of metadata (metadata about data)*: In case, the publisher creates meta data about data set, they are accessed during data set access.
- *Context Association*: The data and meta data are processed further and set of triples (data) are attached with a context information. As a result of this process we get several outputs:
  - *Set of Triple with Context*: The triples which are associated with context URIs will have the form of quadruples.

  - *Source with Context*: Depending on the triple context information also the source is paired with context information. The context information of the source represents the context information of the triples which are contained inside the source.

- *Applying Assessment Metrics*: Each assessment metric relies on a set of quality metrics and specifies a scoring function to calculate an assessment score from these indicators. The outputs of the context computation process are queued for the next step. Assessment metrics are defined according to two different granularity level i) for each quadruple of the data set ii) for the document metadata of the data set.

  - *Assessment Metrics on Data*: Looking at the triples in a fine grained manner in the data set, several statistics can be computed.
  - *Assessment Metrics on Metadata of the Document*: Some statistics can be in a coarse grain manner looking at the resource or data set meta data descriptions.

- *Computing Scores for Defined Metrics*: Scoring functions are defined based on the assessment metrics and quality indicators. As a result, quality scores are computed on portions of the data which are held by the data source context seperately. For this reason there are different quality scores for context information per one source.

- *Aggregate Function for metadata* - Depending on the metrics used for computing data source quality, an aggregation function can be applied over diverse assessment metrics. These aggregations will be computed based on the quality ontology hierarchy. Existing quality scores are going to be computed based on the aggregation function to provide an aggregated score for the quality dimension of the data source. Granularity of the quality scores can be kept both as metric and dimension level.

- *Storing Quality Metadata*: The source quality information which is calculated depending on the context is stored as RDF dataset by using appropriate ontologies.

3.4.3 Quality Assessment Framework

Among the possible frameworks, we have chosen the Luzzu framework [33, 31] to assess the quality of sources. Luzzu is a framework to assess a wide range of metrics defined and surveyed in the several papers and gives the flexibility of defining new metrics depending on the needs of the domain. Subsequently, it provides the advantages of queryable quality metadata and aggregated quality reports for the data sets.

The choice is motivated by the fact that Luzzu provides an RDF processor to assess the quality of data bulks automatically and, as a result, produces quality scores for each assessment criteria. The produced results are represented via ontology driven metadata which can be queryable through

Sparql queries. The framework provides a set of built-in libraries for quality dimensions and metrics to assess data quality with flexible usage. Additionally, it also allows to define add-in libraries for user provided domain metrics. These metrics allow users to filter datasets by multiple quality dimensions.

Luzzu framework provides a data quality ontology to represent the data set quality in a formal way. According to this ontology, a dimension might have more than one metric and a metric might have more than one observation, which records data quality assessment value following a computation on a given dataset. These different observations provide the quality assessment on the dataset in different time points, if it is necessary to keep a quality trace of the dataset in different periods.

The ontology driven metadata is defined as the Dataset Quality Ontology (daQ) which is a vocabulary for describing the results of quality assessment of a dataset as a graph. The daQ provides a core vocabulary and it can be easily extended with additional metrics for measuring the quality of a dataset creating quality profiles for the datasets [32]. User may define multiple metrics for a dimension, therefore, quality scores can be retrieved either in the metric level or aggregated in the dimension level. The framework allows to obtain these metric based scores by performing Sparql queries on daQ graph specific for the dataset. In the following subsection, we will explain the chosen metrics and dimensions for the source selection process.

Abstraction in daQ graph is provided in three level as [30]:

$$C \sqsubseteq Category$$

$$D \sqsubseteq Dimension \sqcap \exists inCategory.C$$
$$M \sqsubseteq Metric \sqcap \exists inDimension.D$$

where C is a possible quality measure category, $D = \{d_1, d_2, ..., d_y\}$ is the set of all possible quality measure dimensions, $M = \{m_1, m_2, ..., m_z\}$ is the set of all possible quality measure metrics, and $x, y, z \in \mathbb{N}$. On the other hand, quality metadata is defined as follows where the graph g entails one or more instances of a category c, having one or more dimensions d, each of which defines one or more metrics m [30].

$$g : QualityGraph; c : C; d : D; m : M;$$
$$c\ hasDimension\ d; d\ inCategory\ c;$$
$$dhasMetric\ m; m\ inDimension\ d;$$
$$g| = \{c : C\}$$

Data quality dimensions and metrics provided by the Luzzu framework are given in Appendix Section A. Here we summarize the categories and given metrics shortly:

- Accessibility Category: The dimensions belonging to this category involve aspects related to the access and retrieval of data to obtain either the entire or some portion of the data for a particular use case [91], such as, availability, interlinking, performance, security
- Contextual Category: This category groups those dimensions and metrics that are highly dependent on the task at hand [91], such as, freshness, understandability
- Intristic Category: Intrinsic dimensions are those that are independent of the user's context [91], such as, completeness, conciseness, consistency, syntactic accuracy
- Representational Category: This category is related to the design of data, or in other words: how well the data is represented in terms of common best practices and guidelines [30], such as, interoperatability, interpretability, representational conciseness, versatility

3.5 Context Dependent Source Quality Assessment

We propose context-dependent quality assessment of sources to discover suitable subset of data for the given query. In this section, we present our experiment outcome from assessing the quality of context dependent data sets from eight DBpedia endpoint (English, Italian, French, German, Dutch, Portuguese, Basque, Wikidata-DBpedia), Geonames, Wikidata, Yago and German National Library. The assessments are performed w.r.t. *dereferencability* metric which is found as relevant among the quality metrics presented in Appendix A. even though the system can use any metrics for the experiments, this metric is chosen for the testing purpose.

We provide an example of the quality assessment of the sources for the "World Heritage Sites in Switzerland" context. Table 3.3 shows the quality results of Bernina Express entity for the dereferencability metric retrieved from each endpoint. Even though, we have RDF sources from 12 endpoint it can be observed that we have results from eight of these endpoints. This is because Bernina Express entity is found only in those endpoints. Initially, we can observe that each data source provide different quality measure for the same context. While English and German DBpedia endpoints provide best results for the context at hand; Yago endpoint provide worst quality results. This feature especially allows us to compare the measures for the requested context and select the best of all.

Entity	**English DB**	**German DB**	**French DB**	**Italian DB**	**Porteguese DB**	**Wikidata**	**Wiki-DBpedia**	**Yago**
Bernina Express	0.652	0.728	0.416	0.384	0.285	0.118	0.333	0.017

Table 3.3: Quality scores for dereferencability metric for an entity

As we mentioned before an assessment metric analyze each triple's condition depending on the pre-defined assessment function for a given Linked Dataset. Dereferenceability metric is measured by looking at valid redirects (303) or hashed links according to LOD Principles and shows a variety of scores depending on the context and source. This metric is particularly found important for the availability of the data source, such that, if data is not accesible then the content possessed by the data source cannot be consumed by the user.

Listing 1 shows a piece of daQ graph computed for the dereferencability metric. Luzzu creates a quality graph with an id for each dataset. This graph consists of the observations computed on the dataset with a quality assessment value and provenance information about the time period of the computation. The metric id is defined as dereferencability metric and it is referred in another triple as a subclass of availability dimension which is defined as a subclass of accesibility category as well.

Listing 1: daQ graph produced by Luzzu framework for a given dataset

```
{ <https://w3id.org/lodquator/resource/9b60ce35-f965-4055-bf75-7d9bc17cfb19>
  a       <http://purl.org/eis/vocab/daq#QualityGraph> ;
  <http://purl.org/linked-data/cube#structure>
  <http://purl.org/eis/vocab/daq#dsd> .
}

<https://w3id.org/lodquator/resource/9b60ce35-f965-4055-bf75-7d9bc17cfb19> {
  <https://w3id.org/lodquator/resource/bdcb7de4-29ae-4578-b69d-3545ab47dbe9>
  a       <http://purl.org/eis/vocab/dqm#Accessibility> ;
  <http://purl.org/eis/vocab/dqm#hasAvailabilityDimension>
  <https://w3id.org/lodquator/resource/679063a7-9062-47e6-83be-1ad79b437f9d> .

  <https://w3id.org/lodquator/resource/679063a7-9062-47e6-83be-1ad79b437f9d>
  a       <http://purl.org/eis/vocab/dqm#Availability> ;
  <http://purl.org/eis/vocab/dqm#hasDereferenceabilityMetric>
  <https://w3id.org/lodquator/resource/35b330bc-e124-49c8-b35c-ce98b899223c> .

  <https://w3id.org/lodquator/resource/d56f6c39-e020-472a-bdc6-1865fb91a02f>
  a       <http://purl.org/eis/vocab/daq#Observation> ;
  <http://purl.org/eis/vocab/daq#computedOn>
  <path/Bernina_Express.nt> ;
  <http://purl.org/eis/vocab/daq#isEstimate>
  false ;
  <http://purl.org/eis/vocab/daq#metric>
  <https://w3id.org/lodquator/resource/35b330bc-e124-49c8-b35c-ce98b899223c> ;
  <http://purl.org/eis/vocab/daq#value>
  "0.6521739130434783"^^<http://www.w3.org/2001/XMLSchema#double> ;
  <http://purl.org/linked-data/cube#dataSet>
  <https://w3id.org/lodquator/resource/9b60ce35-f965-4055-bf75-7d9bc17cfb19> ;
  <http://purl.org/linked-data/sdmx/2009/dimension#timePeriod>
  "2017-05-17T11:24:14.727Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .

  <https://w3id.org/lodquator/resource/35b330bc-e124-49c8-b35c-ce98b899223c>
  a       <http://purl.org/eis/vocab/dqm#DereferenceabilityMetric> ;
  <http://purl.org/eis/vocab/daq#hasObservation>
  <https://w3id.org/lodquator/resource/d56f6c39-e020-472a-bdc6-1865fb91a02f> .
}
```

daQ graphs allow us to organize the data/data collections providing quality information and scores for a set of triples and flexibility of accessing data/data collections from different contexts and granularities. Quality graphs also play an important role in allowing further semantic information to be attached to either data items, data sources, or data collections such as dimensions that metrics belong to or provenance information of the quality graph etc. Specifically, different quality-related metadata can be associated with the data item/source/collection they refer to by exploiting different labels.

Assessment of the quality on referenced datasets showed us that it is not always possible to find an entity in each endpoint and an entity in different endpoints might have different quality values for a metric. Thus, we take into account these issues during source selection phase, such that, *i)* we may avoid executing queries in such endpoints not to make additional cost entity does not exist in the endpoint, *ii)* we may choose the high quality sources for the query execution process.

Chapter 4

Context and Quality Aware Source Selection

In this chapter we focus on the implementation of the source selection framework. As we discussed, indeed, due to source independence of the data sources, certain pieces of information (i.e., RDF triples) can be found in multiple data sources [80]. The ability to select the most relevant among these distributed sources for the query at hand is crucial for live query processing. Thus, we aim at providing solutions to summarize and index the data sources in such a way that we can select the most relevant and highest quality sources, so to get for the best approximation of the ideal sources for live queries. We develop a context-dependent quality-aware framework which exploits context and quality information to select the most relevant and highest quality sources w.r.t. the given query.

More precisely, in this chapter we start from the analysis of source selection and data summaries, which constitute the basis for this study, and then we introduce two extensions of the state-of-the-art technique to exploit context and quality metadata. We briefly recall the challenges of querying Linked Data and the existing source selection approaches in Linked data (Section 4.1), and then, we present the background structure of the QTree on which we base our new structure (Section 4.2). Later, we present the new extended structures enabling a novel source selection approach. Two different extensions are proposed: the first one, referred to as *extended 3 dimensional QTree* and defined in Section 4.3, associates quality information only with QTree buckets and exploits context only in the ranking phase. The second one (referred to as extended 4 dimensional QTree and defined in Section 4.4), by contrast, extends the QTree with the context as a fourth dimension (in addition to subject, predicate, and object). This leads to an increase of the dimensionality of the data summary, but increases the pruning potential and thus, potentially, the effectiveness of the indexing structure. The two indexing techniques are experimentally evaluated and compared in Chapter 5.

4.1 Source Selection Approaches for Linked Data

In a data collection as huge as the Web, users are generally overwhelmed by the vast amount of somewhat relevant results . To overcome this problem there is the need for supporting ranking at different levels (domains, resources, etc) and for incorporating trust and personalisation. All these requirements are already researched by the information retrieval, Web, and database communities, but to an extent that has to be leveraged. This also holds for the crucial requirement of supporting additional “meta” information with query results [86]. Also, it has been addressed the possibilities and limitations of executing a query “live”. Live querying implies to retrieve the content of the sources from the Web at query run-time and processing them for answers [83]. While these live query approaches guarantee up-to-date results, their execution times are usually in the range of seconds, even for simple queries. Moreover, they consider only documents, while a combination of documents and available SPARQL endpoints is necessary [86].

Specifically, Linked Data poses specific challenges above all the problems. Users must first be aware of which exposed datasets potentially contain the data they want and what data model describes these datasets, which can not be expected from a user to have a previous understanding of available linked datasets’ structure and location. Besides user also must master the syntax of structured query languages such as SPARQL, where she can only pose simple and intuitive natural language queries [39]. Even though there are some approaches which enables the creation of heterogeneous, distributed NLP applications, which use the Web as an integration platform [47], still these approaches need to be integrated for the full exploitation. Queries might span over several datasets and current understructure does not allow to span the query over distributed data sources. Beside each triple pattern relevant to the query might be discovered in different data source which cannot be solved in current structure as well [68].

Besides, the availability of live queryable knowledge graphs on the Web still appears to be low. A significant amount of Linked Data knowledge graphs is therefore not reliably queryable, e.g., not published in queryable form and nor easily accessible, e.g., knowledge graphs that are published in a public SPARQL endpoint suffer from frequent downtime [87]. Thus, it is necessary to select the sources which will be used for further query processing step.

Source selection is the identification process of a relevant set of data sources for the query. As remarked in [12], “the notion of relevance is at the center of information retrieval" which puts the importance of selecting sources likely containing the requested information. Source selection usually involves pre-computed summaries and/or the retrieval of extra (meta-)data from the endpoints, and happens as a separate step before the actual execution [87].

As a consequence of granting the dereferenced URI as a virtual data source, there are extreme number of data sources which increases the dynamicity, heterogeneity, descriptions and access options continuously [57]. However, data source selection algorithms *i)* own high computational complexity and can hardly scale to millions of data sources *ii)* have to be handled during data source selection due to the effectiveness *iii)* the large amount of autonomous data sources cause imbalance data quality and complex overlapping relationship among them [59]. Source selection step is intended to reduce the number of requests to servers, however, an overestimation of such sources increases the network traffic, the number of intermediate results and thus can significantly increase the overall query processing time [80]. Besides, the data in the sources might not be relevant anymore, e.g., they might not be fresh or correct etc.

There are several approaches to tackle the source selection problem with the focus of query processing techniques in Linked Data. Hartig [45] discusses these techniques to provide a systematic overview and categorize these techniques. The author distinguishes the approaches in three main topics, live exploration which has a main characteristic of possibility to use data from initially unknown sources, index based approaches which use data structures that index URIs as pointers to the data, and hybrid approaches which combine both index-based and live exploration approach:

Live Exploration: For a given query, these approaches perform the URI look-ups at execution time to find further URIs. Unlike the web crawling approach, which traverses the Linked Cloud with the purpose of pre-processing and indexing the web data, in the look-up based approaches, a link traversal process is performed to discover the relevant sources w.r.t. the given Linked data query. The approach begins by dereferencing URIs found in the query itself and searches further in its related links [44]. These approaches can also establish a base structure for the index or mixed approaches.

Index based approaches: A more stable approach to find the relevant sources is indexing the URIs without taking into account the graph structure of data. This group of approaches keep the copy of remote Linked Data source contents in the local repositories [70, 71]. The content of the retrieved sources are used to discover the most relevant sources during query processing. Therefore, some statistical and metadata information are kept to improve source selection. Comparing to the centralized indexing approach, the aim is not indexing the data itself but using these data for source selection process by summarizing the data. These indexes are implemented by either summarizing the data as a whole inverted index and schema index [85, 71] or approximation index approach such as multidimensional histograms or QTree structure [85]. Umbrich *et al.* [85, 43] declare the superiority of the QTree approach over other index based approaches.

Hybrid approaches: These approaches represent a mixed strategy of index based approaches and live explorations to overcome the drawbacks of both approaches. Ladwig and Tran [**?**] proposes an approach which benefits from both query planning and retrieving, discovering new sources for performing the live queries on Linked Data. They exploit an index structure which is extendable by the online query processing, such that, the newly found URIs are added to the index and included in the future queries.

In this thesis, we exploit aformentioned index based approach for source selection, named QTree [43]. This structure adopts the data summaries approach which collects the description of content and schema of the data sources and enables the selection of the relevant ones for answering a query expressed in terms of a triple pattern. Summary indexing allows to reduce the large volumes of data into smaller subsets efficiently, instead of keeping all the data in the index. Moreover, the ability of indexing instance-level data beside schema-level data allows more specific features to answer the queries. It supports queries not only on the structure but also on the content, thus, a fine-grained answering mechanism is achieved without sacrificing the efficiency.

4.2 QTree

QTrees [43, 85] have been proposed as an approach for efficiently determining which sources may contribute answers to a query in live distributed query systems. The data structure allows the query processor to decide on the relevance of a source with respect to a given query. Cost of query execution can be decreased by querying only relevant sources instead of all available ones in an environment with huge amount of sources.

4.2.1 QTree Data Structure

A QTree [43, 85] is a hierarchical data structure which is proposed for top-k query processing in P2P systems [93]. The structure is a combination of multi dimensional histogram (MDH) [50] and R-Tree [42], such that, leaves of the R-Tree are replaced with the MDH to provide a data reduction method by using data summaries in an approximated way and to prevent the high memory consumption. R-Tree is a tree data structure used for spatial access methods, i.e., for indexing multi-dimensional information. The idea is organizing the neighboring objects together and represent them in the rectangle in the upper-tier encapsulating them. On the other hand, MDH is a compact structure to summarize the big amount of data by providing approximate query answering. In MDH, the numerical space is partitioned into regions and regions are partitioned into leaf nodes which include the statistical information about the data.

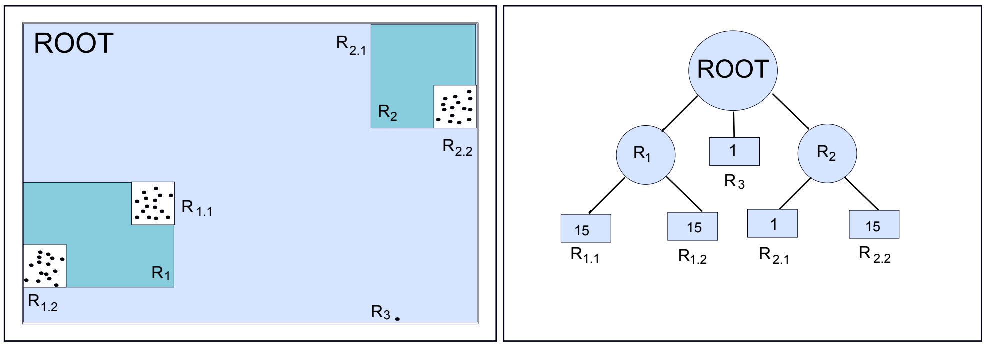
There are several benefits of QTree w.r.t. the R-Trees or MDH to overcome some disadvantages of both structures. QTree provides statistical information about triples while R-Tree indexes triples themselves by their exact coordinates and detailed information. As a result, memory consumption of an R-Tree increases with the number of triples, while space consumption in QTree increases with only the number of data sources. On the other hand, QTree takes some advantages of both MDH (e.g., indexing multidimensional data, capturing attribute correlations), and RTree (e.g., dealing with sparse data, building a hierarchical structure). Thus, QTree enables efficient look-ups for the queries via combining these two structures.

Comparing to histograms, QTrees do not necessarily cover all the data space but only regions containing data. Thus, in case of sparse data the histograms introduced above use same sized regions for areas representing many data items as well as for areas containing only a few data items. QTrees, however, use regions of variable sizes covering only areas containing data [82]. Every node in the QTree is of type *QTree Node* referring to a region in this data space which consist multidimensional spaces represented by minimum bounding boxes (MBB) in the data structure. The MBB is the smallest container in (n) dimension to keep the set of points in the nodes of the QTree and has a 2xdim vector that stores minimum and maximum boundaries for each dimension[93]. The number of nodes is limited in the QTree to avoid memory consumption which is achieved in two ways: 1. by defining a maximum number of child nodes (f_{max}); 2. by defining a maximum number of leaf nodes (b_{max}). The algorithms are designed in such a way, that they ensure the value of current buckets is always lower than the value of maximum number of buckets [93].

Although QTrees have the same functionality of organizing data in multidimensional regions as in RTrees, they have a main structural difference: while RTrees index the actual data by keeping detailed information, QTrees show an approach to approximate the data by keeping statistical summaries. Leaf nodes are the specialized region members so-called *buckets* which contain multi dimensional space in the QTree. A bucket contains statistical information about the data element contained in the respective region. The data elements are the triples, thus, regions are in three dimensions for subject, predicate, and object elements. Even if there are several possible approaches to keep statistical information, a straightforward approach is to maintain a number of data items per region and a list of sources contributing to this count. It has been shown that source selection performs better if a set of pairs $(count, source)$ is maintained, denoting that *count* data items provided by the source *source* are represented by the region [85].

Tree hierarchy can be seen in Fig 4.1.(b) with regions (R) which are represented by nodes, and buckets which are represented by leaf nodes. Fig 4.1.(b) shows an example of hierarchical representation of the points, buckets and regions presented, and Fig 4.1.(a), shows the corresponding QTree in a two-dimensional space, for the sake of simplicity. The overall space, represented in the tree by the root node, has three child regions R_1, R_2, and R_3, corresponding to a first partition of the space. Regions R_1 and R_2 are further split into two child nodes. Leaf nodes corresponds to buckets keeping information about the number of elements in the corresponding region.

A drawback of the QTree is its inability to handle the approximation of the sources. A URI in the QTree is approximated to a hash value according to the prefix trie structure. However, two different URIs with similar prefix might be approximated to the same value which *i)* increases the number of the same valued approximations for the URIs *ii)* increase the number of sources located in the same bucket *iii)* increases the number of sources returned by the query. Another drawback is the quality of the sources returned by the index. More than one source might provide the answer



(a) Data and Regions (b) Hierarchy

Figure 4.1: Two dimensional QTree Example [43]

to the query, but according to the which criteria these sources should be more prominent? If a source is no longer available then

4.2.2 QTree Insertion Operation

A QTree can be initially constructed by pre-fretching, i.e., by crawling the seed URIs from the web via a web crawler, or by SPARQL queries, through direct look-up. Starting from an empty structure, data provided by the sources are obtained and incrementally mapped to the QTree regions and buckets with respect to similar URIs, as described above.

Triples are mapped into the QTree by using hash functions to provide the quick look-up facility and data retrieval operations. Insertion of a new key (s,p,o) requires a two-step process, such that, first components of the triple should be hashed into numerical values, and then these values should be replaced to the positions in the QTree corresponding to their values. Components of the triples are hashed into tables in a very fine-granularity level by using prefix and mixed (prefix-string) hashing functions. These numerical values are positioned in the three dimensional space forming a data point. On the other hand, these data points are related to their data sources to provide the effective source selection process. Thus, the structure proceeds from the root to place the proper bucket for insertion recursively. However, there are several procedures to insert a triple:

- *Inserting triple to the bucket:* This is the first and easiest option to apply, when there is an existing bucket which encapsulates all the hashed values of a triple. In this case, the triple is inserted with related data source and count value of the data source is increased.
- *Inserting triple as a new bucket:* If such bucket that encapsulates the triple does not exist then the search stops and instead most responsible child node is found in the branch. Thus, triple is inserted to the region as a new bucket and triple count is increased as previous one (1, data source).

- *Enforce maximum child node constraint:* If a new bucket is inserted as a child node then it should be checked whether QTree exceeds the maximum number of child nodes. In the case of exceeding, the system should call the method to merge the nodes to decrease the number of child nodes.

- *Enforce maximum bucket constraint:* Eventually, when a new bucket is inserted, it should also be tested whether QTree exceeds the maximum number of buckets. In the case of exceeding, the number of buckets are reduced to optimize the tree by decreasing the number of buckets.

4.2.3 QTree Search Operation

An advantage of the QTree is that they support estimation of relevant sources by processing parts of a query plan before actually fetching any sources. The QTree can be exploited to select relevant sources with respect to basic graph patterns (BGP) of SPARQL queries, namely sets of triple patterns, from the graph for a good selection [85].

4.2.3.1 Simple Search Operation

In this subsection, the queries with only one triple pattern are referred as simple search operation. When a query is executed, the BGPs contained in the query are converted to numerical values and searched in the QTree starting from the root node. This conversion builds a range query over the tree which constructs a cubic region corresponding to the triple pattern multi to provide multi dimensional space comparison.

Algorithm 1 shows how hashing operation is performed for the BGP of a Sparql query [43]. A new query space is defined for each BGP to find numerical representation. Conversion of the BGPs are slightly different than conversion of the triples to hash values, since BGPs might contain variables. Thus in the algorithm, while URI elements are converted into the constant numbers, the variables are converted into the range of QTree space dimensions with the focus of finding any numerical value in the tree w.r.t. the corresponding tree space (Lines 3-9).

Once the range query is executed on the buckets, all buckets contained in the data summary that overlap with query space can be determined. Thus, the output of the source selection algorithm is a set of buckets, each annotated with information about the overlap with the queried region, source URIs, and the associated cardinality [43].

Algorithm 1: Hashing BGP to Find Query Space

```
   Input  : BGP b, QTree space treeSpace
   Output : Query space qs
1  begin
2      int[] qs= new int[3] ;
3      for i=0 to 2 do
4          if b[i] $\neq$ variable then
5              qs[i].low=tripleHash(b[i]); qs[i].high=tripleHash(b[i]) ;
6          else
7              qs[i].low=treeSpace[i].low; qs[i].high=treeSpace[i].high ;
8          end
9      end
10 end
11 B = $\phi$; for Bucket B $\in$ QT : B overlaps qs do
12     O = B.overlap(qs);size(O);
13     $c_O = c_B \cdot size(O)/size(B)$;
14     $B = B \cup \{(O, c_O, S_B)\}$;
15 end
16 return B
```

4.2.3.2 Join Search Operation

The buckets determined for single triple patterns act as input for the join source selection. The join operation is performed on buckets by discovering overlapping regions. A join between two triple patterns is defined by equality in one of the dimensions. Overlaps between the sets of obtained relevant buckets refer to the the triple patterns with respect to the defined join dimensions. If a bucket is overlapped, not to miss any relevant sources, it is assumed all sources from the original bucket to be relevant [82].

The result of a join evaluation over two triple patterns is a set of three-dimensional buckets. Joining a third triple pattern requires a differentiation between the original dimensions, because the third triple pattern can be joined with any of them. For instance, after a subject-subject join we have to handle two different object dimensions; a join between two three-dimensional overlapping buckets results in one six-dimensional bucket with an MBB that is equivalent to the overlap. In general, a join between n triple patterns results in a (3 · n)-dimensional join space [82].

Algorithm 2 presents the source selection by joining the buckets in the space using found join coordinates. First of all, a new class is created from the *Query Result Estimation* class to put the results from the search evaluation. The first BGP of the query is converted into the query space which represent hash numbers (Line 3-4). Later, a search is performed in the QTree to find the overlapping buckets with this first query space. Resulting buckets are retrieved by intersection information and new intersection coordinates of the buckets. Intersection coordinates of the overlapping buckets are found by taking the minimum of the upper boundaries and maximum of the lower boundaries among query space and resulting buckets. Therefore, the new intersection space is used as the overlapping bucket, and, the intersection information is represented as a ratio by dividing the size of overlapping bucket to resulting bucket $(size(O)/size(B))$. The number of sources coming from these buckets are added to the result (Line 5-7).

Algorithm 2: Evaluating Query on QTree

```
   Input  : Bgp Hash Coordinates bgps, Join join
   Output : Relevant ranked sources
1  begin
2      Function evaluateQuery(bgps, join)
3          Create a new QueryResultEstimation as result ;
4          QuerySpace querySpace =getQuerySpace(bgps[0]);
5          Find Left Buckets(querySpace);
6          Create a new 3D JoinSpace as temporaryResult ;
7          Put left result to temporaryResult;
           // if the left result is empty, the whole join is empty
8          if null != temporaryResult then
9              for int j = 1; j < bgps.length; j++ do
10                 QuerySpace rightQS = getQuerySpace(bgps[j]);
11                 Find Right Buckets (rightQS);
12                 temporaryResult = computeJoin(temporaryResult, join[j-1][0],rightResult,
                   join[j-1][1]);
13                 Put temporaryResult to result;
14             end
15         end
16         Rank Sources(currResult);
17 end
```

Then the join part of the search is performed only if there are results from the first query space. The join operation is performed for the rest of the BGPs of query respectively (Line 10). The steps which are performed for first BGP to find overlapping buckets are repeated for the following BGPs (Line 11-14). In case of not having any overlapping buckets for this BGP,the loop is terminated. Otherwise, these resulting buckets are joined with the left result in the Line 16 by computing joins of these two different BGPs. In this algorithm left and right buckets (which are defined as Find Left Buckets or Find Right Buckets methods) are found according to the Algorithm 3. In this algorithm for each query space the list of buckets are compared to find the intersected buckets. When the number of buckets are higher the comparison is also increases accordingly.

Algorithm 3: QTree bucket search

```
   Input  : Query space qs, list of Bucket bucketList
   Output : List of relevant buckets as result
 1 Function FindRelevantBuckets (qs, bucketList)
 2     foreach QTreeBucket currentBucket : bucketList do
 3         Bucket[] result; Space intersection = currentBucket.intersect(qs);
 4         if currentBucket.boundaries overlap qs then
 5             result.add(computeIntersectionInformation(currentBucket, qs));
 6         end
 7     end
```

Join operation is performed as an intersection between 2 or more bucket regions. Since each query space is converted into a region via query spaces, intersection of each BGP with a bucket represents a whole or partial intersection result of the bucket. As a result of this intersection, overlapping space is computed to find the partial cardinality of the source to the intersection space. However, since this is an approximated approach and only the source and number of information information are kept in the bucket, it is not possible to leave any source out, thus, the cardinality is computed as an estimation of the possible number of triples namely cardinality.

Algorithm 4 shows how this cardinality is computed in the source level during join operation. The found buckets as a result of the intersection between query space and QTree regions are further computed for a possible overlap between them by using cartesian product (Lines 6-7). This computation is also followed by the statistical information which are kept for each bucket (Line 8). The $\otimes$ operation shows the cardinality computations between right and left buckets while increasing the dimension of the join. Cardinality of the each bucket is computed separately for right and left bucket depending on their contribution percentage, and then, found result is divided by the intersection bucket dimensions which is computed by subtracting high and low boundaries for the intersected buckets. Line 9 shows the returned intersected bucket information which are overlapping bucket information, overlapping cardinality and union of sources. Following these computations ranking of the sources are computed for the top-k sources.

Algorithm 4: Nested-loop join

```
Input  : JoinSpace $J_{in}$ (left-hand side), left join dimensions l, set of buckets B (right hand
         side), right join dimensions r
Output : New join space (containing relevant buckets and sources)
1  begin
2      Function computeJoin($J_{in}$, l, B, r)
3          $J_{out}$=null;
4          foreach Buckets $L \in J_{in}$ do
5              foreach Buckets $R \in B$ do
6                  if $\exists O_L$ = L[l].overlap(R[r]) then
7                      $O_R$ = R[r].overlap(L[l]);
8                      $c_{O_R \otimes O_L} = \frac{c_L \cdot \frac{size(O_L)}{size(L)} \cdot c_R \cdot \frac{size(O_R)}{size(R)}}{max(L[l].hi - L[l].low, R[r].hi - R[r].low)}$;
9                      $J_{out} = J_{out} \cup \{O_R \otimes O_L, c_{O_R \otimes O_L}, S_L \cup S_R\}$;
10                 end
11             end
12         end
13         return $J_{out}$;
14 end
```

4.2.4 Ranking of the Sources

The source selection of the relevant buckets results as an approximation of the sources because the buckets arrive with a large number of data source and it is not possible to know which one has higher contribution to the result. On the other hand, some of these sources are irrelevant because it is impossible to exclude the sources without being sure of irrelevancy and they become eventually the false positive results. Therefore, it is essential finding the most relevant ones among them to provide the meaningful results.

In the original QTree approach, cardinality measures are used to rank the sources to find the relevance to the query,such that, the ones which have more contribution to the result will be ranked higher in the list. Contribution of each source to the join result by adopting Line 8 in Algorithm 4. Thus, it is achieved by computing the aggregation of number of triples which are multiplied by intersection ratio of the buckets. It is assumed to have a uniform distribution to enable a cardinality estimation. This new source cardinality information is also computed and kept for each source in the new intersected bucket. During ranking of the sources, cardinality of the source c_B^s is summed as it can be seen in Equation 4.1. For each source this formula is applied to find the cardinality of each source over all joined triples.

$$s_r = \sum_B c_B^s \tag{4.1}$$

The computation can be seen on Algorithm 5. Each intersected bucket, which is created as a result of join operation, is iterated to retrieve sources contributing to the result. In the second loop, each source in the bucket is iterated with its intersection information. They are assigned to a rank list where they are kept with their cardinality count. If they exist in different buckets then the contributions are summed, otherwise they are put as a new source. On the other hand, more than one source might have the same cardinality value, in this case a minor value is added randomly to sources in the second loop and the sources are ranked according to the number of contributions to the result in a decreasing way.

Algorithm 5: Ranking Algorithm

```
Input  : JoinSpace result
Output : List of ranked sources
1  begin
2      Function rankSources(result)
3          foreach Buckets B ϵ JoinSpace do
4              foreach Source src ϵ Bucket do
5                  Double tripleCount = src.getCount() ;
6                  Get source count rankCount in rankList ;
7                  if rankCount.isNull then rankCount ==0.0;
8                  rankCount+=tripleCount ;
9                  rankList.put(source,rankCount);
10             end
11         end
12 end
```

4.2.5 Importance of Hashing

QTree exploits several structures to achieve good approximation and prefiz trie is a crucial part of this structure. A trie is a tree for storing strings in which there is one node for every common prefix where the strings are stored in leaf nodes. A prefix hash trie *(PHT)* is a distributed data structure that enables sophisticated queries over a distributed hash table. A key benefit of tries is that they cluster semantically close data items for efficient processing of range queries. In the QTree structure, this trie is used to hash the URIs (subject and object) to the numerical values during insertion of the data and query processing.

In the QTree structure, a PHT [74] is constructed for the triple elements by using the resources which are sampled from the web before constructing the QTree, thus, the PHT data structure is a binary trie created over the data set. Each node of the trie is labeled with a prefix of the string using all the elements in the data set. The structure represents a binary trie, such that, in each level the common prefix inside the dataset is found and assigned to the common node and then for the different part of the string, the trie is partitioned to two child nodes appending *0* or *1* to the edges. The difference from a tree is the numbers of the edges arriving to a node represents a path to locate the prefixes in the nodes which simplifies the link traversals.

Then, this prefix trie is used to find the corresponding numerical representations for the data elements (triples) during insertion and search. In the prefix trie structure, URIs have a binary representation as a result of the path in the trie and a mapping scheme is provided that calculates a binary representation from a URI string to hash value. Thus, when inserting the triples into the QTree each subject and object URI element are mapped as points (coordinates) in the multidimensional space. On the other hand, basic graph patterns of the query are converted to hash numbers during search operation to select the relevant spaces for the given query.

Prefix	Hash Values
	1
→ http://purl.org/ontology/pointOfInterest/	2
→ http://purl.org/ontology/pointOfInterest/History	3
→ http://purl.org/ontology/pointOfInterest/History/Museum/	4
→ http://purl.org/ontology/pointOfInterest/History/Museum/ScienceMuseum	5
→ http://purl.org/ontology/pointOfInterest/History/Museum/ArtMuseum	6

Table 4.1: Prefix Hashing

Algorithm 6 [4] shows the structure to find the binary representation for the given query or triple element. Search starts at the given entry node of the prefix trie and recursively parses down to find the requested data element. The aim of the algorithm is mapping the string URIs into the numerical numbers, thus, in each branch URI is compared to the node lexigrophically which has a prefix representation to calculate a binary key. The algorithm takes the root node of the trie and the element to be searched inside the trie as input and returnes the binary representation of the element as a result. In the first place if the node does not have a right and left child the binary result is 0 (Line 3). If the element equals to the root node then the binary result returns 0 (Line 4). In each cases the computation is terminated. For the last, the element is compared to the node to see whether it is lexicographically bigger of the node prefix to find the binary representation.

The comparison is made through the whole trie recursively until finding binary representation for the string. If the string element is lexicographically smaller than node then 0 is appended and the trie is searched through the left child of the current node, otherwise 1 is appended and the trie is searched through the right child of the current node to search the right subtree. In every step, the current number is added to find the binary representation of the string element. This binary representation shows the path which reaches the element. Finally, the binary hash key which is represented as string is sent to another method to be parsed as a signed decimal long representation. Thus, it is converted to integer to find a decimal number as represented in the Table 4.1.

Algorithm 6: Finding binary hash keys from Prefix Trie

```
  Input  : String URI, TreeNode node
  Output : String binaryHashKey
1 Function findKey(URI, node)
2     if (node.LeftChild == null) AND (node.RightChild() == null) then return "";
3     if (node.getPrefix().startsWith(URI)) then return "";
4     if (node.getPrefix().compareTo(query) > 0) then
5         return ("0" + findKey(query, node.getLeftChild()))
6     else
7         return ("1" + findKey(query, node.getRightChild()))
8     end
```

4.3 Extended 3 Dimensional QTree

As the size of data increases usage of accurate and comprehensive metadata gets more attention to understand and use the content in full extent. Beside, indexing only relevant piece of information is important not to increase the index size and to improve search performance. Thus, as discussed in Chapter 1, in this thesis we aim at extending our index structure with relevant information.

An important aspect for source selection is quality and amount of relevant sources returned for a given query [85]. Even though QTree is a highly scalable structure, there are some drawbacks which courages us to extend the structure to index Linked Data sources. First of all, as it has been declared in the original QTree, source selection is an approximation approach which overestimates the results and returns too many false negatives [85]. On the other hand, they compute the relevance of the source by counting the number of triples contributing to the query result. This is a good way to compute relevance, however, a source which has high number of triples indexed in the structure might mislead the results as well. Another reason is that the structure does not provide a method to assess the source quality. Thus, we propose an approach to extend the QTree to decrease the number of found sources as well as discovering the more relevant to the given query.

For this reason, we propose to extend QTree structure to see the exploit quality and context information. Context information which is gathered through DBpedia SKOS categories are associated with the source inserted into the bucket, such that, it would be possible to use it during selection process. On the other hand, we further add quality information which may be adopted to the user dependent quality search. Data quality allows framework to manage the sources by the quantified measures depending on the several user defined quality metrics. In the next subsection we will further discuss the new structure and relevant operations.

4.3.1 Extended 3D QTree Data Structure

In this section we will discuss extended QTree structure which has all the functionality of original QTree which has already been described in Section 4.2 with additional context and quality values. This extension of the QTree is aimed at:

- selecting sources by aggregating complex source statistics on large datasets
- assessing the impact of context and quality information on source selection in an extended base
- achieving performance results without sacrificing efficiency comparing to the original structure

In addition to original QTree structure, inner nodes also keep the quality scores of the sources to accelerate the search operation and provide a quality dependent search. Search with pruning is inferred from situations where search systematically discards a sub-tree which doesn't meet the quality threshold. Since it would be highly costly to keep these scores for each source separately, instead, a range information is kept in the descendant nodes for each quality dimension. Starting from leaf nodes where sources are indexed locationally, quality scores are first discovered and placed into an interval which later are propagated to the inner nodes finally arriving to the root node.

Leaf nodes specialized as buckets have the same functionality of keeping statistical information for data reduction. The sources are indexed as well in this structure instead of indexing triples themselves, however, source statistics are extended in this structure to include also context and quality information. The context information is captured in the data as we have discussed in Section 2. The sources which are associated with concepts via subject metadata have been gathered using SKOS information. These SKOS concepts are related with the sources in this step to provide a set of context information for the sources. Beside set of quality scores is computed for the source using Luzzu framework. Number of triples which are inserted to the bucket is also kept as it has been in the original QTree, thus we have an index of the source as (source, set of source contexts, set of quality scores, number of triples). We define 3D dimensional extended QTree bucket as follows:

This statistical information allows us to compute an overall relevance score for the given query, such that, we consider here two types of criteria: *i)* Source specific criteria depends on the overall data quality of the source and computed according to the pre-determined metrics which we discussed in Subsection 3.3. *ii)* Query specific criteria determines the relevancy of the source to the given query and it is computed on the fly during query execution. We will further discuss how we use this statistics in Section 4.3.4

This piece of information allows us to select the sources wisely taking into account query circumstances. In the following subsections we will discuss how these metadata about the source will be used to insert and select the sources.

4.3.2 Extended 3D QTree Insert Operation

In this data structure, insertion of new records into extended structure is based on the same philosophy that governs insertions into the original QTree with a difference of statistical information. The triples are inserted by hashing triples as well to numerical representations. At each node, a comparison is made recursively to see if the numerical representation of the triple is contained within any bucket. If there is no bucket which has dimensions covering triple then a new bucket is created. If there are more than one bucket then it is being assured that the triple is located in only one bucket. Since this structure is extended to include also context and quality information, by each triple which is inserted into the tree, bucket statistics are updated to build summaries for RDF data sources by further information.

If a triple is found to be in the range of a bucket, then its source is inserted into the bucket with its specific context, quality and the number of triple information. Statistical information is inserted as follows:

- If the source is not inserted into the bucket, first retrieve the quality information, following retrieve the context information associated with the source and put it in the bucket.
- If the source is inserted into the bucket then update the number of triples associated with the source.

After the insertion of the triples, quality information in the leaf nodes are propagated to the inner nodes to provide a quality dependent search. Algorithm 7 shows the computation of the quality ranges for the nodes. A depth first search algorithm is used to propagate the quality score ranges to the inner nodes. Starting from the extended QTree root each node is recursively visited until the bucket nodes (Line 1-4). Later, it is checked to see if the node is bucket or inner node (Line 5). If it is a node then its child node intervals are used to set node quality interval (Line 5-8). If it is a bucket then for each bucket every source is iterated one by one to retrieve their quality values for each quality measure (provenance, availability etc.) and then set bucket interval. Since it is proposed to keep more than one quality indicators, these quality intervals are computed and set with multiple quality information. This range information is kept in the nodes for each quality dimension (provenance, freshness etc.) as the range of bucket by their lower bounds and then by their upper bounds.

Algorithm 7: Extended QTree QTree Quality Propagation

```
   Input   : Extended QTreeRoot qTreeNode
1  Function IterateQuality (qTreeNode)
2      foreach QTreeNode node children of qTreeNode do
3          IterateQuality(node) ;
4      end
5      if node is not a leaf then
6          if foreach bucketNode in node.children then
7              set node quality interval from child nodes;
8          end
9      else
10         foreach source src in Bucket b do
11             get quality score for each quality measure ;
12             if b.nodeInterval is empty then
13                 set bucket quality interval for each quality measaure;
14             else
15                 change bucket quality interval for each quality measaure;
16             end
17         end
18     end
```

4.3.3 Extended 3D QTree Search Operation

This structure is modeled in such a way that it would be possible to use context and quality information in the source selection approach. Thus, we consider the query as a combination of SPARQL query, context information and quality threshold.

In this extended structure, we utilize the quality information during search process in the QTree structure to prune the branches which do not satisfy the criteria of quality threshold. The BGP which is converted to numbers are recursively searched in the tree starting from the nodes to find the relevant buckets. But beside the range query, the nodes are also required to be checked to discover if the sources satisfy given quality threshold. In this case, it is important to have the nodes which provide at least one source within the threshold. However, the drawback of this approach is that since nodes provide the sources with different quality variety, it is not possible to prune the nodes even though when they have lower quality sources. We can only prune the nodes when they have all the sources under the threshold. Beside join operation becomes more costly since all the context and quality information of the sources should be also moved into the joined buckets. Even though we use the quality information, context is not taking a main role in this structure during search, thus, we use it during ranking which we will further explain the next subsection.

Algorithm 8 presents the search algorithm which returns the buckets whose reference region intersects the query space. This is one of the main differences from original QTree, such that, while this search is performed over all buckets in the QTree, in the extended QTree the nodes are iterated recursively to prune unnecessary nodes or buckets. The search is accomplished by traversing the nodes recursively by respecting to the quality condition. The search starts from the root node and takes the children of the node for further search (Line 3). First step is to check whether the child node is leaf or inner node for advance search in the sub-nodes of the tree (Line 4). If the node is an inner node, overlapping method calculates the intersection of the current node with the given query space to see whether there is an overlap (Line 5). In addition to original QTree structure, inner nodes also keep the range information of the sources in the descendant nodes for each quality dimension to accelerate the search operation. Thus,the node must ensure the condition of quality w.r.t. the given threshold (Line 6). If the quality range of the node cannot satisfy the threshold, the node is pruned for further search. If the node satisfies these both conditions, then the search of the node is processed. If the node is a bucket, then the same comparison of the boundaries is performed in the same way, however, if the bucket provides the conditions then it is added to the list of bucket which might contribute to the source selection process. Thus, the output of this process is a set of buckets with overlapping information. Overlapping information consists of the number of triples in the buckets and intersection information.

Algorithm 8: Extended QTree Basic Graph Pattern Search

```
Input   : Query space qs, Quality threshold th, Extended QTreeRoot qTreeNode
Output  : List of relevant buckets
1  Function FindRelevantBuckets (qTreeNode, qs, th, bucketList)
2      foreach QTreeNode node children of qTreeNode do
3          if node is not a leaf then
4              if node.boundaries overlap qs then
5                  if foreach quality dimension node.qualityLow $<$ th AND node.qualityHigh
                   $>$ th && th $<$ node.qualityLow then
6                      FindRelevantBuckets(node, qs, th, bucketList) ;
7                  end
8              end
9          else
               // this is a bucket
10             if node.boundaries overlap qs then
11                 if foreach quality dimension node.qualityLow $<$ th AND node.qualityHigh
                   $>$ th && th $<$ node.qualityLow then
12                     bucketList.addResult(node) ;
13                 end
14             end
15         end
16     end
```

4.3.4 Extended 3D QTree Ranking

In the source selection systems, a list of sources are presented to the user to enable a more efficient analysis. However, one must decide which of the retrieved documents are relevant to the information they are looking for to proceed with the relevant information. In this subsection, we are looking for the answers for the questions *i)* can we find the sources with highly ranked contexts w.r.t. the query contexts (domains of the user interest)? *i)* can we filter/rank lower the sources unrelated contexts to the query?

In the scope of this work, we aim at improving retrieval performance by using query contexts which are derived from explicit or implicit user context information. This process allows us to ease of discovering suitable sources for the queries and rank them in the top-k results by assigning context-aware ranking scores computed using relations between context nodes in the context ontology model.

Having discovered the intersected buckets, we now can compute scores for data sources, finding the most relevant ones among them to provide the meaningful results. Thus, we exploit from context and quality information to rank the sources according to multi criteria because we have more than one measure to assess the relevancy of a source. Given a query, if the returned source addresses the stated information need, then, the source is relevant [63]. In our case, these criteria are: context similarity, quality score, and source cardinality. The idea is ranking the sources w.r.t. the number of triples related with each context and quality information. Thus, we compute each criteria as follows:

- Context Distance described in Definition 2.5 is computed to find the shortest path between source context and query context for a given query. We compute the shortest path using least common ancestor (lca) of the context nodes for the given contexts c_1 *and* c_2:

$$d(c_1,c_2) = d(lca(c_1,c_2),c_1) + d(lca(c_1,c_2),c_2) \tag{4.2}$$

- In the scope of this thesis, dereferencability metric is used to assess the quality of the data sources. Quality score for dereferencability metric is computed as follows:

$$qs^{S}_{der} = dereferencableTripleNumber\ in\ a\ source/totalTripleNumber\ in\ a\ source \tag{4.3}$$

- Cardinality is the number of triples per source which contributes to the query result depending on the size of the intersected bucket. In our specific case, higher cardinality implies a higher relevance of the source.

In this study, the relevance score is computed for each source via linear combination. Changing the weights of the criteria allows us to change the importance of the criteria dynamically, e.g., giving a higher weight for the context similarity means that context similarity has higher importance. In the scope of this work, we define a ranking function introduced as linear combination which aggregates mentioned three criteria and produces a score for each source.

Definition 4.1. *(Multi Criteria Ranking Function) $R(Q,S)$ is a ranking function which associates a score with the pair (Q,S), where Q is a query and S is a source. Let assume $v_1, v_2, v_3, ...v_n \in \mathbb{R}$ are variables and $c_1, c_2, ..., c_n \in \mathbb{R}$ are scalars (or weights). A ranking function is a linear combination of variables and scalars which is defined as $R(Q,S) = c_1v_1 + c_2v_2 + \cdots + c_nv_n$, where $\sum_{i=1}^{n} c_i = 1$.*

The join space results are sent to the rankSources method in Algorithm 9 to rank the sources according to these join space results. Every source coming from the join space with context , quality and number of cardinality is summed up to find an overall ranking value for the sources.

Algorithm 9 shows the ranking methodology for the sources. In the first loop (Line 3-10), each source in the overall buckets are iterated to assign to find their contribution to the buckets, beside assigning their quality and context values at the same time. In the second loop, (Line 12-15) contexts of the sources are compared to the query context. Maximum context similarity of the source context to the query context is found by computing the shortest path between them which requires the minimum distance between compared contexts. Beside maximum context value is found for the overall sources to normalize their value. In the third foreach loop (Line 17-23), quality score, context similarity and source cardinality are used to find a source relevance value by using linear ranking function. These three criteria values are all normalized according to their maximum values before computation to ensure that criteria dependencies are consistent and logical. In the last loop, to avoid same relevance value a small difference value is added for the sources which have same relevance value.

Algorithm 9: Ranking Algorithm

```
   Input  : JoinSpace result, Query context queryCxt
   Output : List of ranked sources
1  begin
2      Function rankSources(result,queryCxt)
3          foreach Buckets B ϵ JoinSpace do
4              foreach Source src ϵ Bucket do
5                  Double tripleCount = src.getCount() ;
6                  Get source count rankCount in rankList ;
7                  if rankCount.isNull then rankCount ==0.0;
8                  rankCount+=tripleCount ;
9                  rankList.put(source,new ContextQuality());
                   rankList.get(source).setCount(rankCount);
                   rankList.get(source).setContext(); rankList.get(source).setQualityValues();
10             end
11         end
12         foreach Source src ∈ rankList do
13             Compute source triple count as tripleCount;
14             Compute maximum context similarity as maxSim;
15             Retrieve source quality as sourceQuality;
16             Find mean source relevance(tripleCount, maxSim, sourceQuality) as
               sourceRelevance;
17             sourceList.put(source, sourceRelevance) ;
18         end
19 end
```

We have now found a criteria for improving the source selection which guides us to the question that remains is whether can we do better. In the next section we will discuss another version of the Extended QTree to have more optimal results.

4.4 Extended 4 Dimensional QTree

Especially with the help of social media, everyday more information is published with several context information e.g. location, time etc. At the same time, queries are personalized according to several private and contextual information. In the first approach, we extended our structure to discover the results of context dependency but we did not fully employ the context information during query processing. Since the availability of context information is more and more relevant, we consider an extension of QTree that fully exploits context information.

In this second approach, we propose to expand the original QTree approach by adding context information and quality metadata in the query processing step. We enriched the triples with contexts thus obtaining a structure which has 4 elements, we further extended original QTree structure by (*i*) associating context information to data and (*ii*) associating quality metadata to the contextualized subsets of data sources.

This new structure helps us in several ways: *i)* it allows us to distribute the portions of data with an efficient context classification *ii)* it allows us to select the sources according their context-dependent quality measures *iii)* it allows us to efficiently prune the tree according to quality measures *iv)* it allows us to discover the approximated number of data items represented by bucket w.r.t. the context relevance which is used to rank the sources. In this section, we discuss the obtained data structure and its operations.

4.4.1 Extended 4D QTree Data Structure

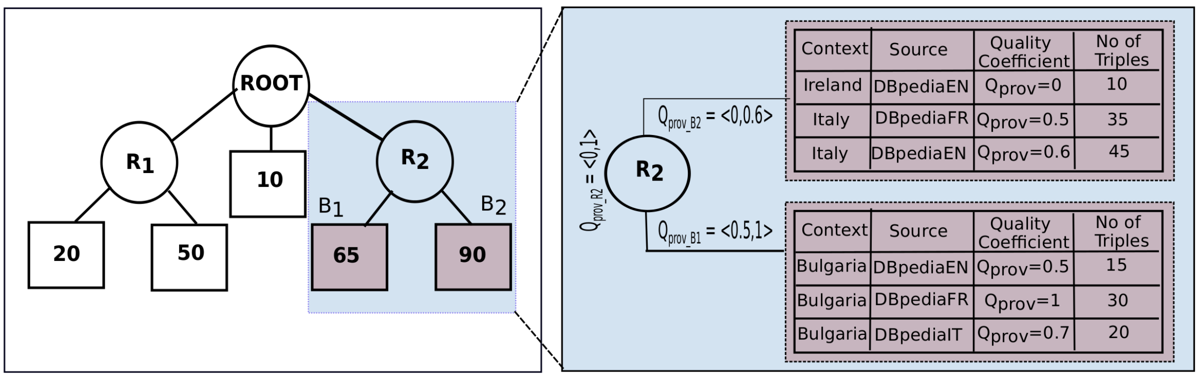
Given a set of quadruples (triples with context), and a set of URLs (U) representing the data sources, an Extended QTree is a directed acyclic graph consisting nodes and edges. Leaf nodes of tree (buckets) contains set of points (p) in 4 dimensional space for the set of quadruples representing location. Difference from the QTree structure is, while QTree hashes triples, extended QTree hashes the quadruples to the tree structure which forms a 4 dimensional space. 4th dimension in QTree represents the triple in a context.

The extended QTree, numerical space is also partitioned into four dimensional regions represented by MBBs. So every node keeps the hashed MBB information which correspond to its boundary, such that, having an upper and lower level of hashed s, p, o, c components. Hashed information are kept in the form of [s.high, s.low], [p.high, p.low], [o.high, o.low] and [c.high, c.low] in the QTree. Based on the used hash functions, similar data will be clustered into the same bucket via similar hash values. We define an extended for dimensional QTree Ndode as follows:

On the other hand, while QTree buckets keep the number of triples per source (count, source); extended QTree buckets keep list of sources with the number of triples they contribute to a bucket related with their context information, and multiple quality measures as statistical information. Each source is paired with one or more contexts depending on the SKOS categories $((source, context), quality, count)$. This piece of information allows us to discover the approximated number of data items represented by bucket w.r.t. the context-quality information which is used to rank the sources.

Details of the nodes and buckets are given in Figure 4.2. Fig. 4.2.(a) shows an example of hierarchical representation of the points, buckets and regions presented. The overall space, represented in the tree by the root node, has three child regions R_1, R_2, and R_3, corresponding to a first partition of the space. Regions R_1 and R_2 are further split into two buckets. Figure 4.2.b shows the R_2 node by detail of buckets and its statistical elements. The buckets keep the information of the contexts and the sources belong to these contexts with quality information of the data collection. The rightmost of the table shows the data collection contributed to that bucket from the data source. Later, these triple counts are used to measure the relevance of that source dependent on the context. For the sake of simplicity, we show quality measures for only provenance metric but it is possible store all the quality metrics described in Section 3.3.

Regarding to the statistics, in the bucket B1, there is only Bulgaria context with 3 different sources and their quality values. Moreover, bucket B_2 contains Ireland and Italy contexts. As it can be seen a context might consists more than one source in a bucket, as well as a source might belong to more than one context either in one bucket or devised into more than one bucket. Additionally, a source having various measures for the different contexts can be noted. These measures are also summarized as a range in the node to show the overall quality of the nodes for that specific dimension. Each parent node contains the minimum and maxiumum boundary of its descendent nodes to show the consisted quality values, such that R_2 node range is recursively computed by comparing the minimum and maximum values of B_1 and B_2 buckets. Minimum of these two buckets (0) and maximum of these two buckets (1) are represented in the upper node as a node quality range.



Context	Source	Quality Coefficient	No of Triples
Ireland	DBpediaEN	$Q_{prov}=0$	10
Italy	DBpediaFR	$Q_{prov}=0.5$	35
Italy	DBpediaEN	$Q_{prov}=0.6$	45

Context	Source	Quality Coefficient	No of Triples
Bulgaria	DBpediaEN	$Q_{prov}=0.5$	15
Bulgaria	DBpediaFR	$Q_{prov}=1$	30
Bulgaria	DBpediaIT	$Q_{prov}=0.7$	20

(a) Data, Buckets and Regions (b) Detail of R_2

Figure 4.2: Detail of Extended QTree

4.4.2 Extended 4D QTree Insert Operation

Having this structure, extended QTree is constructed by inserting quadruples to the MBBs for the data summarization. Insertion of the data items to the extended QTree is performed following the same steps with QTree. However, the contributions which are discussed in Section 4.4 are applied in two levels: buckets and inner nodes.

Associating context is achieved by adding fourth component to triples to form quadruples which allows us to keep more comprehensive information about the triples. In the bucket level, two-step insertion process is applied for the triple part of the quadruples, such that, hash functions are thus applied to the contexts as well as RDF triples to obtain numerical representation for each quadruple, and, then these values are placed corresponding region in the extended QTree. So the formed quadruple is placed in the 4 dimensional space. For the context hashing a new prefix trie is constructed using context URIs during initializing phase of the extended QTree.

Context Taxonomy Tree: During insertion of the Triples with Context, we employed Context Taxonomy tree to find numerical numbers for the context URIs. We discussed in Section 2.5 that we label minimum spanning tree according to the post-ordering of the nodes, instead during insertion of the elements, we utilize these post-order labels to hash the contexts into the numeric values. Since number of context URIs are not very high, minimum spanning tree is chosen instead of prefix tree to provide more accurate results.

The insertion process has been performed incrementally for each quadruple. We also update the bucket statistics, by incrementing the number of contained data items per source w.r.t. the context-quality information, in a detailed manner to provide a better ranking of the sources. Each quadruple represented as a point in the four dimensional space is summarized by its source in the bucket level dependent on its context. During insertion process, context dependent quality scores of the sources are retrieved from the quality graph which are produced by the Luzzu framework. This process is performed by executing Sparql query to the context-dependent quality graph of the source to retrieve the requested metric or dimension. Each source is indexed by one or more contexts possessed in its data collections in the buckets.

Having finished inserting triples into the buckets, as discussed in Subsection 4.3.2, source quality scores are propagated into the inner nodes. Such that, the lowest and highest quality scores of the sources in the buckets are discovered and they are added as quality ranges in the leaf nodes and inner nodes of the tree. Different quality dimensions are kept only as a range to contribute to the search operation, e.g., quality range of a node for provenance dimension is $q_{prov} =< 0.2, 0.7 >$ as it can be seen in Fig. 4.2.

4.4.3 Extended 4D QTree Search Operation

This structure is modelled in such a way that it would be possible to use context and quality information in the source selection approach. Thus, we consider the query as a combination of SPARQL query, context information and quality threshold. We associate a Sparql query with context to be able to match with triples represented in context dimensions. It is possible to execute range queries using QTree structures. I can be applied to BGPs as well as contexts to retrieve results according to a multidimensional space. In that case, only one context can be looked up in the tree as well as only one context.

The user performs a SPARQL query consisting of the basic graph patterns (BGPs) which are a set of triple patterns with a subject, predicate and object, some of which can be variables. We assume that the BGPs come with particular context information from the system indicating different concepts (*BGPs+Context*). On the other hand, the selection of the quality dimensions are processed by the user from the list of supported quality dimensions w.r.t. the established threshold. Process of comparing source and query contexts is performed by matching the context of the query to the ones associated with sources in our structures. Quality thresholds are used in the structure to choose the high quality sources.

Search operation in extended QTree follows the same steps as original structure with difference of context dimension and quality pruning. A novel issue in the way we assess and exploit quality, is the focus on context-dependence. The framework relies on context-dependent data quality assessments according to different dimensions, starting from metadata and annotations stored in quality graphs, and an extension of QTrees. Since the triples are inserted with context component in the insertion phase, during search operation the relevant sources are found in the tree w.r.t. context information.

4.4.3.1 Simple Search Operation

Query with context (Definition 2.4) is performed against extended QTree as a means of converting BGPs and context to numerical values. Each quadruple is represented as a range of constants and variables to build the query space. The query space is processed recursively in the nodes and buckets of the tree regions to match against MBBs of the tree. Algorithm 1 shows how hashing operation is performed for the BGP in the original QTree. Beside the BGP conversion, we also apply hashing for the context to create numerical values by using Context Taxonomy. Thus, this structure searches a region representing the *BGP+Context* instead of only BGPs. Such that, we provide pruning during search with the context information.

Moreover, our extended structure, is able to prune the nodes during recursive search by performing a quality check for the sources in the inner nodes. For the given quality threshold q_{th} from the user, quality range $q_{node} =< q_{lower}, q_{upper} >$ of the node is checked to see whether the node has a higher value than the threshold. This operation prevents to visit the unnecessary nodes for the deeper visits of the branches. In this manner, we aim at accelerating the search and get the most qualified sources.

Having posed a four dimensional query space on the extended QTree, a search is elaborated to benefit from the hierarchical structure of the QTree. Such that, original QTree structure uses FindRelevantBuckets function in Algorithm 3 to evaluate query space in an iterative manner in the QTree buckets. However, extended QTree structure discovers the buckets which might contribute to the sources exploiting recursive search and quality pruning. Thus, we point out that while original QTree compares query space in the bucket level, extended QTree benefits from the tree structure to avoid irrelevant searches.

The result of the query is a set of buckets which are representatives of MBBs in the leaf nodes. Since the query has only one BGP, then the returned results are a set of buckets which are independent of each other. Following, the number of the triples belonging to each source are summed up to discover the cardinality of a source to the given query. In our data structure, this aspect is also expanded by counting the triples attached to their context for each source to be able to discover relevant sources context-dependently for the query. Since a source might have more than one context information, only the triples similar with query context will be counted by the structure to measure relevance. Thus, the output of the source selection algorithm is set of buckets with intersection information about the overlap, *(source, context)* pair with quality and associated cardinality information.

4.4.3.2 Join Search Operation

Join operation is performed when there are more than one BGPs in the query. Evaluation of the query space for join operation was presented before for original QTree source selection (Algorithm 2) by joining the buckets in the space. Query evaluation for the extended structure is changed by adopting it to the new structure. Assuming a query is given with context information, it is evaluated in the Extended QTree structure by searching the relevant regions for each (BGP+Context) in the tree separately and then combined by finding the common coordinates. Before, (3 · (i + 1))-dimensional regions resulting from the i-th join were stored in a join space. Since this new structure has 4 dimensions, the join space is also created with (4 · (i + 1)) for the join number i. Fourth dimension of the tree which comes as an extension to the QTree structure permits to limit the relevant sources w.r.t. the context information.

Query evaluation is performed in QTree as it is represented in the original QTree structure with some differences. (BGP+Context) of the query is converted into the query space which represent 4 dimensional query space. Later, recursive search is performed in the Extended QTree to find the overlapping buckets with this first query space. Finding left and right buckets (Algorithm 3) are replaced with Algorithm 8 to provide quality dependent, recursive processing. Besides, the source cardinality is computed for each source depending on their context information in the bucket by $c_{sc} = c_{sc} * size_O / size_B$. If a source has more than one context information, it is inserted to the list for each context information. The number of buckets which contributes to the result is added to the *temporaryResult* variable for right and left bucket as well.

Join space is further established by the computeJoin method (Algorithm 4). This method is employed for joining the left and right side of the results. The left side of the join space represents the (4 · (i+1)) dimensional regions resulting from the i-th join. Left results are kept in the J_{in} join space for further joins. The actual overlap of all intersection spaces for left hand side is determined by only restricting the join dimension for each bucket coming from the right side of the result. This new space is used to find the overlaps for each buckets coming from the left-hand side of the operation (Line 6). Depending on this new overlapping information new join dimensions are set for the join space. Later boundaries of the current intersection is used to limit the buckets of the right hand side. As a consequence we have a Join Space with increased dimensions and intersection information. The $\otimes$ operator represents the join of left side and right side buckets with increasing number of dimensions according to new structure. The cardinality result of this join is represented with $c_{O_R \otimes O_L}$ and it is computed by using the cardinality of left and right side of the join operation (Line 8). This new computation is given in Equation 4.4.

$$c_{sc} = \frac{c_{scL} \cdot \frac{size(O_L)}{size(L)} . c_R \cdot \frac{size(O_R)}{size(R)} + c_L \cdot \frac{size(O_L)}{size(L)} . c_{scR} \cdot \frac{size(O_R)}{size(R)}}{max(L[left].hi - L[left].low, R[right].hi - R[right].low)} \tag{4.4}$$

On the other hand, according to the intersection information the source contribution to the join is computed and returned by the join space (Line 9). c_{sc} represents the total cardinality result of join operation per specific source w.r.t. the context information. For each source, the cardinality is computed for each side of the join operation (right and left side buckets) (Equation 4.4). The join cardinality of the sources are computed by taking the cross product of the sources in the joining buckets. First, left side contribution is computed by multiplying the number of triples per context-dependent source (c_{scL}) with related bucket and all the triples from the right hand side of the join with related bucket. This process is adopted to find the cross product of sources from both sides of the results. The same process is performed for the right side of the join(c_{scR})Later, the sum of the computation is divided by the maximum range of join coordinate among left and right hand sides to provide uniform distribution. This join space result with overlapping space and source information is returned to the *evaluateQuery* method in Algorithm 9 for the further ranking evaluation.

4.4.4 Extended 4D QTree Ranking

In this structure, the idea is ranking the sources w.r.t. the number of triples related with each context information associated with the number of sources. During join operation we computed the intersection information and then put this information to the new intersected buckets for further computation of the ranking of the sources. However, we need to compute the ranking information as a sum of all the join buckets.

In this structure, we applied a straightforward approach, such that, every *(source, context)* pair is introduced as a new source, thus, we adopted the same ranking approach that we proposed in the three dimensional extended QTree. Ranking of the sources is performed for each pair by counting related cardinality. In this approach context similarity is found for each pair and quality score is retrieved from the auxiliary structures. Moreover, normalized context distance, quality score and source contribution are computed for each source. As a result, a relevance score computed for each *(source,context)* pair from this calculation which is used to rank them. One drawback of this structure comparing to previous approaches is high number of the sources which increases by each context related source pair. Since a source is associated with more than one context, it is possible to obtain a source with different context representations for a given query.

Chapter 5

Experimental Results

In this chapter, we experimentally compare the index data structures for source selection, introduced in Chapter 4. In particular, we first discuss the experimental setting, with a special reference to the considered queries. Then, the original QTree, the three dimensional extended QTree, and the four dimensional extended QTree will be compared with respect to performance issues and accuracy of the contexts associated with the returned sources. The role of quality-based pruning and of different types of rankings will also be investigated.

5.1 Experimental Setting

In the following, we shortly describe the features of the machine, dataset, index data structures, and queries used in the experiments we have performed.

Environment: The experiments were performed on a machine with Intel Core i3-2350M 2.30GHz as CPU and 6 GB of RAM size, running 64-bit Windows 7 Professional as operating system. Applications were developed in Java HotSpot(TM) 64-Bit Server VM with version of Java SE 1.8.

Dataset: A real-world set of RDF data sources has been collected from the Web via LDSlicer,[1] as described in Section 2.5. The set has been gathered through a breadth-first crawl starting from a specific concept node in DBpedia (`http://dbpedia.org/resource/Category:Tourist_attractions_in_Europe`) by using content negotiation technique to retrieve the content of RDF sources. As a result, a dataset consisting of 1.086.874 RDF statements has been retrieved

[1] urlhttps://github.com/keme686/LDSlicer

from 13.466 RDF sources. We then associated a set of contexts (i.e., SKOS concepts) with each triple, as discussed in Chapter 2.

Index Data Structures: Experiments have been performed by considering the index data structures presented in Chapter 4: the original QTree data structure (denoted by QTree in the following), the QTree extended with context information in the leaves (Ext QTree), and the QTree generated over four dimensional points (4D QTree). QTree and Ext QTree have been generated by considering 25.000 as maximum number of buckets, 20 as maximum fanout, and interval $[0-2000]$ for hash values associated with each triple component (subject, predicate, object, and context). These are the same values proposed in [85]. For what concerns 4D QTree, we increased the maximum number of buckets from 25.000 to 50.000, in order to provide a better data partitioning, since the number of tuples to be indexed in 4D QTree is much higher than the number of triples to be indexed in QTree and Ext Tree.

Tree	# of generated buckets/maximum # of buckets	Index Size	Hash interval
QTree	8.092/25.000	31,3 MB	$[0-2000]$
Ext QTree	8.092/25.000	64,8 MB	$[0-2000]$
4D QTree	50.000/50.000	188.4 MB	$[0-2000]$

Table 5.1: QTree details

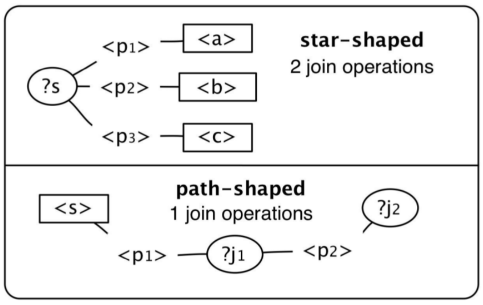
Table 5.1 shows some parameters describing the used index data structures. As we can see, while 25.000 buckets are enough for QTree and Ext QTree, since less buckets than the maximum available are created in the tree, 50.000 seems to be a too strict limit for 4D QTree. Indeed, in this case, all 50.000 buckets are created. A higher number of buckets would probably guarantee a better data distribution. Due to time constraints, we leave the investigation of this case to future work.

Queries: We developed a query generator for creating random queries, guaranteeing at least one result on the considered dataset.

Similarly to what has been done in [85], two different types of SPARQL queries have been considered, depicted in Figure 5.1. The first class of queries consists of *star-shaped queries* with one variable at the subject position. The second type of queries are *path-shaped queries* with join variables at subject and object positions. The query classes of choice are generally understood to be representative for real-world use cases and are also used to evaluate other RDF query systems. The star-shaped queries were generated by randomly picking a subject URI from the input data and arbitrarily selecting distinct outgoing links. Then, we substituted the subject in each BGP with a variable. Path queries were generated using a random walk approach. We randomly chose a subject URI and performed a random walk of pre-defined depth. The result of such a random walk was transformed into a path-shaped join by replacing the connecting nodes with variables.

We considered queries with a variable number of joins, ranging from 0 to 7 (up to 2 joins were considered in [85]) and we generated 10 queries for each number of join, leading to the generation of 70 different queries for each query type.

Even if we created two distinct workload, one for each query type, due to time constraints, we performed experiments only for star-shaped queries. We leave the execution of experiments related to path-shaped queries to future work.



[82]

Figure 5.1: Abstract representation of the considered query types

Each star-shaped query has then been associated with a context, randomly selected among the contexts associated with resources satisfying the query. Matching between resources and contexts has been done by executing specific SPARQL queries relying on the Apache Jena framework, [2] an open source Semantic Web triplestore for Java.

Query execution: Each generated query was executed 10 times, on the reference tree (QTree for queries without context, Ext QTree and 4D Qtree for queries with an associated context). For each query execution, corresponding to an index search, values for several parameters related to selectivity and performance, as well as to context result accuracy, have been collected and average values have been computed for each number of join(from 0 to 7). Executions on Ext QTree and 4D QTree were performed by considering two distinct values as quality thresholds: 0 and 0.50.

The following parameters related to selectivity and performance were computed for each index search, on the reference trees:

- Execution time, excluding ranking time.
- Number of accessed buckets, obtained as the sum of the number of the buckets accessed for each BGP belonging to the query.

[2]https://jena.apache.org/

- Number of buckets returned by the overall query execution (after join computation).
- Number of sources returned as results.

For what concerns result accuracy, only for Ext QTree and 4D QTree executions, we computed aggregate distances between the contexts associated with the returned sources and the context provided with the query. More precisely:

4D QTree: since each returned source is associated with a single context, we computed the distance between each returned context and the context associated with the query; we then computed the average distance among all them.

Ext QTree: since, as discussed in Section 4.3, each returned source is associated with a set of contexts, we first computed an aggregate distance to be associated with each returned source, we then computed the average distance among all them. Three different aggregate functions have been considered: AVG, MIN, and MAX.

We additionally considered ranking values associated with the returned sources, computed by considering various ranking functions.

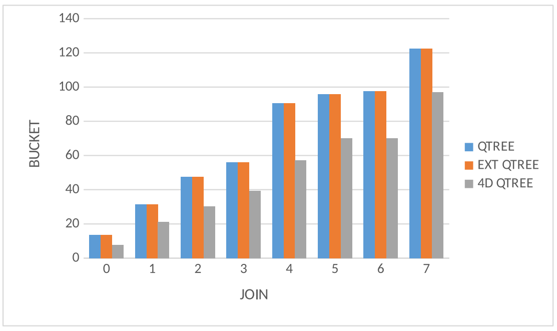
5.2 Experiment Results

In the following, performed experiments and related results will be presented, aggregated into four groups: *i)* results related to selectivity and performance issues; *ii)* results related to context result accuracy; *iii)* results related to quality pruning; *iv)* results related to ranking.

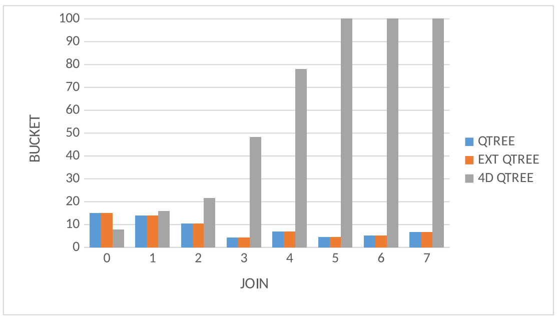
5.2.1 Selectivity and Performance

The aim of the first group of experiments is to investigate: *i)* the ability of ExtQTree and 4D QTree, and of context information in general, to reduce the number of false positives, taking the original QTree as baseline; *ii)* the execution time of the three considered index data structures.

Selectivity: Figure 5.2.a shows the average number of buckets returned by the considered queries, with a variable number of join from 0 to 7. We can observe that, even if the number of buckets contained in 4D QTree is much higher than those contained in QTree and Ext QTree, thanks to context information, which is indexed together with triples, the 4D QTree is able to filter out more buckets with respect to 3D approaches. From the same figure we can see that the



(a) Average number of returned buckets



(b) Average number of join buckets

Figure 5.2: Selectivity estimation

same number of buckets is always returned by QTree and Ext QTree, since the two data structures differ only for information contained at the bucket level.

Figure 5.2.b shows the number of buckets obtained as result by the join execution. As we can see, 4D QTree performs much worst than QTree and Ext QTree. This is due to the fact that, as discussed in Subsection 5.1, setting the maximum number of buckets to 50.000 does not guarantee a good data distribution among the buckets. In particular, many regions associated with buckets intersect and are returned as result. This is why the average number of join buckets is so high.

Figure 5.3 shows the average number of returned sources from the resulting buckets, after removing duplicates, as discussed in Chapter 4. We can observe that, even if the number of returned buckets is higher with 4D QTree, the number of returned sources is much lower when compared with QTree and Ext QTree. This is due to the usage of the context information associated with the query during index search in 4D Qtree (see Chapter 4). We remark that the number of returned sources can further be reduced by taking ranking into account (see Subsection 5.1).

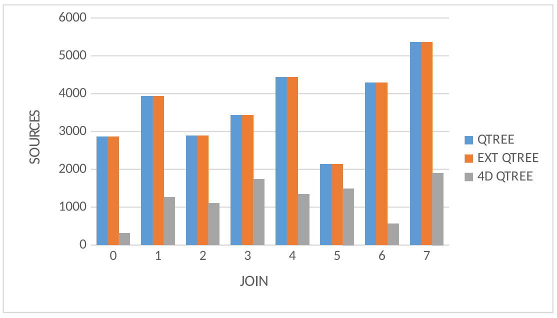


Figure 5.3: Selectivity estimation: average number of returned sources

Performance: Figure 5.4 shows the average execution time, in milliseconds, for the three tree index data structures. As expected, the query execution time for QTree and Ext QTree is the same.

On the other hand, query execution time for 4D QTree is often higher, depending on the number of joins to be performed and the number of intersections among buckets returned by single BGP executions (that, as discussed before, is quite high).

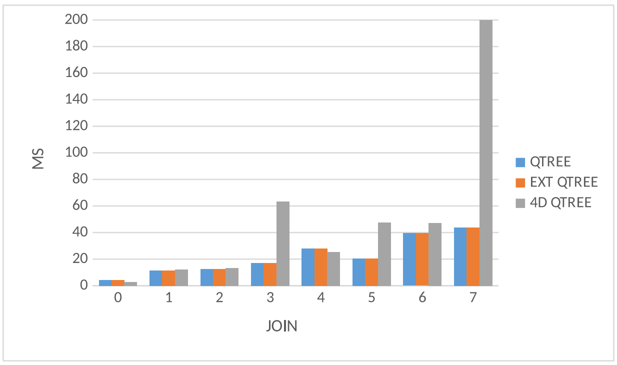


Figure 5.4: Performance evaluation: average execution time

5.2.2 Context Accuracy

The aim of the second group of experiments is to investigate the accuracy of the contexts associated with the returned sources. Accuracy is computed as an aggregate distance between the contexts returned as results by the index search and the context associated with the query, as pointed out in Subsection 5.1.

Figures 5.5, 5.6.a, and 5.6.b show various average distances computed between contexts associated with the returned sources in Ext QTree and 4D Qtree and the context provided with the query, while varying the number of joins. Distances coincide for what concerns 4D QTree; on the other hand, for what concerns Ext QTree, they are computed by considering the average, the maximum, and the minimum context distance for each source, respectively.

As we can see, the average context distance obtained with 4D QTree is always lower than the average context distance obtained with Ext QTree, when using AVG as aggregation function (Figure 5.5). Combined with Figure 5.3, this means that 4D QTree always retrieves less sources with a more relevant context, in average, than Ext QTree. Of course, a similar results also holds when we consider MAX as aggregate function for computing distances in Ext QTree (Figure 5.6.a).

On the other hand, if we consider MIN as aggregate function for computing distances in Ext QTree (Figure 5.6.b), Ext QTree performs better than 4D QTree. This is due to the fact that each data source returned by an index search over Ext QTree is associated with all its related contexts, including the context leading to the minimum distance with respect to the query context. On the other hand, with 4D QTree, the context associated with each returned data source might not be that corresponding to the minimum distance with respect to the query context.

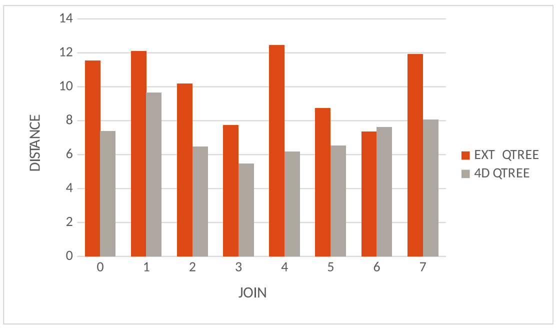
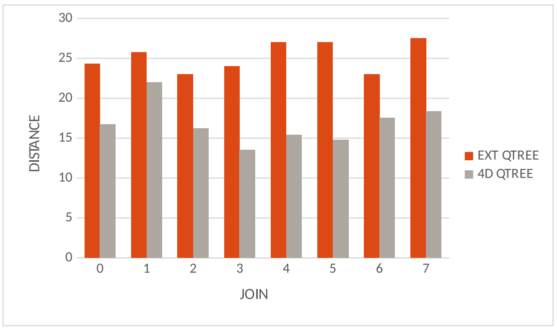
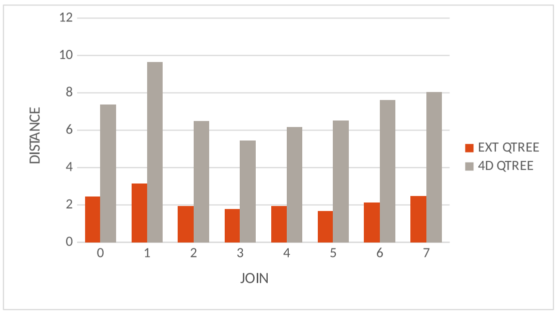


Figure 5.5: Context result accuracy: average distance



(a) Average Maximum Distance



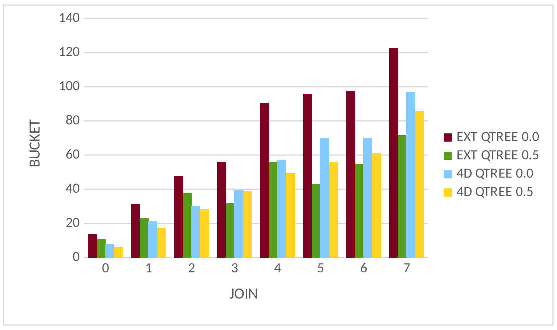
(b) Average Minimum Distance

Figure 5.6: Context result accuracy, with respect to MAX/MIN distance aggregate function

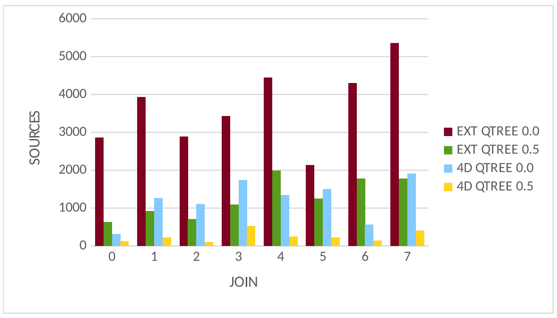
5.2.3 Impact of Quality Pruning

The aim of the third group of experiments is to investigate the impact of quality checks during index searches, with the aim of pruning subtrees that index sources whose associated quality value is lower than the specified threshold (see Chapter 4). To this aim, we consider 0.5 as quality threshold and we compare the obtained results with those obtained without quality checking (corresponding to a quality threshold equal to 0). Since quality checks impact the set of buckets returned by index searches, selectivity will be taken into account for the comparison.

From Figure 5.7.a we can see that the total number of accessed buckets decreases while increasing the threshold, since quality checks prune subtrees that index sources whose associated quality value is lower than the specified threshold, A similar consideration holds for the returned sources (see Figure 5.7.b), since the number of buckets returned by BGP executions has an impact on the number of buckets returned by the join execution and, as a consequence, on the total number of returned sources.



(a) Average number of returned buckets



(b) Average number of returned sources

Figure 5.7: Quality pruning: performance estimation

5.2.4 Impact of Ranking

The aim of the fourth group of experiments is to investigate the impact of ranking on the generated results. As discussed in Section 4.3.4, a ranking function can be defined as a linear combination of three elements: *i)* the number of triples associated with each retrieved source; *ii)* the distance of the contexts associated with each retrieved source and the query context; *iii)* deferenceability metric associated with each retrieved source.

We performed experiments by considering three different ranking functions, associating different weights α and β to parameters *i)* and *ii)* described above and setting 0 for parameter *iii)*, for the sake of simplicity. More precisely, we considered the following values:

- *Only number of triples*, corresponding to $\alpha = 1$ and $\beta = 0$ (ranking function $f_{1,0}$).
- *Equal weights*, corresponding to $\alpha = 0.5$ and $\beta = 0.5$ (ranking function $f_{0.5,0.5}$).
- *Only context distance*, corresponding to $\alpha = 0$ and $\beta = 1$ (ranking function $f_{0,1}$). In this case, aggregate context distances in Ext QTree are computed using the MIN aggregate function.

We then investigate dhow ranking values vary while varying k and considering only the k retrieved sources with the highest ranking value. We consider $k = 10, 50, 100, 200$ and we compared the obtained ranked results with those obtained by considering all the retrieved sources.

Figures 5.8, 5.9, and 5.10 show average ranking values computed for all data sources and for varying k values, computed with $f_{1,0}$, $f_{0.5,0.5}$, and $f_{1,0}$, respectively. As we can see from the charts, ranking values decreases for both Ext QTree and 4D QTree while increasing the number of the considered sources. This behaviour holds for all the considered ranking functions, showing that the selection of the first k ranked sources is a good option for reducing the total number of sources to be accessed for query processing while guaranteeing a good result quality.

From the same charts, we can observe that, by considering at least one join, 4D QTree average ranking values computed for all the returned sources is lower than the corresponding Ext QTree ranking values. This is coherent with previously presented results (compare, for example, Figures 5.6.b and 5.10, recalling that, for $f_{0,1}$, ranking values are inversely proportional to context distances). On the other hand, when we consider the first k ranked sources, for $k = 10, 50, 100, 200$, $4DQTree$ ranking values are higher than corresponding Ext QTree ones. Based on the computation of ranking values, described in Sections 4.3.4 and 4.4.4, this means that 4D QTree is able to select, among all the returned ones, "better" sources with respect to Ext QTree. Such sources can be easily identified through the proposed ranking approach.

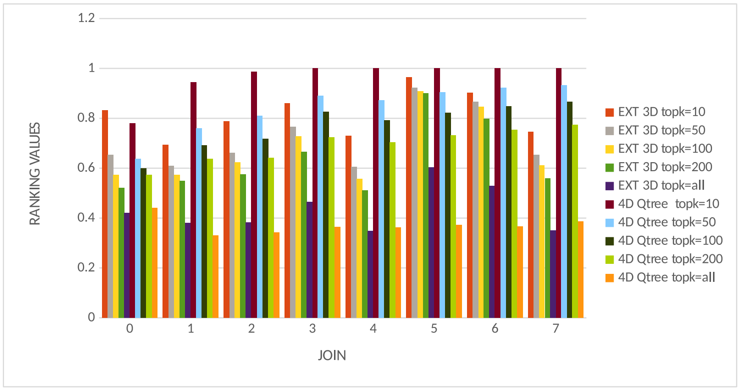


Figure 5.8: Ranking values for $f_{1,0}$

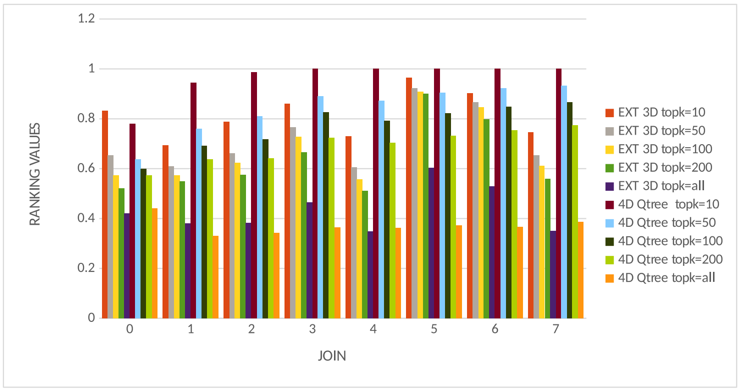


Figure 5.9: Ranking values for $f_{0.5,0.5}$

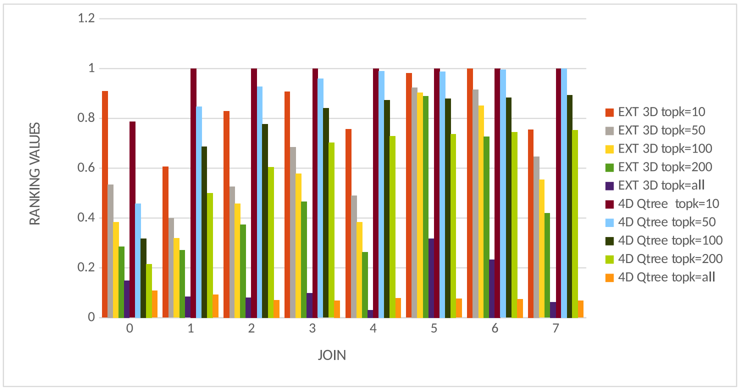


Figure 5.10: Ranking values for $f_{0,1}$

Chapter 6

Conclusions

The thesis contributes an approach for live query processing on Linked Data that takes into account context and quality during source selection. To the best of our knowledge no other approaches exist in this direction.

The main contribution of the thesis is the definition of two different indexing structures that allows to exploit context and quality metadata (associated with data and sources during a pre-processing phase) during the source selection phase, so to take them into account in an efficient and effective way for selecting the most relevant and highest quality sources. A source selection framework and the specific indexing and source selection techniques have been implemented, and the techniques are experimentally evaluated on a real dataset demonstrating that employing context and quality measures improves source selection in Linked Data live query processing.

Once the relevant dataset (consisting of 13466 sources) and the corresponding context taxonomy (grounded in SKOS [67] concepts) had been extracted, we firstly used the Luzzu [33] quality assessment framework to evaluate the *quality* of Linked Data sources. Quality profiles are generated by applying an automatic process. We assessed the quality of the sources according to the *dereferencability* metric and store the produced graphs for the following source selection phase. This metric had been selected because accessibility of the sources plays an important role for further query processing.

During source indexing, quality scores retrieved from the quality graphs are associated with the corresponding sources so that they can inform the source selection phase. Such scores are indeed the basis on which our data summary can ensure to select highest quality sources for each given query. We showed that we are able to select the highest quality sources by pruning the unnecessary visits in the nodes of the tree and filtering out the low quality sources. Since we measured the accessibility of the sources, in this case, we have pruned the sources which are less likely to be accessible than others. The approach can however be adapted to other quality metrics and indicators, demonstrating the potential of data quality assessment for source selection.

As a further enhancement to source selection, we propose to take advantage of *context* to select the most relevant sources. Thus, context information (represented in a context taxonomy) are associated with triples and sources. We proposed to measure context similarity of the sources to the given query context by computing distances among them. In the experiments, we showed that the new systems are able to discover the sources with minimum distances to the query context. Besides, we also demonstrated that using various ranking functions one can further improve the accuracy of the results but s/he might decide to trade-off by choosing among different parameters, such as, quality, context similarity ,and cardinality.

In evaluating the framework, we have seen that efficiency of the results is not only dependent on the indexing techniques but also it is highly dependent on the external factors such as amount of relevant data portion, query types, approximation techniques, etc. Another issue that we coped with was the lack of benchmark datasets and queries for the evaluation task at hand. We had to overcome this problem by extracting data from several endpoints and generating our queries upon that dataset.

Future Work. There are several interesting directions for future extensions to the work developed in the thesis. In the thesis, we relied on a context notion and modeling related to a domain taxonomy providing a high-level, abstract view of the information content. However, the approach can be adopted for different notions of contexts as well. For instance, context can be automatically associated with users and queries by integrating the framework with a pervasive system. In the scope of this work we used the domain context model centered around the *what*, *where*, *when* elements for the user as well. But it is also possible to take advantage of the user contexts which might employ the search history or preferences of the user as context information. These other elements might increase the relevance of the sources to the user query since they are going to increase the user information available in the system.

One of them is the possibility of taking *user feedbacks* into account for assessing source quality and relevance with respect to a context. This is graphically represented in the diagram in Figure 1.1. Relevance is highly dependent on the user requirements, such that, it seems reasonable to incorporate in the framework a component that gathers relevance of the returned results and add this new information to the quality profile of the source. This would also allow to present subjective quality measures besides the objective ones which are measured looking at the data and source itself. Another option related to quality would be the implementation of the dynamical updates of the quality profiles according to the user feedbacks on the quality dimensions.

A final direction is concerned with the query rewriting phase. For the one who would like to follow a more traditional approach for Sparql query processing, they may consider associating context information as another BGP to the query and exploiting the contexts in that way. Further experiments are needed to assess the pros and cons of this query rewriting alternative with respect to the indexing framework with ad hoc treatment of context we proposed.

Appendix A

Following tables provide the data quality dimensions, metrics provided by each dimension and metric function by the descriptions.

A.1 Accessibility

Quality dimension	**Metric Name**	**Assessment Metric**	**Description**
Availability	Dereferencability	metric = der. triple number/ total triple number	metric calculates the number of valid redirects (303) or hashed links according to LOD Principles
	Dereferenceability Forward Links	metric = the ratio between the number of subjects that are "forward-links" a.k.a "outlinks" (are part of the resource's URI) and the total number of subjects in the resource	if a subject resource has a description, then it is a valid forward link because from that URI its description can be accessed directly
	Misreported Content Type	metric = resources with correct reported type / resources with (misreported+correct) type	check if RDF/XML content is returned with a reported type other than application/rdf+xml
	SPARQL Accessibility	metric returns true or false	check if a SPARQL endpoint (matching void:sparqlEndpoint) is available and returns a result
	RDF Accessibility	metric returns true or false	check if all data dumps (void:dataDump) exist, are reachable and parsable
Interlinking	Dereference Back Links	metric = total valid back links / size of uri (?)	metric checks if an object resource is adequatly described as a Semantic Resource, thus a valid backlink
	Link External Data Provider	metric = the number of valid external links in a dataset	identify the total number of external linked used in the dataset
	Human Readable Licens	metric = Valid Licences / human Licence Per Dataset	In contrast with the Machine-readable Indication of a License metric, this one looks for objects containing literal values and analyzes the text searching for key, licensing related terms.
	Machine Readable License	metric = valid licences/ licensed datasets	A dataset should have its licence attached to a void:Dataset description
	Correct URI Usage	metric = no of hash URI / no of triple	the metric will return 1.0 (or true) if slash uri's are used when a dataset has a large amount of data or hash uri's are used if otherwise
Security	Digital Signature Usage	metric = no of documents without endorsement / no of total documents	

Table A.1: Quality dimensions and quality metrics for accesibility category

A.2 Contextual

Quality dimension	Metric Name	Assessment Metric	Description
Freshness	Currency Of Dataset	metric = (observation date - update date) or metric = (observation date - creation date)	freshness (currency) of the data, i.e. how fresh is the data when delivered to the user
	Freshness	maxFreshness = max(0, (1 - (currency / volatility)))	if the data in a dataset is fresh wrt currency and volatility.
	Timeliness Of Resource	metric = avg[(observation time)-(expiration/validity time)]	
Understandability	Human Readable Labelling	metric = 1 - no of human labels/no of entities	measures the percentage of entities having an rdfs:label or rdfs:comment
	Presence Of URI RegEx	metric returns true or false w.r.t. the condition	checks if the URI reqular expression exists
	Vocabulary Usage Indication	metric = total namespaces indicated/total different namespaces used;	

Table A.2: Quality dimensions and quality metrics for contextual category

A.3 Intrinsic

Quality dimension	Metric Name	Assessment Metric	Description
Completeness [9]	Data set metadata existence	Boolean true or false	Check if there is a valid metadata file by issuing a package show request
	Resources format field existence	metric = resources with meta and void value/ all resource numbers	Check the resources format field for meta/void value
	Topic tag existence	Boolean true or false	Check if the dataset has a topic tag
	Format and mimetype fields existence	metric = resources with format and mimetype value/ all resource numbers	Check the format and mimetype fields for resources

Table A.3: Quality dimensions and quality metrics for intrinsic category

Quality dimension	Metric Name	Assessment Metric	Description
Conciseness	Extensional Conciseness	metric = no of unique subjects / total no of subjects	provides a measure of the redundancy of the dataset at the data level
Consistency	Advanced Entities As Members Of Disjoint Classes	metricValue = 1 - (no of entities as members of disjoint classes /types Of resource)	metric checks both explicit resource type and their implicit (inferred) subclasses for disjointness
	Misplaced Classes Or Properties	metric = 1.0 - ((misplaced (properties+classes) count) / (total (properties+classes) count)	checks if the assessed dataset has a defined classed placed in the triple's predicate and defined property in the object position
	Misused Owl Datatype Or Object Properties	metric = 1.0 - (misused (Datatype+Object) properties / valid predicates)	detects properties that are defined as a owl:datatype property but is used as object property and properties defined as a owl:object property and used as datatype property
	Ontology Hijacking	metric = 1.0 - (total hijacks / total possible hijacks)	detects the redefinition by analyzing defined classes or properties in data set and looks of same definition in its respective vocabulary
	Simple Entities As Members Of Disjoint Classes	metric = 1 - (no of entities as members of disjoint classes / no of types of resource	counts number of entities that are members of disjoint classes
	Usage Of Deprecated Classes Or Properties	= 1 - (deprecated (Types + Properties) / total (Types + Properties))	metric checks if a dataset makes use of Deprecated Classes or Properties. A high usage of such classes or properties will give a low value (closer to 0), whilst a low usage of such classes will give a high value (closer to 1)
	Usage Of Incorrect Domain Or Range Datatypes	metric = 1 - (incorrect Domain + incorrect Range + undereferenceable Predicates + unknown Domain And Range) / total Predicates * 2)	metric tests if a property's domain and range are of the same type as declared in the corresponding schema.
Syntactic Accuracy	Valid Inverse Functional Properties (IFP) Usage	metric = 1.0 -(total Violated IFPs/ total IFPs)	metric checks if IFP is used correctly, i.e. if we have S P O and P is set to be an owl:InverseFunctionalProperty, then S is the one and only resource connected to O. If there is a triple S1 P O, then the IFP is not used correctly and thus since S1 will be "reasoned" to be the same as S.
	Compatible Datatype	double metricValue = number of correct literals / (number of incorrect literals + number of correct literals)	metric checks the compatability of the literal datatype against the lexical form of the said literal. This metric only catches literals with a datatype whilst untyped literals are not checked in this metric as their lexical form cannot be validated
	Correct Language Tag	metric = total correct strings / total valid lang strings	
	Untyped Literals	1.0 -(number untyped literals / number (typed + untyped) literals)	checks for the number of untyped literals. Untyped literals, albeit the possibility of being correct, are undesirable as agents should be able to convert a Literal value to the actual data type

Table A.4: Quality dimensions and quality metrics for intrinsic category

A.4 Representation

Quality dimension	Metric Name	Assessment Metric	Description
Interoperability	Reuse Existing Terms	metric = (overlap classes + overlap properties) / (total classes + total properties)	metric assesses if dataset reuse relevant terms for a particular domain. A dataset is deemed conformant to this metric if it exhibits a higher overlap within the recommended vocabularies.
	Reuse Existing Vocabularies	metric = no of used namespace from suggested list / total no of suggested vocabularies	metric calculates the number of unique vocabularies used in a dataset, but suggested to be useful in the dataset's domain. metric checks if the term used is defined or not before adding to the seen-suggested set.
Interpretability	Blank Node Usage Number	metric = number of blank nodes/ (number of data level constraints + number of blank nodes)	metric calculates the number of blank nodes found in the subject or the object of a triple.
	Undefined Classes And Properties	metric = undefined (classes + properties)/ total (classes + properties)	metric measures the number of undefined classes and properties (without any formal definition) used by a data publisher in the assessed dataset.
	Basic Provenance Metric	metric = valid provenance / dataset size (?)	metric measures if a dataset have the most basic provenance information, that is, information about the creator or publisher of the database
	ExtendedProvenanceMetric	metric = sum(value of entities) / number of entities	metric measures how much datasets conform to the W3C standard PROV-O ontology. (An agent and activities are both given a weighting of 0.5. The 0.5 weighting for activities are split amongst all activities. Agent and Datasource are equally given a weighting of 0.5 per activity)
Representational Conciseness	No Prolix RDF	metric = total number of prolix (RCC) triples against the / total number of triples	metric detects the use of standard RDF Prolix Features. These features are Collections (rdf:Alt, rdf:Bag, rdf:List, rdf:Seq), Containers and Reification (rdf:Statement).
	Short URIs	metric = total number of short uris/ number of local URIs of a dataset	detects long URIs or those that contains query parameters, thereby providing a measure of how compactly is information represented in the dataset. The W3C best practices for URIs say that a URI (including scheme) should be at max 80 characters.
Versatility	Different Serialisation Formats	metric = datasets with more than one feature / total number of datasets	if in a dataset has 1 or more triples descibing the different serialisation formats that a dataset is available in, using the void:feature. metric returns 1 if the data published is represented in 2 or more formats
	Multiple Language Usage	metric = average number of languages used throughout the dataset (per resource)/ multiple language size	check if data (in this case literals) is available in different languages, i.e. a dataset supports multilinguality. In this metric, we will check all literals having a language tag. Those without a language tag will be ignored.

Table A.5: Quality dimensions and quality metrics for representational category

Bibliography

[1] Rdf dataset description. `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-dataset`. Accessed: 16-01-2018. [Cited on page 13.]

[2] Rdf dataset description. `https://www.w3.org/TR/rdf11-mt/#rdf-datasets`. Accessed: 16-01-2018. [Cited on page 13.]

[3] Rdf source description. `https://www.w3.org/TR/rdf11-concepts/#dfn-rdf-source`. Accessed: 16-01-2018. [Cited on page 13.]

[4] K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt. Improving data access in p2p systems. *IEEE Internet Computing*, 6(1):58–67, 2002. [Cited on page 54.]

[5] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *International Symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer, 1999. [Cited on page 6.]

[6] G. D. Abowd and E. D. Mynatt. Charting Past, Present, and Future Research in Ubiquitous Computing. *ACM Trans. Comput.-Hum. Interact.*, 7(1):29–58, 2000. [Cited on page 7.]

[7] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 191–226. Springer, 2015. [Cited on page 8.]

[8] A. Assaf, A. Senart, and R. Troncy. Roomba: automatic validation, correction and generation of dataset metadata. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 159–162. International World Wide Web Conferences Steering Committee, 2015. [Cited on page 32.]

[9] A. Assaf, A. Senart, and R. Troncy. An objective assessment framework & tool for linked data quality: Enriching dataset profiles with quality indicators. *International Journal on Semantic Web and Information Systems (IJSWIS), Special Issue on Dataset Profiling and Federated Search for Linked Data, ISSN: 1552-6283*, 05 2016. [Cited on pages 32 and 84.]

[10] S. Auer. Creating knowledge out of interlinked data: making the web a data washing machine. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 4. ACM, 2011. [Cited on page 26.]

[11] S. Auer, L. Bühmann, C. Dirschl, O. Erling, M. Hausenblas, R. Isele, J. Lehmann, M. Martin, P. N. Mendes, B. Van Nuffelen, et al. Managing the life-cycle of linked data with the lod2 stack. In *International semantic Web conference*, pages 1–16. Springer, 2012. [Cited on page 26.]

[12] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern Information Retrieval*, volume 463. ACM Press, New York, 1999. [Cited on page 43.]

[13] J. Bao, J. Tao, D. L. McGuinness, and P. Smart. Context representation for the semantic web. 2010. [Cited on page 8.]

[14] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR)*, 41(3):16, 2009. [Cited on page 33.]

[15] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Data-Centric Systems and Applications. Springer, 2006. [Cited on page 32.]

[16] T. Berners-Lee. Linked data, 2006, 2006. [Cited on page 1.]

[17] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001. [Cited on page 1.]

[18] L. Bertossi, F. Rizzolo, and L. Jiang. Data quality is context dependent. In *International Workshop on Business Intelligence for the Real-Time Enterprise*, pages 52–67. Springer, 2010. [Cited on pages 2 and 35.]

[19] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010. [Cited on page 7.]

[20] C. Bizer. *Quality Driven Information Filtering: In the Context of Web Based Information Systems*. VDM Publishing, 2007. [Cited on pages 32 and 33.]

[21] C. Bizer and R. Cyganiak. Quality-driven Information Filtering using the WIQA Policy Framework. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):1–10, 2009. [Cited on pages 31, 33, and 34.]

[22] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227, 2009. [Cited on page 1.]

[23] C. Bizer and R. Oldakowski. Using Context-and Content-based Trust Policies on the Semantic Web. In *WWW*, pages 228–229. ACM, 2004. [Cited on page 33.]

[24] C. Böhm, G. Kasneci, and F. Naumann. Latent topics in graph-structured data. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2663–2666. ACM, 2012. [Cited on page 29.]

[25] C. Böhm, F. Naumann, Z. Abedjan, D. Fenz, T. Grütze, D. Hefenbrock, M. Pohl, and D. Sonnabend. Profiling linked open data with prolod. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 175–178. IEEE, 2010. [Cited on page 29.]

[26] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. A data-oriented survey of context models. *ACM Sigmod Record*, 36(4):19–26, 2007. [Cited on page 7.]

[27] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM, 2005. [Cited on page 8.]

[28] B. Catania, G. Guerrini, and B. Yaman. Context-dependent quality-aware source selection for live queries on linked data. In E. Pitoura, S. Maabout, G. Koutrika, A. Marian, L. Tanca, I. Manolescu, and K. Stefanidis, editors, *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 716–717. OpenProceedings.org, 2016. [Cited on page 4.]

[29] P. B. Crosby. *Quality is free: The art of making quality certain*. Signet, 1980. [Cited on page 25.]

[30] J. Debattista. *Scalable Quality Assessment of Linked Data*. PhD thesis, Universitäts-und Landesbibliothek Bonn, 2017. [Cited on pages 33, 34, 38, and 39.]

[31] J. Debattista, S. Auer, and C. Lange. Luzzu–a framework for linked data quality assessment. In *Semantic Computing (ICSC), 2016 IEEE Tenth International Conference on*, pages 124–131. IEEE, 2016. [Cited on page 37.]

[32] J. Debattista, C. Lange, and S. Auer. daq, an ontology for dataset quality information. In *LDOW*, 2014. [Cited on page 38.]

[33] J. Debattista, C. Lange, and S. Auer. Luzzu-a framework for linked data quality assessment. *arXiv preprint arXiv:1412.3750*, 2014. [Cited on pages 5, 32, 37, and 80.]

[34] R. Delbru, A. Polleres, G. Tummarello, and S. Decker. Context dependent reasoning for semantic documents in sindice. In *Proc. of 4th SSWS Workshop*, 2008. [Cited on page 8.]

[35] A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001. [Cited on pages 6 and 7.]

[36] I. Ermilov, M. Martin, J. Lehmann, and S. Auer. Linked open data statistics: Collection and exploitation. In *Knowledge Engineering and the Semantic Web*, pages 242–249. Springer, 2013. [Cited on page 30.]

[37] M. Färber, B. Ell, C. Menne, and A. Rettinger. A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web Journal*, 1:1–5, 2015. [Cited on page 20.]

[38] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005. [Cited on page 2.]

[39] A. Freitas, E. Curry, J. G. Oliveira, and S. O'Riain. Querying heterogeneous datasets on the linked data web: challenges, approaches, and trends. *IEEE Internet Computing*, 16(1):24–33, 2012. [Cited on page 43.]

[40] F. Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, 16:345–364, 1993. [Cited on page 8.]

[41] R. Guha, R. McCool, and R. Fikes. Contexts for the semantic web. In *International Semantic Web Conference*, pages 32–46. Springer, 2004. [Cited on page 8.]

[42] A. Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984. [Cited on page 45.]

[43] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data Summaries for on-demand Queries over Linked Data. In *WWW*, pages 411–420. ACM, 2010. [Cited on pages 1, 44, 45, 47, and 48.]

[44] O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *The Semantic Web: Research and Applications*, pages 154–169. Springer, 2011. [Cited on page 44.]

[45] O. Hartig. An Overview on Execution Strategies for Linked Data Queries. *Datenbank-Spektrum*, 13(2):89–99, 2013. [Cited on page 44.]

[46] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011. [Cited on page 1.]

[47] S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer. Integrating nlp using linked data. In *International semantic web conference*, pages 98–113. Springer, 2013. [Cited on page 43.]

[48] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. *LDOW*, 628, 2010. [Cited on pages 26 and 27.]

[49] A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker. An empirical survey of linked data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:14–44, 2012. [Cited on page 27.]

[50] Y. Ioannidis. The history of histograms (abridged). In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 19–30. VLDB Endowment, 2003. [Cited on page 45.]

[51] J. Joseph. Juran's quality control handbook new york, 1988. [Cited on page 25.]

[52] T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing linked data dynamics. In *Proceedings of the 10th Extended Semantic Web Conference (ESWC2013)*, pages 213–227. Springer, Mai 2013. [Cited on page 27.]

[53] S. Khatchadourian and M. P. Consens. Exploring RDF usage and interlinking in the linked open data cloud using explod. In C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, editors, *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*, volume 628 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. [Cited on page 30.]

[54] D. Kontokostas, A. Zaveri, S. Auer, and J. Lehmann. Triplecheckmate: A tool for crowdsourcing the quality assessment of linked data. In *Knowledge Engineering and the Semantic Web*, pages 265–272. Springer, 2013. [Cited on page 32.]

[55] G. Koutrika, E. Pitoura, and K. Stefanidis. Preference-based query personalization. In *Advanced Query Processing*, pages 57–81. Springer, 2013. [Cited on page 15.]

[56] R. Krummenacher and T. Strang. Ontology-based context modeling. In *Proceedings*, 2007. [Cited on page 7.]

[57] G. Ladwig and T. Tran. Linked data query processing strategies. In *International Semantic Web Conference*, pages 453–469. Springer, 2010. [Cited on page 43.]

[58] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015. [Cited on page 17.]

[59] Y. Lin, H. Wang, J. Li, and H. Gao. Data source selection for information integration in big data era. *arXiv preprint arXiv:1610.09506*, 2016. [Cited on page 44.]

[60] F. Mahdisoltani, J. Biega, and F. M. Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2013. [Cited on page 20.]

[61] E. Mäkelä. Aether–generating and viewing extended void statistical descriptions of rdf datasets. In *European Semantic Web Conference*, pages 429–433. Springer, 2014. [Cited on page 30.]

[62] A. Malaki, L. Bertossi, and F. Rizzolo. *Multidimensional contexts for data quality assessment*. Carleton University, 2012. [Cited on pages 2 and 35.]

[63] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008. [Cited on page 61.]

[64] P. N. Mendes, H. Mühleisen, and C. Bizer. Sieve: Linked Data Quality Assessment and Fusion. In D. Srivastava and I. Ari, editors, *EDBT/ICDT Workshops*, pages 116–123. ACM, 2012. [Cited on page 31.]

[65] R. Meusel, B. Spahiu, C. Bizer, and H. Paulheim. Towards automatic topical classification of lod datasets. [Cited on page 29.]

[66] N. Mihindukulasooriya and M. P. Villalon. Loupe-an online tool for inspecting datasets in the linked data cloud. [Cited on page 31.]

[67] A. Miles, B. Matthews, M. Wilson, and D. Brickley. Skos core: simple knowledge organisation for the web. In *International Conference on Dublin Core and Metadata Applications*, pages 3–10, 2005. [Cited on pages 5, 9, and 80.]

[68] I. Millard, H. Glaser, M. Salvadores, and N. Shadbolt. Consuming multiple linked data sources: Challenges and experiences. 2010. [Cited on page 43.]

[69] F. Naumann. *Quality-driven query answering for integrated information systems*, volume 2261. Springer Science & Business Media, 2002. [Cited on page 33.]

[70] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *PVLDB*, 1(1):647–659, 2008. [Cited on page 44.]

[71] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice. com: A Document-oriented Lookup Index for Open Linked Data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008. [Cited on page 44.]

[72] M. Palmonari, A. Rula, R. Porrini, A. Maurino, B. Spahiu, and V. Ferme. Abstat: Linked data summaries with abstraction and statistics. In *The Semantic Web: ESWC 2015 Satellite Events*, pages 128–132. Springer, 2015. [Cited on page 30.]

[73] L. L. Pipino, Y. W. Lee, and R. Y. Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002. [Cited on page 33.]

[74] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Prefix hash tree: An indexing data structure over distributed hash tables. In *Proceedings of the 23rd ACM symposium on principles of distributed computing*, volume 37, 2004. [Cited on page 53.]

[75] T. Rekatsinas, X. L. Dong, L. Getoor, and D. Srivastava. Finding quality in quantity: The challenge of discovering valuable sources for integration. In *CIDR*, 2015. [Cited on page 23.]

[76] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995. [Cited on page 11.]

[77] P. Resnik et al. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res.(JAIR)*, 11:95–130, 1999. [Cited on page 11.]

[78] D. Ritze, O. Lehmberg, Y. Oulabi, and C. Bizer. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 251–261, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee. [Cited on page 29.]

[79] A. Rula and A. Zaveri. Methodology for assessment of linked data quality. In *LDQ@ SEMANTICS*, 2014. [Cited on page 34.]

[80] M. Saleem. Efficient source selection for sparql endpoint query federation. 2015. [Cited on pages 42 and 44.]

[81] H. Thakkar, K. M. Endris, J. M. Gimenez-Garcia, J. Debattista, C. Lange, and S. Auer. Are linked datasets fit for open-domain question answering? a quality assessment. In *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, page 19. ACM, 2016. [Cited on page 36.]

[82] J. Umbrich. *A Hybrid Framework for Querying Linked Data Dynamically*. PhD thesis, 2012. [Cited on pages 46, 49, and 72.]

[83] J. Umbrich, A. Hogan, and A. Polleres. Improving the recall of decentralised linked data querying through implicit knowledge. *arXiv preprint arXiv:1109.0181*, 2011. [Cited on page 43.]

[84] J. Umbrich, A. Hogan, A. Polleres, and S. Decker. Improving the recall of live linked data querying through reasoning. In *International Conference on Web Reasoning and Rule Systems*, pages 188–204. Springer, 2012. [Cited on pages 13 and 16.]

[85] J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing Data Summaries for Processing Live Queries over Linked Data. *World Wide Web*, 14(5-6):495–544, 2011. [Cited on pages 1, 44, 45, 46, 48, 55, 71, and 72.]

[86] J. Umbrich, M. Karnstedt, J. X. Parreira, A. Polleres, and M. Hauswirth. Linked data and live querying for enabling support platforms for web dataspaces. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 23–28. IEEE, 2012. [Cited on page 43.]

[87] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, and P. Colpaert. Triple pattern fragments: a low-cost knowledge graph interface for the web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 37:184–206, 2016. [Cited on page 43.]

[88] R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996. [Cited on pages 33 and 34.]

[89] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. Ieee, 2004. [Cited on page 7.]

[90] G. H. Watson. Feigenbaum's enduring influence. *Quality progress*, 38(11):51, 2005. [Cited on page 25.]

[91] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2016. [Cited on pages 34 and 39.]

[92] A. Zilli. *Semantic Knowledge Management: An Ontology-Based Framework: An Ontology-Based Framework*. IGI Global, 2008. [Cited on page 7.]

[93] D. Zinn. Skyline queries in p2p systems. *Master's thesis, TU Ilmenau*, 2004. [Cited on pages 45 and 46.]