LDBC

Cooperative Project

FP7 – 317548

D1.1.1 Overview and analysis of existing benchmark frameworks

Coordinator: [I. Fundulaki]

With contributions from: [Alex Averbuch, Eva Daskalaki, Giorgos Flouris, Norbert Martinez]

1st Quality Reviewer: Peter Boncz
2nd Quality Reviewer: Thomas Neumann

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M06
Actual delivery date:	M06
Version:	1.0
Total number of pages:	45
Keywords:	Linked Open Data, RDF, Graph Databases, Data Integration, ETL

Abstract

Our goal in the LDBC project is to provide a set of open, fair, and vendor-neutral benchmarks for RDF/graph databases. As a first step towards this goal, this deliverable presents the state of the art of benchmarks for (a) core database functionalities, (b) graph databases, (c) RDF databases, and (d) RDF data integration approaches (the latter covering instance matching and ETL techniques). In addition to surveying existing works, we also discuss benchmarking principles focusing mainly on the purpose and scope of benchmarks.

Executive Summary

Objective, well-defined and good quality benchmarks are important for comparing the performance of products and systems along different dimensions and for different application needs. They provide vendors and users with an objective and reliable assessment of the behaviour of such products/systems in real-life situations and uncover useful insights related to their limitations.

With this in mind, the LDBC project was set up to provide open, fair, and vendor-neutral benchmarks for RDF/graph databases. The present deliverable is aiming at understanding and describing the current state of the art in benchmarking, along different types of benchmarks; in particular, we have been looking at benchmarks for (a) core database functionalities, (b) graph databases, (c) RDF databases, and (d) RDF data integration approaches (the latter covering both instance matching and ETL techniques).

Our objective here is not just to present the existing works in this area, but to identify shortcomings and limitations of existing systems in order to properly focus our efforts in LDBC into extending those benchmarks to deal with the challenges imposed by new applications. Thus, in addition to surveying existing works, this deliverable also discusses benchmarking principles focusing mainly on the purpose and scope of benchmarks.

Regarding core database functionalities, our study focuses on various benchmarks from the TPC family, a well-established and highly influential family of benchmarks dealing with transaction processing, querying, decision support, data maintenance and other functionalities of a relational database. However, these benchmarks are generally not suitable for other data models, as they have been specifically designed for relational databases. Despite this fact, TPC benchmarks are a nice point of reference, as they constitute a success story of mature benchmarks that can be used for inspiring similar efforts in other areas.

As explained above, benchmarks from the TPC family are not suitable for benchmarking functions of graph databases, mainly due to the different nature of the data. We examine various benchmarks from other related areas (e.g., OODB benchmarks) for their suitability, as well as benchmarks specifically designed for graph databases (e.g., HPC Scalable Graph Analysis Benchmark, Graph 500, Ciglan and others). We identify the strengths and limitations of each such benchmark, and provide a comparative analysis. Our conclusion is that the area of benchmarks for graph databases is at its early stages and that one of the inhibiting factors towards better benchmarks for graph databases is the lack of standards for graph modelling and query languages.

TPC benchmarks are not suitable for benchmarking RDF databases either, mostly because of the fact that RDF data is highly skewed and follows patterns that are not eminent in relational query workloads. We study various benchmarks for RDF databases, and we differentiate between the ones that use real datasets and queries and the synthetic ones. Our conclusion is that existing RDF benchmarks have several limitations, mainly because they are not using representative enough datasets, they employ simple queries and do not consider updates.

Data integration benchmarks have been mainly driven forward by the Ontology Alignment Evaluation Initiative (OAEI), which provides a family of data integration benchmarks and also organizes an annual campaign for evaluating existing data integration and instance matching solutions. In the context of OAEI, a multitude of data integration benchmarks (using both real and synthetic datasets) have been proposed as part of OAEI's annual evaluation of data integration and instance matching tools. Even though data integration benchmarks are generally mature and of good quality, many of them still have shortcomings, as described in this deliverable. Furthermore, we study well-known RDF link discovery systems that are using different workloads for benchmarking purposes as well as ETL benchmarks, used to evaluate tools that identify certain common types of data transformations.

Finally, this deliverable studies a set of benchmarking principles, with the aim of having a common reference point for future work on benchmarking in LDBC and elsewhere. In particular, we identify and study various types of benchmarks, such as generic benchmarks or micro-benchmarks, explain their purpose and scope, and outline the principles that have been proposed in the literature for such benchmarks.

Document Information

IST Project Number	FP7 – 317548	Acronym	LDBC
Full Title	LDBC		
Project URL	http://www.ldbc.eu/		
Document URL	`http://www.ldbc.eu:8090/display/PROJECT/Deliverable+summary`		
EU Project Officer	Carola Carstens		

Deliverable	Number	D1.1.1	Title	Overview and analysis of existing benchmark frameworks
Work Package	Number	WP1	Title	Common Benchmark Methodology

Date of Delivery	Contractual	M06	Actual	M06
Status	version 1.0		final ☒	
Nature	Report (R) ☒ Prototype (P) ☐ Demonstrator (D) ☐ Other (O) ☐			
Dissemination Level	Public (PU) ☒ Restricted to group (RE) ☐ Restricted to programme (PP) ☐ Consortium (CO) ☐			

Authors (Partner)	G. Flouris (FORTH)			
Responsible Author	Name	I. Fundulaki	E-mail	fundul@ics.forth.gr
	Partner	FORTH	Phone	+302810391725

Abstract (for dissemination)	Our goal in the LDBC project is to provide a set of open, fair, and vendor-neutral benchmarks for RDF/graph databases. As a first step towards this goal, this deliverable presents the state of the art of benchmarks for (a) core database functionalities, (b) graph databases, (c) RDF databases, and (d) RDF data integration approaches (the latter covering instance matching and ETL techniques). In addition to surveying existing works, we also discuss benchmarking principles focusing mainly on the purpose and scope of benchmarks.
Keywords	Linked Open Data, RDF, Graph Databases, Data Integration, ETL

Version Log			
Issue Date	Rev. No.	Author	Change
09/03/2013	0.1	Irini Fundulaki, Eva Daskalaki, Alex Averbuch, Norbert Martinez	First version
20/03/2013	1.0	Irini Fundulaki	Final version

Table of Contents

Executive Summary 3

Document Information 4

List of Figures 5

List of Tables 6

1 Introduction 8

2 Background 9
2.1 Resource Description Framework (RDF) . . . 9
2.2 OWL: Web Ontology Language . . . 10
2.3 SPARQL . . . 10
2.4 Graph Query Languages . . . 11
2.4.1 Graph Query Languages . . . 12
2.4.2 Imperative vs Declarative . . . 13
2.4.3 Graph querying in the real world . . . 14

3 Benchmarking Core Database Functionalities 15

4 Benchmarking Graph Databases 16
4.1 Early efforts . . . 16
4.2 Graph Database Benchmarks . . . 16
4.2.1 HPC Scalable Graph Analysis Benchmark . . . 16
4.2.2 Graph 500 . . . 17
4.2.3 Ciglan . . . 18
4.2.4 XGDBench . . . 19
4.2.5 Other Industrial Initiatives . . . 19
4.2.6 Comparative Analysis . . . 19

5 Benchmarking RDF databases 22
5.1 Benchmarks Using Real Datasets . . . 22
5.2 Benchmarks Using Synthetic Datasets . . . 23
5.3 Concluding Remarks . . . 25

6 Data Integration Benchmarks 26
6.1 Benchmarking Instance Matching Systems . . . 26
6.1.1 Ontology Evaluation Alignment Initiative (OAEI) . . . 27
6.1.2 ONTOlogy Matching Benchmark With Many Instances (ONTOBI) . . . 29
6.1.3 STBenchmark . . . 30
6.1.4 Discussion on Instance Matching Benchmarks . . . 30
6.2 Benchmarking RDF Link Discovery Systems . . . 32
6.3 ETL Benchmarks . . . 32
6.3.1 TPC-DS . . . 32
6.3.2 The Linked Open Data Integration Benchmark (LODIB) . . . 33

7 Benchmarking Principles 34

8 Conclusions 38

List of Figures

6.1 Modifications for Instance Matching 27

List of Tables

4.1 Comparative Table of Supported Features by the Existing Graph Database Benchmarks . . . 20
4.2 Graph Operations, Areas of Interest and Categorization 21

6.1 Links from the NYTimes dataset to FreeBase, DBPedia and GeoNames 29
6.2 Simple and Complex Modifications in ONTOBI 29
6.3 Comparison of Instance Matching Benchmarks: "+", fully satisfied, "o" partially satisfied and "-" not satisfied. 31

1 Introduction

Objective, well-defined and good quality benchmarks constitute a critical component for the progress of technology, as they allow vendors and users to compare the performance of products and systems in a comprehensive manner along different dimensions and for different application needs. The main objective of the Linked Data Benchmark Council (LDBC) is the development of benchmarks for different technology areas including core data management (query processing, query optimization, transactions), graph analysis, and data integration and reasoning; as well as building consensus over these benchmarks in the RDF and graph data management communities at large.

Our effort towards this goal starts with *i)* a *review* of the existing benchmarks for RDF and graph databases and *ii)* with a discussion on the *principles* that benchmarks should adhere to. More specifically, we present the state of the art of benchmarks for (a) core database functionalities, (b) graph databases (c) RDF databases and (d) RDF data integration approaches (the latter covering instance matching and ETL techniques). In addition to surveying existing works, we also discuss benchmarking principles focusing mainly on the purpose and scope of benchmarks.

The objective of this deliverable is two-fold:

- we review and provide a comprehensive survey of existing benchmarks for a) core database functionalities, (b) graph databases (c) RDF databases and (d) RDF data integration approaches (the latter covering instance matching and ETL techniques); based on this report, we can then identify ideas, practices, techniques or solutions that can be reused for the envisioned suite of LDBC benchmarks;

- second, by identifying the shortcomings and limitations of existing benchmarks, we can properly focus our efforts in LDBC into proposing benchmarks that deal with the challenges imposed by new applications. A result of this effort is a first attempt to express a set of principles to which new benchmarking efforts should adhere.

This deliverable is structured as follows: Chapter 2 presents the basics of the Resource Description Framework (Section 2.1) and Web Ontology Language (Section 2.2) for representing and processing semantic web data. We discuss SPARQL, the W3C standard language for querying RDF data in Section 2.3 and provide a brief overview of graph database query languages in Section 2.4. In Chapter 3 we discuss existing benchmarks for core database functionalities. Graph and RDF database benchmarks are presented in Chapters 4 and 5 respectively. Chapter 6 discusses data integration benchmarks. In Sections 6.1, 6.2 and 6.3 we present existing benchmarks for instance matching, link discovery and ETL processes. Benchmarking principles are presented in Chapter 7. Finally we discuss conclusions in Chapter 8.

2 Background

2.1 Resource Description Framework (RDF)

The objective of the Semantic Web is to build an infrastructure of machine-readable semantics for data on the Web. In 1998, World Wide Web Consortium (W3C) issued a recommendation for a data model to serve as the basis for such infrastructure, the Resource Description Framework (RDF) [31]. This framework enables the encoding, exchange, and reuse of structured data, while providing the means for publishing both human-readable and machine-processable vocabularies. As RDF evolves, it is increasingly gaining attraction from both researchers and practitioners, and is being implemented in world-wide initiatives such as the Open Directory Project [114], Dublin Core [91], FOAF [95], and RSS [119]. RDF follows the W3C design principles of *interoperability*, *extensibility*, *evolution* and *decentralization*. Particularly, the RDF data model is *simple*, with a formal semantics and provable inference and with an extensible URI-based vocabulary which allows anyone to make statements about any resource. In addition, the RDF specification includes a built-in vocabulary with a normative semantics (RDFS). This vocabulary deals with inheritance of classes and properties, as well as typing, among other features [13].

The basic assumptions underlying RDF are:

- In RDF essentially anything we wish to describe is a *resource*. A resource may be anything from a person to an institution, the relation a person has with an institution, a Web page, part of a Web page, an entire collection of pages or a Web site. In the context of RDF, a resource is uniquely identified by a URI (Universal Resource Identifier) [138].
- The building block of the RDF data model is a *triple*. A triple is of the form *(subject, predicate, object)* where the *predicate* (also called property) denotes the *relationship* between *subject* and *object*. An *RDF graph is a set of triples*.

An RDF graph can be viewed as a *node* and *edge labeled directed graph* with subjects and objects of triples being the nodes of the graph and predicates being the edges. RDF data do not necessarily come with a schema or semantics (expressed by constraints). The same predicate may be used between different multiple classes of subjects/objects that are not explicitly stated as classes. In RDF data the predicates themselves may also re-appear as subjects and objects mixing data and metadata in a big graph. Those intrinsic properties of RDF raise interesting challenges for RDF engines and that should be tackled by RDF benchmarks.

Resource Description Framework Schema vocabulary (RDF Schema [13]), is designed to introduce useful semantics to RDF triples. In particular, RDFS introduces some special constructs useful for classifying resources into classes, properties, instances etc. (*typing of resources*), as well as some useful relationships (properties) between resources, like *subsumption* or *instantiation*. More specifically, the most important constructs of the RDFS vocabulary are:

- A set of *classes*, that denote sets of resources and are used for classifying resources into different *types*, like classes, properties, instances etc. An instantiation of a resource under one of these classes (via the property rdf:type) indicates its type (e.g., instances of the class rdfs:Class are classes).
- A set of *properties*, which denote certain relationships between resources. For example, subsumption between properties is denoted by the property rdfs:subPropertyOf, subsumption between classes is denoted by the property rdfs:subClassOf and properties can be attributed specific domains and ranges via rdfs:domain and rdfs:range respectively.
- RDFS proposes also a set of *inference rules* that are used to *entail* information from the *explicit triples* in an RDF graph. Inference rules essentially encode the semantics of the RDFS classes and properties described previously, such as the transitivity of rdfs:subclass and rdfs:subproperty relations, inheritance of instantiation and attributes along the class subsumption hierarchy, and others. Inference is implemented by applying the rules *forward* (i.e., *forward reasoning*), or treating the conclusion as a

pattern to be recognized in the consequent of a proposed entailment and searching *backwards* for an appropriate match with other rule conclusions or the proposed antecedent *backward reasoning*.

2.2 OWL: Web Ontology Language

The OWL Web Ontology Language [50] is designed for use by applications that need to process the content of information instead of just presenting information to humans, and is used to (a) *create an ontology*, (b) *state facts* about a domain and (c) *reason about ontologies* to determine consequences of what was named and stated.

OWL is similar to RDF/S, but provides a much richer set of constructs and semantics that allows more complicated reasoning. As with RDF/S ontologies, an OWL ontology is a collection of *classes* and *properties*. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDFS) by providing *additional vocabulary* along with a *formal semantics*.

OWL statements come in three syntaxes: *(i)* RDF, *(ii)* XML and *(iii)* an abstract syntax that is easier to read and write, which corresponds more to Description Logic languages [6]. The RDF/XML syntax is used in order to make OWL part of the Semantic Web and use it as an extension of RDF. In this way, RDF specific tools can parse and manage OWL statements.

OWL has three increasingly-expressive sublanguages, namely *OWL Lite*, *OWL DL*, and *OWL Full*. *OWL Lite* is the least expressive of the three sublanguages (but still more expressive than RDF/S) and allows the specification of *classification hierarchies* and *simple constraints*. It enjoys *lower computational complexity* than OWL DL. *OWL DL* provides *increased expressive power* compared to OWL Lite, while retaining *computational completeness* (all conclusions are guaranteed to be computable) and *decidability* (all computations will finish in finite time). Finally, *OWL Full* provides *maximum expressiveness* but *no computational guarantees*. It allows an ontology to augment the meaning of the predefined (RDF or OWL) vocabulary, but due to the lack of any computational guarantees it is unlikely that OWL Full will be supported by a reasoning system.

OWL includes all features of RDFS and additional features that allow one to:

- distinguish between *object* and *data type* properties,
- state that two *classes* and *properties* are *equivalent* (owl:equivalentClass and owl:equivalentProperty)
- specify that a property is *symmetric, transitive, functional, inverse functional*, and get its values from a specific class (OWL properties owl:allValuesFrom, owl:someValuesFrom),
- define anonymous classes as in the case of XML Schema Language,
- state that two instances refer to the same real world object (owl:sameAs), or are the same or different (owl:sameIndividualAs, owl:differentFrom),
- use of occurrence indicators (owl:cardinality) and finally to define classes using set operations (owl:intersectionOf, owl:unionOf, owl:complementOf), or by enumerating its members (owl:one of) or stating that it is disjoint from another owl:disjointFrom).

OWL DL and OWL Full support all the above constructs, with OWL Full also allowing one to treat simultaneously a class as an individual and as a collection of individuals. OWL Lite supports a light version of cardinality constraints (it only permits cardinality values of 0 or 1), and does not support set operations for defining classes.

2.3 SPARQL

During the past years, many query languages have been proposed for the RDF data model, such as RQL [41], RDQL [71], SeRQL [14], and TRIPLE [72]. SPARQL 1.0 [63] is the official W3C recommendation language

for querying RDF graphs and has the ability to extract information about both data and schema. SPARQL 1.1 [35] has only recently become a W3C recommendation and extends SPARQL 1.0 with more complex constructs that we will describe below. In this deliverable, we focus on SPARQL.

SPARQL is based on the concept of *matching graph patterns*. The simplest graph patterns are *triple patterns*, which are like an RDF triple but with the possibility of a *variable* in any of the subject, predicate or object positions of the triple. A query that contains a *conjunction of triple patterns* is called a *basic graph pattern*. A basic graph pattern matches a subgraph of the RDF graph when variables of the graph pattern can be substituted with the RDF terms in the graph, and the assignment of values to variables is called a *binding*.

SPARQL queries follow the SQL paradigm and consist of three parts: the *pattern matching part*, which includes several interesting features of pattern matching of graphs, like *optional* parts (operator optional), union of patterns, (operator union) nesting, filtering (or restricting) values of possible matchings (operator filter), and the possibility of choosing the data source to be matched by a pattern using the named graph concept (clause GRAPH). SPARQL support *(a)* select queries that return variable bindings (in the relational sense), *(b) boolean queries* through the use of the ask clause that essentially check whether a graph pattern is satisfied for a specific RDF graph, and *(c)* queries that create new graphs out of existing ones through the use of construct clause. The solution modifiers, which, after the output of the pattern has been computed (in the form of a table of values of variables), allow to modify these values by applying classical operators like *projection*, *distinct*, *order*, *limit*, and *offset*.

SPARQL 1.0 was lacking *aggregates* or *subqueries* of any sort, features that are significant for business intelligence or decision support applications. SPARQL 1.1 was proposed to fix most of these significant limitations, and one can now express almost everything that can be expressed using SQL [122].

SPARQL 1.1 [35] introduces new features that were lacking from SPARQL 1.0 and are significant for business intelligence or decision support applications. Such features include *aggregates*, *path expressions*, *subqueries* and *value assignment*. Path expressions, otherwise known as *property paths*, are essentially *regular expressions*, that are used to retrieve pairs of nodes from an RDF graph if they are connected by paths conforming to the ones specified in the query. SPARQL 1.1 also supports *existential quantification* and *negation*. The set operators are unfortunately limited to the SPARQL 1.0 union operator, but there are other ways of expressing this missing functionality. SPARQL 1.1 provides an extensive path language for abbreviated expression of long series of joins that also involve transitive and optional steps. The path language does not allow returning the intermediate steps along the provided path and so it cannot be used for queries like *"find the shortest path"* between nodes x and y that are very common in graph databases. The new features of SPARQL 1.1 allow one to express almost everything that can be expressed using SQL [122].

SPARQL refers to the XQuery built-in function library for the use of functions. It does not have a concept of *node sequences* as in XML since the building block of the RDF data model is the triple and not the node (as is the case with XML). XML elements can appear in RDF triples, but SPARQL 1.1. does not support operations on these elements. SPARQL 1.1. does not discuss stored procedures, their support is left to the underlying system used for the evaluation of the queries. SPARQL 1.1. also supports updates, but considers that *all* operations are atomic and hence it does not have any notion of transactionality.

In addition, SPARQL 1.1 supports *federation*, through the definition of federated queries that support the explicit delegation of subqueries to different SPARQL endpoints.

2.4 Graph Query Languages

Due largely to the Web, an exponentially increasing amount of data is generated each year. Moreover, a significant fraction of this data is unstructured, or semi-structured at best. This has meant that traditional data models are becoming increasingly restrictive and unsuitable for many application domains – the relational model in particular has been criticized for its lack of semantics. These trends have driven development of alternative database technologies, including graph databases [5]. Graph database models are applied in areas where information about data inter-connectivity or topology is as important as the data itself. The development of graph databases was driven by applications where data complexity exceeded the capabilities

of the relational data model: flatness of permitted data structures, difficulty of discovering data connectivity, and challenges of modeling complex domains. Examples of such application domains are those working with complex networks: social networks, information networks, technological networks, and biological networks.

- Social networks – the representation of people/groups and their relationships (friendship, business connection, sexual contact, research collaborator, etc.).
- Information networks – information flow or connectivity in domains such as the World Wide Web and peer-to-peer networks.
- Technological networks – networks in which the spatial aspects of data are dominant: the Internet, power grids, transportation networks, etc.
- Biological networks – biological information such as networks that occur in gene regulation, metabolic pathways, chemical structure, and homology relationships among species.

The proliferation of applications dealing with complex networks has resulted in an increasing number of graph database deployments. This, in turn, has created demand for a means by which to compare the characteristics of different graph database technologies, such as: performance, data model, query expressiveness, as well as general functional and non-functional capabilities.

To fairly compare these technologies a uniform way of expressing queries is beneficial, if not essential. Although much research on graph query languages exists, there is still no standardized method of expressing queries for graph databases – no graph database analog to SQL from the relational database world.

2.4.1 Graph Query Languages

Associated with graphs are specific graph operations in the query language algebra [5], which refer directly to the underlying graph structure. These operations can be condensed to the following query types [68]:

- **Subgraph query**: find all graphs in the database such that a given query graph is a subgraph of them.
- **Supergraph query**: find all graphs in the database that are subgraphs of the given query graph.
- **Pattern match query**: find instances of a pattern graph (e.g. path, star, subgraph) within a large graph.
- **Reachability query**: special case of pattern matching, find the path(s) between any two vertices in the large graph.
- **Shortest path query**, special case of reachability queries, return only the shortest path(s) between any two vertices in the large graph.

Further, as specific queries are regularly used in the same types of applications, the above query types can be categorized according to the domains in which they are applied:

- **Applications dealing with many small graphs**: These are common in domains like bioinformatics, cheminformatics, and repositories of business process models. Query types commonly used:
  - Subgraph query
  - Supergraph query
- **Applications dealing with few large graphs**: These are common in domains like social networks, bibliographical networks, and knowledge bases. Query types commonly used:
  - Pattern match queries
  - Reachability queries

– Shortest path queries

The databases and languages discussed here mostly fall into the second of these categories - those that model large graphs, and query for patterns and the existence of paths. Moreover, special focus will be given to languages designed for querying attributed graphs; applications in this category commonly require the ability to express pattern matching queries based on path patterns [68], where patterns combine both structural (topological) predicates and value (attributes of vertices and edges) predicates.

When working with graph query languages, database transformations are regarded as graph transformations. Based on graph-pattern matching, they allow insertions and deletions to be specified graphically - often regarded to be more intuitive. A significant strength of the graph database model is that it makes graphs and graph operations explicit; that is, it allows users to express queries at a higher level of abstraction, easing the task of working with data in the graph/network domain. This contrasts with query languages for deductive databases (i.e. the relational data model), where complex rules must be written to perform graph manipulations. In particular, when using deductive databases it is difficult for the query language to explore the underlying graph structure, to compute: paths, neighborhoods, patterns, reachability, and graph statistics (e.g. diameter or centrality).

On the other hand, computing reachability of information, which translates to path problems characterized by recursive queries, is natively supported by graph query languages like GraphLog [23], Gram [4], Cypher [80], and G-SPARQL [68], and partially supported by GOAL [37], Hyperlog [62] , GraphDB [34] , G-Log [59], and HNQL [46] - **the problem is, few of these languages simultaneously support both the attributed graph model, and path/pattern graph queries**.

Though rapidly maturing, graph databases and their related technologies are still relatively young. In industry, various query language implementations are in use, most undergoing active development. In academia, numerous works of related research exist, and many more are produced each year - graph query language research spans over 30 years, and has received contributions from many different fields. However, unlike the world of relational databases, where SQL [122] has become ubiquitous, a standard query language for querying graph databases is yet to emerge - the field is still vast and unsettled.

2.4.2 Imperative vs Declarative

One useful way of categorizing query languages is by separating them into one of the two following groups, imperative and declarative.

In imperative languages algorithms are implemented in terms of explicit steps. A program defines some sequence of commands for the computer to perform; computation is described in terms of statements, which change the state of a program. In contrast, declarative languages express computation logic without describing its control flow; they specify what a program should accomplish rather than the sequence of actions to be taken - the how (specific sequence of actions) is left to the language implementation. Additionally, such languages often attempt to minimize, or eliminate, side effects.

Although two of the most successful database query languages, SQL and SPARQL, are both declarative, each approach has its unique set of advantages. In particular, there are clear trade-offs regarding: expressiveness, simplicity, performance, optimization, and portability.

Generally, imperative languages are more expressive, i.e., it is possible to implement arbitrary algorithms. Moreover, they provide greater control over how a task is executed, giving skillful, experienced developers the ability to manually optimize the implementation of a query. The advantages of declarative query languages are manifold: using the correct level of abstraction simplifies the task of expressing queries; removing the need to describe how a query should be performed results in more concise syntax; they are easier to port across different database systems; and, perhaps most importantly, it allows the database to create its own query plan, making it possible to apply optimizations based on access patterns, graph structure, etc.

Native graph database systems (e.g., Neo4j [109], DEX [88], OrientDB [116], InfiniteGraph [101]), are designed for efficient querying of persisted graphs. However, many such systems currently lack declarative language support, limiting portability and optimization opportunities, and increasing programmer burden. In the absence of declarative language support the only means of querying these databases is via proprietary,

low-level, language-specific, imperative interfaces. As a result, the efficiency of executing a graph query not only becomes programmer-dependent, but assumes that the programmer has sufficient knowledge and skill, which is rarely the case.

2.4.3 Graph querying in the real world

With a few notable exceptions [68, 36], most graph query methods from academia focus on querying the topological structure of graphs, but few consider the attributes on graph elements (nodes and edges). However, graph datasets used in modern applications are not only directed and labeled, but also attributed, and the queries of large graph database applications (e.g. social or citation networks) often consider both these attributes and the graph topology. There is clear a disconnect between the graph query work being performed by academia and industry.

As querying these graphs has received limited attention in the literature, there is little theoretical foundation for building query engines that query such graphs efficiently, especially at the scale (volume of data and load) of modern applications.

3 Benchmarking Core Database Functionalities

In the context of relational databases there exist some well-established benchmarks, namely the well-known TPC benchmarks [129]. Although relational databases are not considered in this project, relational benchmarks introduced several best practice techniques that could serve as inspiration for benchmarking core database functionalities in LDBC. TPC is a family of benchmarks:

- TPC-C [130] is an on-line transaction processing (OLTP) benchmark. It supports multiple transaction types and complex database queries. It simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. The database is comprised of nine types of tables with a wide range of record and population sizes. TPC-C involves a mix of five concurrent transactions of different types and complexity either that are executed on-line or queued for deferred execution.
- TPC-E [133] simulates the OLTP workload of a brokerage firm where the focus of the benchmark is the central database that executes transactions related to the customer accounts of a specific company. The underlying database schema, data population, transactions, and implementation rules are representative of modern OLTP systems.
- TPC-H [134] is an ad-hoc decision support benchmark. It is the most widely used benchmark for testing the performance of database systems. It consists of a suite of business oriented ad-hoc queries with high degree of complexity and concurrent data modifications that have been chosen to have broad industry-wide relevance.
- TPC-DS [132] is the new decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance.

TPC benchmarks use different metrics for measuring the performance of the systems. OLTP benchmarks (TPC-C and TPC-E) use transactions per minute ($tpmC$) metric. The analytical benchmarks (TPC-H) and the obsolete TPC-R [135] and TPC-D [131] ones, use the *queries-per-hour-at-size* ($QphH@Size$), and *costs-per-performance* ($Price/QphH@Size$) metrics. The first of the two, $QphH@Size$, allows comparing the performance at different scale factors, as some architectures might be superior for very large data sets and others might aim at smaller data sizes. The second metric, $Price/QphH@Size$, normalizes the performance by costs which are specified in the respective benchmarks, and include, for example, hardware, software licenses, and maintenance costs for a certain period of time. Again, this metric allows for more reasonable performance comparisons, as some systems might offer excellent performance but have very high hardware and software costs, while other systems might be slower but offer a better cost/performance ratio.

Although highly influential, the TPC benchmarks are not flawless. One particular limitation of the most commonly used TPC benchmarks (TPC-H and TPC-C) is that the generated data is uniformly distributed and uncorrelated. While this simplifies the setup, it also ignores very challenging real-world problems. The TPC-DS[132] benchmark was designed to explicitly test for data skew, and also to test for more complex queries. Another limitation of TPC-H is that it does not show the classical star schema as expected from warehousing applications. A star schema is a normalized schema that has a main table and several dimension tables (fact tables) that represent the points of the star. On the other hand, the Star Schema Benchmark [57] uses a schema that is closer to the canonical form.

While TPC-C is a well known benchmark for online transaction processing, and TPC-H is a well known benchmark for decision support systems, they both concentrate either on update transactions or on complex queries. But many applications need not just one of the two sides, but both. The CH-benCHmark mixed workload benchmark [22] therefore combines TPC-C and TPC-H into one unified benchmark that stresses both update and query functionality.

4 Benchmarking Graph Databases

4.1 Early efforts

Popular database benchmarks, such as TPC-C or TPC-H [129], focus on evaluating relational database queries that are typical of a business application. These benchmarks emphasise queries with joins, projections, selections, aggregations and sorting operations. However, since Graph Databases (GDBs) aim at different types of queries, these widespread benchmarks are not adequate for evaluating their performance. Graph use cases often involve recursive steps, e.g. graph neighborhoods within n steps or even an unbounded (at least a priori unbounded) number of steps. Graph queries may involve structural similarity, e.g. comparing structures of chemical compounds for similarity with a similarity score quantifying the deviations. Graph analytics often produce large intermediate results with complex structure, e.g. edge weights or iteratively calculated ranks (e.g. PageRank). Widespread relational benchmarks do not contain such operations. All those are operations that, in some cases, are difficult to imagine in RDBMSs and that find a good alliance in the RDF area and GDB area since the former adhere to a graph data model.

Object oriented databases (OODB) share some similarities with GDBs. The data of an OODB also forms a graph structure, where the entities that are represented as objects draw relationships among them. The OO1 benchmark [18], one of the earliest proposals, is a very simple benchmark that emphasises three basic operations for OODB: (a) lookup, which finds the set of objects for a given object identifier; (b) traversal, which performs a 7-hop operation starting from a random node; and (c) insertion, which adds a set of objects and relations to the database. OO1 defines a dataset that only contains one type of objects with a fixed number of outgoing edges per object. Since the links mostly go to objects with a similar document identifier, the graphs are very regular. Another popular benchmark for OODB is the OO7 proposed by Carey et al. [15] In OO7, the database contains three types of objects, which are organised as a tree of depth seven. The connectivity of the database is also very regular because objects have a fixed number of relations. The benchmark is made up by a rich set of queries that can be clustered into two groups: (a) traversal queries, which scan one type of objects and then access the nodes connected to them in the tree, and (b) general queries, which mainly perform selections of objects according to certain characteristics. Although these OODB benchmarks create graphs, the graphs have a very different structure from typical graphs in graph analysis applications. More specifically, graphs in GDBs are very irregular: the degree of the nodes exhibit a large variance, nodes are clustered in communities and graphs have small diameters. Furthermore, the applications that interact with GDBs are mainly interested in analyzing the graph structure, i.e. the relationships, instead of the attributes in the objects. For example, operations such as finding the shortest path connecting two objects or finding patterns (e.g. a clique) are common GDB operations that are not considered in OODB.

XML databases also follow a model which relates entities. An XML database is a collection of typed data and attributes organized as a tree. One of the most well known benchmarks for XML databases is XMARK [69], which models an auction site. The queries in the benchmark cover many aspects: selections, sorted access, tree path location, aggregation, etc. Nevertheless, XML only models trees, which are a limited subclass of graphs.

4.2 Graph Database Benchmarks

4.2.1 HPC Scalable Graph Analysis Benchmark

The *graphanalysis.org* initiative was one of the first to start a project to evaluate graph performance. After some preliminary benchmark proposals such as the DARPA High Productivity Computing Systems (HPCS) Compact Application (SSCA) suite, which refined the queries in the system, the project released the final version of the benchmark as the "HPC Scalable Graph Analysis Benchmark v1.0" [7].

The benchmark consists of four separated operations (kernels) on a graph that follows a power law distribution generated with the Recursive MATrix (R-MAT) generator [19]. This model recursively sub-

divides the adjacency matrix of the graph into four equal-sized partitions and distributes edges within these partitions with unequal probabilities. Initially, the adjacency matrix is empty, and edges are added one at a time. Each edge chooses one of the four partitions with probabilities *a*, *b*, *c*, and *d*, respectively. At each stage of the recursion, the parameters are varied slightly and re-normalized. For a given $SCALE$ factor, the generated graph contains $N = 2^{SCALE}$ nodes and $M = 8N$ directed edges connecting two adjacent nodes in N. Each edge also has an integer weight chosen uniformly at random from the distribution $[1, 2^{SCALE}]$. An implementation may use any set of N distinct integers to number the vertices, but at least 48 bits must be allocated per vertex number.

The four kernels of the benchmark are:

1. **Graph Construction**: *insert* the graph database as a bulk load from a list of *tuples*; each tuple contains the start and end vertex identifiers for a directed edge, and the a weight that represents data assigned to the edge. Implementers are free to represent the graph in any manner, but it cannot be modified by or between subsequent kernels.

2. **Classify Large Sets**: *retrieve* the set of *edges with maximum weight*. The output is an edge list that will be saved for use in the following kernel.

3. **Graph Extraction**: *perform* a *k-hops* operation (e.g. Breadth-First Search). For each of the edges in the previous list, extract the subgraph which consists of the vertices and edges of all the paths of length 3 (3-hops) starting with that edge.

4. **Graph Analysis**: *calculate* the *betweenness centrality* of a graph and identify the set of vertices in the graph with the highest betweenness centrality score. Betweenness Centrality of a vertex v is defined as:

   $$BC(V) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

   where σ_{st} is the number of shortest path between vertices s and t, and $\sigma_{st}(v)$ is the number of those paths passing through V. Because of the high computation cost of the exact solution, the benchmark allows for an approximate implementation that uses a subset of starting vertices. The performance of the operation is measured as the number of traversed edges per second (*TEPS*), in the spirit of well-known computing rates such as the floating-point operations per second (flops) measured by the Linpack benchmark.

The validation of the benchmark is easy for the first three linear-time kernels, but for the fourth one with an approximate result it is necessary to generate data using a synthetic 2-dimensional torus generator. Finally, the goals for the evaluation criteria of the benchmark results are, in order of importance, are (i) the fair adherence to the benchmark specification; (ii) the maximum problem size; and (iii) the minimum execution time.

Concluding remarks This benchmark does not evaluate some features expected from a GDB such as object labelling or attribute management. In Domínguez-Sal et al. [27], this benchmark is evaluated on four representative graph data management alternatives (Neo4J [109], DEX [88], Jena [103] and HypergraphDB [100]) giving some insights about the strengths and weakness of each system. Based on the experience implementing and testing it on real hardware, its recommendation to the HPC benchmark is to harmonize the costs of the query kernels (b, c and d) with operations that scale similarly with the graph size in order to obtain a more balanced benchmark, and perform the k-hops operation in Kernel 3 over a larger set of nodes as origin, for example by picking a fixed percentage of edges.

4.2.2 Graph 500

The performance of supercomputers has been traditionally tested using the Linpack benchmark, which is derived from the Linpack library that computes linear algebra operations. According to the Linpack results,

a list of the top 500 computers is published biannually, which determines the most powerful computers in the world. Nevertheless, the use of supercomputers has spread from computationally intensive integer and floating point computation, to memory intensive applications. For such applications, Linpack is not a good reference and other evaluation methods have been proposed, including graph related computation. In 2010 a steering committee of over 50 international HPC experts from academia, industry, and laboratories defined the Graph 500 benchmark [52] based on the previous experience with the HPC scalable benchmark. The intent of the benchmark is to develop a compact application that has multiple analysis techniques (multiple kernels) accessing a single data structure representing a weighted, undirected graph. It also expects that in the future there will be at least three variants of implementations: on shared memory and threaded systems, on distributed memory clusters, and on external memory map-reduce clouds. The committee is in the process of developing comprehensive benchmarks to address three application kernels: concurrent search, optimization (single source shortest path), and edge-oriented (maximal independent set).

The current version 1.2 of the benchmark classifies the problems based on the input size, from *toy* (17GB) to *huge* (1.1PB), and proposes two kernels: one to load data generated from a synthetic graph generator (R-MAT), and another to measure the edge traversals of the Breadth-First Search (BFS) starting from 64 sampled search keys. The dataset generator follows the same approach as in the HPC benchmark but with an edge factor of 16 for each vertex instead of the original 8. For kernel 2, the 64 random vertices selected must be connected to some other vertex, so their degrees, not counting self-loops, must be at least one. Also, the Breadth-First Search runs without a limit on the number of hops. Finally, BFS is measured as in the HPC benchmark in traversed edges per second (*TEPS*), counting any multiple edges and self-loops. Prefix multipliers are used to denote large TEPS factors such as KTEPS (thousands), MTEPS (millions) and GTEPS (billion). Until today, the largest and fastest experiment reported was in November 2012 with a measure of 15363 GTEPS for a problem scale of 40 (input size *large*, approx. 281TB of data). The benchmark was executed at the Lawrence Livermore National Laboratory in a supercomputer IBM - BlueGene/Q, Power BQC 16C 1.60 GHz with 65536 nodes and 1048576 cores.

Concluding remarks This benchmark lacks of the same issues as the HPC benchmark: the graph is very simple, with no labelling and attributes, and there is a single query kernel for graph traversals. Also, the last specification is v1.2 from September 2011 and since then new query kernels have not been proposed.

4.2.3 Ciglan

In year 2012 Ciglan [20] published a set of traversal oriented queries over graph databases. The proposed benchmark had two intentions: query-like traversals, where one searches for related vertices for a given vertex (e.g. a breath-first traversal), and graph analysis operations that require one or multiple traversals of the whole graph (e.g. centrality measures).

For this benchmark, the graph data model is a directed multigraph with attributes in vertices an edges. Datasets are generated synthetically using the LFR benchmark generator [45], which generates power-law degree distributions. It is widely used to detect communities into networks.

The benchmark consists in three operations: (i) graph population; (ii) a 2-hop BFS of ten thousand randomly chosen vertices to simulate the measure of the vertex clustering coefficient; and (iii) a 3-hop BFS of another ten thousand randomly chosen vertices. The benchmark was tested with five graph database repositories: Neo4J [109], DEX [88], OrientDB [116], Native RDF repository (NativeSail) [115] and a prototype of SGDB [21]. All the database engines are accessed to a standard Blueprints 1.0 interface [78]. Experiments are run with different scale factors up to 100000 vertices and 1 million edges.

Concluding remarks Even if the intentions of the benchmark are "*to extend the discussion on the design of graph database benchmarks, focusing on traversal operations in a memory constrained environment, where the whole graph can not be loaded and processed in memory, and to present the complete design of a graph database benchmark*", the scope of the benchmark is very narrow, with only two BFS queries limited in depth, and the experimental validation was made with small graphs that can be processed easily with limited

memory resources. Also, even if the graph model supports attributes, there not exists any query with attribute filters or aggregates.

4.2.4 XGDBench

The latest initiative in graph database benchmarks is XGDBench [25]. It has been designed to operate in current cloud service infrastructures as well as on future exascale clouds (computing systems capable of at least one exaflop), even if exascale computing is not expected at least until 2018. The infrastructure is built on top of the Yahoo! Cloud Serving Benchmark YCSB [24], a standard benchmark and benchmarking framework to assist evaluation of cloud data serving systems. The benchmark platform is written in X10 which is a PGAS language intended for programming future exascale systems.

The dataset generator is the Multiplicative Attribute Graph (MAG) [42], an approach for modeling the structure of networks which have node attributes to create realistic attribute graphs. An important part of the XGDBench work consists in the analysis of the generator and its comparison with the classical R-MAT, in particular to detect communities.

The experimental validation of the benchmark is done with four graph database engines: AllegroGraph [74], Fuseki [96], Neo4J [109], and OrientDB [116]. The experiments are executed in a simple simple architecture with one server and one client on an HPC could environment because most of the graph database engines are not distributed. Also, the number of vertices is very small (1024) because some database servers used performed poorly. For the queries, the benchmark identifies six operations: *read* a vertex and its properties, *insert* a new vertex, *update* all attributes of a vertex, *delete* a vertex, *scan* the neighbors of a vertex, and *traverse* the graph from a given vertex using a breadth-first search. However, the *delete* and *traverse* operations are not used in the five workloads executed, and the *update* is not for all attributes.

Concluding remarks The only real contribution of this work is the proposal and analysis of a different dataset generator that supports attributes and simulates the phenomenon of social affinity that is present in real social networks. The workloads and queries are very poor, even more if they are intended for future exascale clouds. Also, the experimental validation is very simple and does not match the intentions of the benchmark.

4.2.5 Other Industrial Initiatives

Other initiatives from graph database vendors have proposed simple benchmarks to evaluate the performance of graph databases. Tinkerpop initiated a project (currently stopped) to build a framework for running benchmarks on graph databases [127]. InfiniteGraph, another GDB vendor [101], has published an internal benchmark to analyze the performance of their distributed GDB and to compare it with another Open Source GDB. Nevertheless, such initiatives lack a wide acceptance because of their individual approach and limited resources.

4.2.6 Comparative Analysis

Table 4.1 shows a comparison of the supported features by the existing graph database benchmarks: HPC, Graph 500, Ciglan and XGDBench. It shows clearly that this area is still in an early stage and a lot of work must be done to cover all graph database characteristics and benchmark goals. Also, the lack of standards for graph modeling and query languages will make the benchmark definition and construction process even more difficult.

Recently, a more detailed survey has reviewed some of the main operations and uses cases of graph databases [26]. It starts by describing and analyzing the type of applications where it is necessary the use of GDBs. such as Social Network Analysis (SNA), proteomics, recommendation systems, travel planning and routing, which gives a catalog of representative areas where huge graph datasets are appearing. First, the survey identifies all the different graph features, topologies and properties: attribute values for nodes

Class	Operation	HPC	Graph 500	Ciglan	XGDBench
Dataset	*R-MAT*	edge factor 8	edge factor 16	-	-
	LFR	-	-	yes	-
	MAG	-	-	-	yes
Topology	*insert nodes*	-	-	-	yes
Attributes	*read node*	-	-	-	yes
	update node	-	-	-	yes
Filtering	*edge attribute*	yes	-	-	-
Traversals	*k-hops BFS*	3-hops	unlimited	2-hops, 3-hops	1-hop
Centrality Measures	*betweenness*	yes	-	-	-

Table 4.1: Comparative Table of Supported Features by the Existing Graph Database Benchmarks

and edges, directed and undirected edges, node and edge labelling, multigraphs and hypergraphs. Then, the authors classify the different graph operations in several groups:

1. *traversals*: operations that start from a single node and explore recursively the neighborhood until a final condition, such as the depth or visiting a target node, is reached.

2. *graph analysis*: includes the study of the topology of graphs to analyze their complexity and to characterize graph objects.

3. *components*: subsets of the nodes of the graph where there exists a path between any pair of nodes.

4. *communities*: set of nodes where each node is closer to the other nodes within the community than to nodes outside it.

5. *centrality measures*: to give a rough indication of the social importance of a node based on how well this node connects the network.

6. *pattern matching*: algorithms which aim at recognizing or describing input patterns.

7. *graph anonymization*: process to generate a new graph with properties similar to the original one, avoiding potential intruders to re-identify nodes or edges.

Table 4.2 gives a summary of the different graph operations used in each use case scenario and graph topologies. Finally, the survey also starts the discussion on the experimental setup and the measures for a GDB benchmark.

Group	Operation	Social Network	Protein Interaction	Recom-mendation	Routing	Analytical	Cascaded	Scale	Attr.	Result
Generic operations										
General Atomic / Local Information Extraction	Get node/edge	+	+	+	+	Yes	No	Neigh.	No	Set
	Get attribute of node/edge	+	+	+	+	Yes	No	Neigh.	No	Set
	Get neighborhood	+	+	+	+	Yes	No	Neigh.	No	Set
	Node degree	+	+	+	+	Yes	No	Neigh.	No	Agr.
General Atomic Transformations	Add/Delete node/edge	+	+	+	+	No	No	Neigh.	No	Set
	Add/Delete/Update attrib.	+	+	+	+	No	No	Neigh.	E/N	Set
Application dependent operations										
Traversals	(Constrained) Shortest Path	+	+		+	Yes	Yes	Glob.	Edge	Graph
	k-hops	+		+	+	Yes	Yes	G/N	No	Graph
Graph Analysis	Hop-Plot	+				Yes	No	Glob.	No	Agr.
	Diameter	+	+			Yes	Yes	Glob.	Edge	Set
	Eccentricity	+				Yes	Yes	Glob.	Edge	Agr.
	Density	+	+			Yes	No	Glob.	No	Agr.
	Clustering coefficient	+				Yes	Yes	Glob.	No	Agr.
Components	Connected Components	+	+			Yes	Yes	Glob.	No	Graph
	Bridges	+	+		+	Yes	Yes	Glob.	No	Set
	Cohesion	+				Yes	Yes	Glob.	No	Set
Communities	Dendrogram	+				Yes	Yes	Glob.	No	Graph
	Max-flow min-cut	+				Yes	Yes	Glob.	Edge	Graph
	Clustering	+	+	+		Yes	Yes	Glob.	No	Graph
Centrality Measures	Degree Centrality	+		+		Yes	No	Glob.	No	Set
	Closeness Centrality	+		+		Yes	Yes	Glob.	No	Set
	Betweenness Centrality	+		+		Yes	Yes	Glob.	No	Set
Pattern Matching	Graph/Subgraph Matching	+	+			Yes	Yes	Neigh.	No	Graph
Graph Anonymization	k-degree Anonym.	+		+		Yes	No	Glob.	No	Graph
	k-neighborhood Anonym.	+		+		Yes	Yes	Glob.	No	Graph
Other Operations (Similarity, ranking,...)	Structural Equivalence	+				Yes	Yes	Glob.	No	Graph
	PageRank			+		Yes	No	Glob.	Node	Set

Table 4.2: Graph Operations, Areas of Interest and Categorization

5 Benchmarking RDF databases

The recent increase of the number of real-world applications that use RDF data, is raising the need for more interesting benchmarks that should test all aspects of RDF query processing. As a result, in the last few years we have witnessed the development of various different RDF benchmarks, but no single benchmark has emerged as a standard. One of the reasons for the lack of a widely-accepted RDF benchmark is the fact that, until the recent development of SPARQL 1.1, there was no standard query language supporting some of the query operators that are essential for query processing (such as aggregates, subqueries, path queries, etc).

Benchmarking RDF systems presents different challenges than the ones posed in relational database engines. As a result, existing relational benchmarks are not really suitable for RDF benchmarking. This is due to the intricacies of RDF data, which *a)* are expressed using a simple data model based on the concept of *triple* and *b)* is not necessarily accompanied by a schema.

Nevertheless, existing relational benchmarks have and can be used for testing the performance of RDF engines. Online Transaction Processing (OLTP) benchmarks are not interesting in the RDF world since the large number of indexes build to support efficient querying do not perform well in a situation where updates are prominent. Consequently, the benchmarks that can be used are the business intelligence use cases. TPC-H can be translated into RDF, but the queries and especially the data still remain relational at heart. In addition, the workload in these benchmarks does not take into account *inference* and *graph operations*. The inference imposed by RDFS is essential in RDF benchmarking due to the increasing number of schemas that are used in Linked Open Data. Graph operations such as *path traversals* are also essential for RDF data which are not covered by existing relational benchmarks.

5.1 Benchmarks Using Real Datasets

A number of RDF benchmarks that use real datasets have been proposed over the last years. DBPedia [84] extracts structured information from Wikipedia [140] to create a large dataset for benchmarking. The English version of the DBpedia knowledge base currently describes 3.77 million things, out of which 2.35 million are classified in the DBPedia ontology [86]. The DBPedia ontology is a shallow ontology defined across different domains and contains concepts such as Person (764,000 instances), Creative Works (333,000 instances), Place (573,000 instances), Organizations (192,000) instances) among others. A number of properties are used in the DBPedia dataset including domain specific ones, as well as RDFS properties (rdfs:subClassOf, rdf:label, rdf:type, rdf:comment) and the OWL property owl:sameAs that links resources that refer to the same object but originate from different datasets. The dbpedia:wikiPageWikiLink property is also used in order to link wikipedia pages and is the one that has the most instantiations. WordNet 3.0 [141], that is used in DBPedia, contains 4.6 million triples.

Although the DBPedia dataset is one of the reference datasets for Linked Open Data, there is no clear set of representative queries for it. Recently, the University of Leipzig [51] developed the DBPSB [87] (DBPedia SPARQL Benchmark) benchmark, which created a query workload derived from the DBPedia query logs.

In particular, the process followed for the creation of the benchmark query workload was based on mining the query logs, creating clusters, and analyzing the features of the SPARQL queries that were most frequently used. The most frequently occurring queries were selected as the basis for the benchmark, and covered most of the SPARQL 1.0 language features such as graph pattern constructors (union, optional, join), solution sequences and modifiers (distinct), as well as filter conditions and operators (filter, lang, regex, str). The selected queries considered also different number of triple patterns in order to test the efficiency of the join operator in the tested triple stores. These selected queries were transformed into *templates* by varying one part of the query. The produced query workload consists of mostly simple lookups and does not consider more complex language features such as *inference* or even *decision support* queries.

The UniProt KnowledgeBase (UniProtKB) [64] is a high-quality dataset describing protein sequences and related functional information. It is expressed in RDF and contains approximately 1.6 billion triples. UniProt RDF is one of the central datasets in the bio-medical part of the Linked Open Data initiative. UniProtKB

uses an OWL ontology expressed in a sub-language of OWL-Lite (more expressive than OWL Horst), but still tractable. The dataset consists of instances of classes Resource (for life science resource), Sequence (for amino acid sequences) and Protein (for proteins). The UniProt queries are mainly lookup queries [137] but some are also used to test the reasoning capabilities of RDF databases (i.e., *taxonomic queries*).

The YAGO [143] is a vast knowledge base that integrates statements from Wikipedia, Wordnet, WordNet Domains, Universal WordNet and GeoNames [97]. It describes more than ten million entities (which include persons, organizations and others) and contains more than 120 million facts (triples) about these entities. The original dataset is not accompanied by a set of queries (as in the case of UniProtKB). However, Neumann et. al. provided eight queries for an earlier version of the YAGO ontology, in the context of the benchmark RDF-3X engine [54]. These queries are mostly lookup and join queries that use the SPARQL filter, distinct and count operators; join queries are *path* or *star queries*, the most complex one containing up to ten triple patterns.

The Barton Library Dataset [75] was obtained by converting the Machine-Readable Catalogue (MARC) of the MIT Libraries to RDF and it consists of approximately 45 million triples. Abadi et al. [1] proposed seven simple join queries for this dataset. These queries involve a small number of triple patterns as a representative workload, and employ the SPARQL operatorscount, filter and distinct.

The Linked Sensor dataset [60] contains expressive descriptions of approximately 20 thousand weather stations in the United States. Each weather station reports information such as temperature, visibility, precipitation, pressure, wind speed and humidity. The dataset contains also location attributes (latitude, longitude, and elevation) of each weather station, and uses links to entities in Geonames. The dataset is expressed in RDF and contains close to two billion RDF triples (2010), but is not accompanied by a set of queries.

A benchmark for the evaluation of Spatial Semantic Web Systems was proposed in [43]. The benchmark extends the Lehigh University BenchMark [107] by adding spatial locations to several of the objects that are created, and by introducing new spatial entities. The GeoRSS [98] ontology and the Region Connection Calculus vocabulary [117] are used to represent the spatial data. Moreover, the LUBM data generator was extended to produce spatial data using the Poisson distribution. The workload contains the LUBM queries in addition to twelve queries that target spatial data.

5.2 Benchmarks Using Synthetic Datasets

The Lehigh University Benchmark (LUBM) [107] is intended to evaluate the performance of Semantic Web repositories over a *large data set* that adheres to a university domain ontology. The dataset contains *customizable* and *repeatable* synthetic data, a set of *test* queries, and several *performance metrics*. The Univ-Bench ontology used in LUBM is a relatively small ontology consisting of 43 classes and 32 properties, and is expressed in OWL-Lite, the simplest language of OWL [50]. In addition to the user-defined properties, the ontology includes OWL specific properties such as TransitiveProperty, someValuesFrom restrictions, and intersectionOf. We found that the ontology of LUBM is less expressive than some real ontologies. LUBM data are *extensional* synthetic data that adhere to the aforementioned ontology. The process used to obtain the *synthetic* data controls the *selectivity* and the *output size* for each query. The data obtained from the data generator is repeated and random, and is dictated by a set of parameters such as the minimum and maximum number of instances for the classes in the ontology. As the LUBM data is regular, the benchmark does not explore any of RDF's distinguishing properties, and does not pose any interesting challenges for query optimizers. In addition, the generated graph consists of multiple relatively isolated subgraphs, making it inadequate for testing the performance of join algorithms in the query engine.

The LUBM benchmark consists of *fourteen* mainly extensional lookup and join queries, that do not consider some of the complex SPARQL 1.0 operators such as optional and union or *complex reasoning* (e.g., RDFS and OWL inference). The parameters considered in the query definition were the *(i) query selectivity* (estimated proportion of the class instances involved in the query that satisfy the query criteria), *(ii) complexity* (number of classes and properties considered in the query), *(iii) inference or subsumption* (refers to whether class and property hierarchies are used to achieve the completeness of the answer using the simple inference along the rdfs:subClassOf, rdfs:subPropertyOf and owl:sameAs properties). The performance

metrics considered in the LUBM are *load time*, *repository size*, *query response time*, *query completeness* and *soundness*, and a *combined metric for query performance*. The latter helps users better appreciate the potential trade-off between the query response time, inference capability and overall performance of the system. As a standard benchmark, the LUBM itself has several limitations. Firstly, it covers only part of the inference supported by OWL Lite and OWL DL, so it cannot exactly and completely evaluate an ontology system in terms of inference capability. Second, it only supports *extensional queries*, that is queries that about instance data. Third, the ontologies used in LUBM are of *moderate* size, so they are not stressing the tested systems enough.

The University Ontology Benchmark (UOBM) [48] extends the LUBM in order to tackle complex inference, as well as scalability issues. In contrast with LUBM, UOBM uses both OWL Lite and OWL DL ontologies and covers most of the constructs of the these two sublanguages of OWL (namely, owl:sameAs from OWL Lite, and owl:disjointWith, owl:oneOf and owl:EquivalentClass among others from OWL DL). UOBM ontology contains more classes and properties than LUBM and can generate a larger number of instances. UOBM queries are designed based on two principles: (a) queries should consider *multiple lookups* and *complex joins*, and (b) each query should support *at least a different type of OWL inference*.

SP2Bench [69] is one of the most commonly used benchmarks for evaluating the performance of RDF engines. The benchmark contains both a data generator and a set of queries. The data-generator builds upon DBLP [82] bibliographic schema, which is a simple schema using 8 classes (Article, InProceedings, Proceedings, Conferences etc.) and 22 properties (author, title, etc). The SP2Bench generator produces arbitrarily large datasets by taking into account the constraints that are expressed in the DBLP schema. The produced data is mostly relational-like and hence do not exhibit the intrinsic properties related to real-world RDF data. The benchmark consists of fourteen queries (some of them contain minor modifications of the original ones by removing filters and add explicitly the constraints in the triple patterns), and consider different characteristics such as selectivity, query and output size, and different types of joins. The queries employ different SPARQL 1.0 operators such as optional, union, filter, orderby, bound, and are designed to test the different approaches for SPARQL optimization. The designed queries include long path chains, bushy patterns as well as combinations of both and test also *closed world negation*, through a combination of filter, optional and bound. Joins can be explicit and implicit, the latter defined through the use of the filter operator. SP2Bench queries do not address reasoning tasks.

The Berlin SPARQL Benchmark (BSBM) [10, 76] is built around an e-commerce use case where the schema models the relationships between products and product reviews. Version 3.1 [77] comes with a data generator and a test driver, as well as a set of queries that measure the performance of RDF engines for very large datasets, but do not test the ability of the RDF engines to perform complex reasoning tasks (i.e., RDFS and OWL inference). Moreover, BSBM provides performance metrics together with a set of rules on how to run the benchmark and use the metrics. BSBM dataset is scalable to different sizes based on a scale factor and hence can be used for testing the scalability of the RDF engines.

In order to test and compare the performance of native RDF and relational engines, the BSBM dataset comes in three different representations: one using the RDF triple data model, another using RDF quadruples to encode context through the use of named graphs [17], and a third using a relational data model. The BSBM queries are translated accordingly to take into account the different representations. The query workload illustrates the types of search and navigation patterns that a consumer uses in order to identify a product. BSBM 3.1, explores three use cases in the general e-commerce scenario: explore, explore and update, and business intelligence use cases. The first focuses on read only queries, the second on read and update queries and the last on analytical queries expressed using the SPARQL 1.1. language. The queries are not fixed (as in most RDF benchmarks except LUBM) but parameterized, and they consider all SPARQL 1.0 operators including limit, offset, regex and describe. The explore and update use case considers the query mix of the explore use case and includes two simple update queries (insertion and deletion of triples). The business intelligence use case, although it addresses basic SPARQL 1.1. features in the queries (e.g., aggregation and subqueries), is still primitive in comparison with comparable relational benchmarks like TPC-H and TPC-DS that require high volumes of data and execute queries with a high degree of complexity that give answers to critical business questions.

The performance metrics used by BSBM go beyond the metrics used by other benchmarks (which focus on measuring only the *query performance time*). In particular, the performance metrics used by BSBM are *Query Mixes per Hour (QMpH)*, *Queries per Second (QpS)* and *LoadTime*. The first metric measures the number of complete BSBM query mixes that are answered by the tested system within one hour. The second measures the number of queries of a specific type that were answered by the tested system within a second; this metric essentially aims at testing the different aspects of a query processor. Finally, the last metric measures the time required to load an RDF dataset to the tested system; note that the measured time includes also the time needed by the system to build indexes and generate the necessary statistics to be used for query optimization. The major disadvantage of this benchmark is that the used queries do not examine a lot of data, hence the time measured mostly refers to the query optimization and not to the query execution time. This problem is partially solved by the business intelligence use case.

The Social Intelligence Benchmark (SIB) [121, 61] has been developed at CWI in the context of the LOD2 [106] FP7 project. SIB simulates an RDF backend of a social network site (such as Facebook), in which users and their interactions form a social graph of social activities such as writing posts, posting comments, creating/managing groups, etc. SIB comes with a data generator, a set of queries and a set of metrics. The generator considers a set of parameters to produce the social graph, namely social degree (i.e., number of friends per user), clustering coefficient (the connectivity among the immediate neighbors/friends of a user), "assortativity" coefficient (probability that a node connects to a node with similar degree), and average path length (average of all-pairs-shortest-paths). The dataset scales according to the number of users. The purely synthetic generated data is linked with the RDF datasets from DBpedia in order to exploit the vast amount of information in the DBpedia knowledge base.

The benchmark specification contains three query mixes, namely interactive, update, and analysis, which are expressed in terms of SPARQL 1.1 and use most of its operators and advanced features, such as path expressions and nesting. SIB is an initial attempt at creating a synthetic social graph with interesting features such as realistic data correlations. One of the objectives of SIB is to define queries related to business intelligence and graph analytics that could challenge existing database systems. The benchmark is currently limited to a lookup oriented workload but more complex workloads are planned. The metrics proposed for the benchmark are similar to the ones used in BSBM, namely: (a) query per second (b) query mix per hour and (c) total execution time. In a query mix, each query will be executed a number of times reflecting how popular the query is in a real social network.

5.3 Concluding Remarks

As is obvious by the above discussion, existing benchmarks do not cover the features found in relational analytics benchmarks. The data remains essentially relational (and thus not adequate for RDF benchmarks), it is sparsely interrelated, specified using inexpressive ontology languages and does not exhibit traits specific to RDF or graph workloads. The majority of the queries consider *simple lookups* and *simple joins* involving a small number of triple patterns, whereas complex SPARQL operators such as *optional*, *union* are not extensively used. On the other hand, operators such as count, filter and distinct are used, but these do not add any complexity for query processing. The synthetic SP2Bench benchmark considers queries that return very large result sets, such that the generation of the results dominates the query execution time. None of the above benchmarks extensively consider complex SPARQL operators such as *optional*, *aggregation*, *negation* and *inference*. Moreover, existing benchmarks do not take into account *updates* neither at the schema not at the instance level. Last, the queries are usually small and manually created, whereas the real SPARQL queries are usually produced by applications and are extremely complex.

6 Data Integration Benchmarks

6.1 Benchmarking Instance Matching Systems

The widespread adoption of Semantic Web Technologies and the publication of large interrelated RDF datasets and ontologies in the Web has made the integration of data a crucial task. In this context, instance matching [32] (also referred to as record linkage [47], duplicate detection [29], entity resolution [9], and object identification in the context of databases [56]) refers to identifying instances that correspond to the *same real world object*.

Instance matching systems and methods need to be tested to determine the weak and strong points of a method or system, as well as its overall quality for different use case scenarios. Currently, there has not been one single authoritative benchmark for testing the quality of the instance matching systems and methods; each of the existing approaches/techniques create their benchmarks making the comparison of the quality of the systems even harder. Moreover, it is sometimes important that a matching method works well for a specific application domain or use case, which means that the matching process is more use case driven, whereas in other cases matching methods should be all-rounders, meaning that they should perform well to a large number of applications and domains.

A number of benchmarks have already been proposed to test the performance of instance matching techniques mostly for XML and relational data [145, 53, 44]. These benchmarks do not take into consideration the features inherent to ontologies and the RDF data model (such as the flexibility offered by the absence of a rigid schema, as well as reasoning), and none of them can be used for testing the performance of instance matching techniques for Linked Open Data.

An instance matching benchmark comprises of:

- benchmark dataset(s) that must be associated with a domain of interest in order to be able to have meaningful results that can be interpreted and should consider *value*, *structural* and *logical* modifications; depending on the complexity of the modifications, these can be *simple* or *complex* ones, the latter mostly referring to *structural* and *logical* heterogeneities.

- a *ground truth* or *gold standard* that is used to judge the completeness and soundness of the instance matching approach. The gold standard can be either *a set of instances* that the instance matching process should return or *a set of matching links* (known as *instance alignment*) that identify *similar* instances (i.e., instances that refer to the same real world entity).

- a set of *test cases*, each addressing a different kind of requirements that an instance matching algorithm should test. A test case that is included in all synthetic benchmarks is one that considers that the datasets to be tested are the same.

Figure 6.1 (taken from [32]) shows the types of heterogeneities that instance matching algorithms should address, and, thus, should be considered by benchmarks in order to get a meaningful results. In particular, [32] considered the following types:

Value differences: these include misspellings of the names at both the schema and instance level (e.g., typographical errors), as well as the use of different formats to represent the same kind of information.

Structural heterogeneities: these include changes mostly at the schema level, such as different nesting levels for properties, class and property hierarchies, the use of different aggregation criteria for the representation of properties etc.

Logical heterogeneities: these include instantiation of instances to classes that belong to the same or different explicitly or implicitly defined hierarchies, as well as the specification of implicit values that belong to this category of heterogeneities.

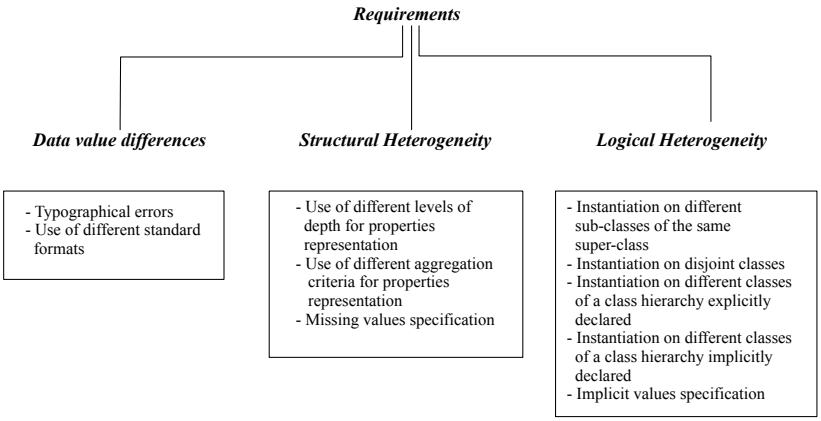

Figure 6.1: Modifications for Instance Matching

6.1.1 Ontology Evaluation Alignment Initiative (OAEI)

The most popular framework for testing ontology matching systems is the one published by the Ontology Alignment Evaluation Initiative (OAEI) [113]. Since 2005, OAEI organizes an annual campaign aiming at evaluating *ontology matching* solutions and technologies using a *fixed set of benchmarks*. In 2009, OAEI introduced the *Instance Matching (IM) Track*, which focuses on the evaluation of different instance matching techniques and tools for RDF and OWL. The track proposed three different benchmarks towards this end: *ARS*, *TSD* and *IIMB* benchmarks [30]. The metrics used to measure the effectiveness of the instance matching tools for each of the benchmarks were the standard metrics of *precision*, *recall* and *f-measure*.

ARS is a benchmark that draws its data from the domain of scientific publications and more specifically from the AKT-Eprints [93], Rexa [118] and SWETO-DBLP [123] datasets. The AKT-Eprints dataset contains information about scientific papers and is produced in the context of the AKT research project [73] and contains 1130 instances. The Rexa dataset covers computer science research literature, people, organizations, venues and research communities, and is more voluminous, containing 18492 instances. Both datasets were obtained from HTML sources and are structured according to the SWETO-DBLP Ontology [124].The SWETO-DBLP dataset, is a publicly available dataset represented as RDF. It describes information about scientific publications from the computer science domain and their authors. Authors are represented as instances of the foaf:Person class (extending the well-known FOAF ontology [95]), and a special class sweto:Publication is defined for publications, with two subclasses sweto:Article and sweto:Article_in_Proceedings for journal and conference publications respectively. The results of the instance matching process are alignments for each pair of datasets in the benchmark, namely AKT-Eprints/Rexa, AKT-Eprints/DBLP-SWETO, and Rexa/DBLP-SWETO.

The TSD benchmark (named after the initials of the datasets TAP [126], SWETO-Testbed [125] and DBPedia [83] respectively) includes these three datasets that cover several topics and are structured according to different ontologies. The TAP dataset contains approximately 180 classes and 64.449 instances.The ontology considered in the SWETO dataset has approximately 200 classes and hundreds of thousands of instances.

DBPedia version 3.2 [83] is a knowledge base that has been extracted from the Wikipedia dumps. It is structured according to a version of the DBPedia Ontology, a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia. The ontology covers over 170 classes, which form a subsumption hierarchy, it has 940 properties and contains approximately 900,000 instances extracted from Wikipedia infoboxes using appropriate mappings. DBPedia 3.2 also contains approximately 2.5 million RDF links to Freebase [94], an open source database which provides data about millions of objects from various domains.

The OAEI 2009 IM track included also the Very Large Crosslingual Resources (VLCR) [139] task, whose purpose was to match the Thesaurus of the Netherlands Institute for Sound and Vision [99] (called GTAA) to two other resources: the English WordNet [92] from Princeton University and DBpedia [85]. GTAA

contains approximately 160.000 terms (about 3800 Subject keywords, 97.000 Persons, 27.000 Names, 14.000 Locations, 113 Genres and 18.000 Makers). The goal of the VLCR task is twofold. The first goal is to match vocabularies in different languages, since a large number of European collections are indexed using different languages; in this respect, it is extremely important to have tools that produce correct alignments between different languages in order to, for instance, enable integrated access to different collections. The second goal is to interlink resources that are rich in semantics but are not rich in structure, which is still the case in the Web. The objective of this task was to test discovering exact and close matches between the vocabulary terms in the involved datasets. The reference alignment was manually produced for a subset of the input data and in contrast to the aforementioned benchmarks, the metrics of *precision* and *recall* were used in the VLCR task to measure the effectiveness of the instance matching techniques and tools.

In addition to these real datasets, the OAEI 2009 IM track used the synthetic ISLab Instance Matching Benchmark [102] (IIMB). The benchmark considers a single dataset from the OKKAM project [112] which is modified using different criteria. The reference ontology is comprised of RDF/OWL data describing actors, sport persons and business firms, and contains approximately 200,000 instances. The modified ontologies (i.e., the target ontologies to which the reference ontology should be matched) are relatively small, and contain 200-2,000 instances. IIMB is organized in 37 different test scenarios, each of which is comprised of the reference ontology (schema and instances), the modified ontology that is obtained by applying a set of simple and complex *value*, *structural* and *logical* modifications, as well as combinations of the above.

The OAEI 2010 Instance Matching track [110] included two new tasks, namely the *Data Interlinking (DI)*, and the *OWL Data* tracks, which are described below. The purpose of the Data Interlinking track was to test the ability of systems to produce links in the Linked Open Data cloud. Five real datasets were used for this task, namely DailyMed [81], Diseasome [89], DrugBank [90], Sider [120] and LinkedMDB [105], all published as linked data. The first four datasets contain information on drugs and their adverse effects, diseases, cheminformatics etc, whereas the last is the first open semantic web database for movies. The Data Interlinking Task was performed *with no a priori knowledge* of the application domain, the datasets or the respective schemas, and tackled the *scalability dimension* of the instance matching problem, since the datasets were as large as DBPedia (hundreds of thousands of instances). For each of the different test cases, a gold standard was provided to test the effectiveness of the instance matching tool and technique. The gold standard was provided in the form of a reference alignment dataset (i.e., a set of links between the reference and the target ontologies).

The OWL Data Task considered the dataset from the IIMB benchmark (described above), as well as the PR dataset, a small dataset that contains data for persons and restaurants. The task was focused on two main goals, namely to provide an evaluation dataset for various kinds of *transformations*, and to cover a wide spectrum of possible techniques and tools.

The main difference of the OWL Data Task with the aforementioned ones, is that some of the transformations require *reasoning* at the level of the ontology in order to find the expected alignments.

The OAEI 2011 Instance Matching Track [111] considered two tasks: the first one considered the IIMB benchmark where the synthetic data were generated from the Freebase dataset. The second considered interlinking the New York Times (NYT), DBPedia, Freebase and GeoNames datasets. The NYT dataset contains three concepts/facets (Person, Organization, Location) and their instances (approximately 10,000 instances of Person, 6,000 instances of Organization and 4,000 instances of Location), as well as an important number of links to the remaining datasets (see Table 6.1) The reference alignments were extracted from the links provided and curated by the New York Times, and subsequently updated to reflect the changes made to the external datasets during the year. The purpose of the second task was to re-build the links within the NYT dataset and to build the links with the external data sources. In particular, several missing links were added, links pointing to non-existing DBPedia instances were removed, and links to instances redirecting to others were updated. As before, the effectiveness of the instance matching systems was measured using the standard metrics of precision and recall.

The OAEI 2012 Instance Matching Track uses the previously mentioned New York Times (NYT) and IIMB datasets for testing instance matching tools. The Sandbox dataset is added to provide examples of some specific matching problems like name spelling and other controlled variations). The objective here is

	Classes		
Statistics	*Person*	*Organization*	*Locations*
Total #sameAs links	14884	8003	8786
Links to Freebase	4979	3044	1920
Links to DBPedia	4977	1949	1920
Links to NYTimes	4979	3044	1920
Links to GeoNames	0	0	1789

Table 6.1: Links from the NYTimes dataset to FreeBase, DBPedia and GeoNames

Simple Modifications	*Complex Modifications*
Misspellings	Expanded/flattened Schema
Insertion/Deletion of Comments	Use of Different Language
Different Data formats	Use of Random names
No use of data types	Use of Synonyms
Overlapping Datasets	Disjoint Data Sets

Table 6.2: Simple and Complex Modifications in ONTOBI

to use the dataset to test those tools that are in an initial phase of their development process and/or for tools that are facing very focused tasks.

Discussion on OAEI

The OAEI 2010 IM track does not tackle the aforementioned weaknesses of the OAEI 2009 IM track. Again, the datasets of IIMB and the PR contained only a few instances (from 400 instances up to 1500 instances). In the case of the DI task, the very large size of the data sources (hundreds of thousands of instances) leads to an error-prone gold standard since it is more or less impossible to construct manually a correct alignment. On the other hand, the OAEI 2011 and the latest 2012 tracks have overcome the problems of the previous years. They include a big amount of data with thousands of instances, while offering a precise and accurate golden standard.

In general the OAEI benchmarks have used both real and synthetic data. The ARS, TDS and VLCR benchmarks from the IM 2009 Track use real datasets, whereas IIMB 2009 uses synthetic datasets. The Data Interlinking Track for OAEI IM 2010 uses real data whereas the PR dataset for the OWL Track employees synthetic data. Last, the NYT dataset considers real data, whereas Freebase is comprised of synthetic datasets. This way, every year the OAEI Instance Matching track takes advantage of both real data and at the same time it constructs synthetic data in order to test specific aspects of RDF instance matching.

The above benchmarks employ the *standard metrics of precision and recall* to measure the performance and quality of the instance matching techniques against a predefined reference alignment (the golden standard). However, the golden standard provided for the VLCR task was manually constructed, and, due to the large size of the tested datasets, it contained several mistakes.

To conclude with, despite the shortcomings of the OAEI IM benchmarks through the years, they are widely known and used for the evaluation of instance matching tools.

6.1.2 ONTOlogy Matching Benchmark With Many Instances (ONTOBI)

The ONTOBI [147, 148] is one of the most recent instance matching benchmarks. The benchmark is organized in 16 different test cases that take into account *simple* and *complex transformations* that are applied on the reference ontology shown in Table 6.2.

The simple modifications that are considered by ONTOBI address mostly *value* and *logical modifications* at the schema and instance levels, whereas *complex* modifications are associated primarily with *structural* heterogeneities.

Simple modifications include *misspellings* of concepts, property names and instance values. They also include changes in the comments of classes. This is important because comments of classes (whenever available) describe the semantics of the class, so this information can be used to detect homonyms or synonyms and consequently help in the process of schema and instance matching. Another simple modification at the schema level is the removal of information related to data types; such information provides additional hints on the semantics of the instance values. In the case of "Disjoint Datasets", the modified reference ontology and the reference ontology contain the non disjoint set of instances (modulo the random assignment of identifiers to instances). This simple test case is used to examine whether the tool examined can discover all matching instances.

Complex structural modifications refer to *expanding* (e.g., adding more structure), or *flattening* (e.g., removing classes/properties or hierarchies) the schema of the ontology.A complex modification may also include changes in the language used to write the class and property names. The use of random names in resources is a complex value modification, much more complex than simply introducing misspellings. An interesting kind of logical modification is the use of synonyms and homonyms for class and property names.

The reference ontology used in ONTOBI is obtained from DBPedia Version 3.4. DBPedia contains a very large schema (205 classes, 1144 object properties and 1024 data types properties). Instances are created by applying the previously used modifications on a small set of instances (13704 instances). OAEI benchmarks and ONTOBI bare many similarities in the modifications of instances. However, in contrast to the OAEI IIMB benchmarks, modifications in ONTOBI affect also the schema.

6.1.3 STBenchmark

STBenchmark [3] is a benchmark that takes as input one reference ontology, and applies several transformations in order to get a modified reference ontology. STBenchmark supports a basic set of mapping scenarios that represent the minimum set of transformations that should be supported by any mapping system. In addition, it contains a generator for instances and mapping scenarios that can be used to produce more complex mapping scenarios, namely *instance copying* with the same or *randomly generated* identifiers, *constant value generation*, *assignment* of instances of a class to different classes and *unnesting* (flattening) or *nesting* of the schema.

The mapping scenario generator, *SGen*, takes as input parameters related to the characteristics of the reference ontology (*schema level only*), and produces a mapping scenario. This means that SGen can be used as the generator of the target schema (since it takes into account transformations also at the schema level). The instance generator *IGen* uses the template-based XML data generator ToxGene [8], takes as input a schema and a set of configuration parameters and returns the set of instances that conform to the input schema obtained from the instances of the original reference ontology. The template created by IGen contains information about the data ranges and the desired vocabulary and is used to randomly (according to a Gaussian distribution) generate data values.

The STBenchmark shows an interesting systematic way of creating different kinds of testbeds for instance matching. STBenchmark employs artificial, randomly created instances with meaningless content, rather than real-world data, which are not that useful for testing all aspects of matching systems, because artificial data follow a more or less strict pattern and make it difficult to completely simulate a human creator. Another disadvantage of the STBenchmark is that no reference alignment is generated, as with ONTOBI and OAEI, so it cannot be disseminated easily and used by many instance matching systems.

6.1.4 Discussion on Instance Matching Benchmarks

The benchmarks presented above are constructed to test matching systems, especially with respect to instance-based matchers. We reviewed them and pointed out their strengths and weaknesses. We summarize here the characteristics of the benchmarks following the aspects introduced in [40]. These are:

	OAEI ARS/ TDS/ VLCR/	**OAEI IIMB**	**OAEI DI**	**STBenchmark**	**ONTOBI**	**OAEI NYT/FreeBase**
systematic procedure	+	+	+	+	+	+
continuity	+	+	+	n/a	n/a	+
quality	+	+	+	n/a	+	+
equity	o	o	+	n/a	+	o
large ontologies	-	o	+	n/a	+	+
schema modifications	-	-	+	o	+	+
instance modifications	+	+	+	o	+	+
dissemination	+	+	+	-	o	+
intelligibility	+	+	+	-	o	+

Table 6.3: Comparison of Instance Matching Benchmarks: "+", fully satisfied, "o" partially satisfied and "-" not satisfied.

- *systematic procedure* according to which the matching tasks are reproducible and the execution has to be comparable
- *continuity*, related to the continuous improvement of the matching tasks improved
- *quality* and *equity*. The first requires that the evaluation rules are exact and the quality of the ontologies is high. The second ensures that no system should be privileged during the evaluation process.
- *dissemination* meaning that the benchmark and the results should be publicly available and other systems have reported them in their evaluations.
- *intelligibility* ensures that the reference alignments and the alignments produced by the systems should be available.

Zaiss et. al. [148] adds four more aspects to the previously mentioned ones to evaluate instance matching systems. The majority of these aspects refer to the *types* of modifications applied to produce the target ontology from the reference ontology. Namely, these are *schema* and *instance modifications*. Table 6.3 summarizes the comparison of the discussed benchmarks according to the aforementioned dimensions.

From Table 6.3 we can see that the OAEI benchmarks satisfy the *systematic procedure*, *continuity*, *quality*, *dissemination* and *intelligibility* requirements. Hence these benchmarks can be eventually adopted as a standard for testing instance matching tools and techniques. STBenchmark and ONTOBI do not satisfy the "continuity" requirement which is ensured only when the benchmarks are used by a number of instance matching systems over the years. ONTOBI is publicly available and consequently partially satisfies the dissemination and intelligibility requirements.

The benchmarks that use DBPedia ontologies (OAEI and ONTOBI) are in general of *good quality*: Wikipedia data is reliable since the instances are assigned to correct concepts and for matching purposes it does not matter if the information is always up-to-date. On the other hand, the STBenchmark is mostly an ontology modifier and consequently it cannot be judged for its quality. *Equity* is ensured only by using different modifications and combinations of the reference ontology to produce the target data such that no type of matching system is favored or disadvantaged. OAEI benchmarks partially satisfy this requirement: instance-based methods are disadvantaged due to the lack of a reasonable amount of instances. Furthermore, the ontologies are quite small and there are no modifications executed at the instance level (except in the case of the IIMB Benchmark).

6.2 Benchmarking RDF Link Discovery Systems

RDF Link Discovery systems aim at discovering relationships between data items within different Linked Data sources. Different data publishers use systems such as LIMES (Linked discovery framework for Metric Spaces) [55] and Silk-Linking Framework [144, 39, 38] for this task; however, neither LIMES nor SILK use the well established instance matching benchmarks discussed earlier to test their performance.

LIMES has developed an efficient matching algorithm that does not take into consideration semantic information that is found in the datasets. An in-house benchmark was used to measure the performance of LIMES which consists of three different test cases: (a) DrugBank (b) SimCities and (c) Diseases. In (a) approximately 4300 DBPedia [84] drug instances were matched against 4700 DrugBank [90] drug instances. The goal of (b) was to detect duplicate cities in a DBPedia subset (approximately 12700 instances). The last experiment (c) considered mapping 23000 instances from the LinkedCT [104] dataset and 5000 instances from MeSH [108] vocabulary. LIMES measures the *time* needed to perform the above linking tasks and does not provide any information on the quality of the obtained result.

SILK is a tool that is used for discovering relationships between data items within different data sources. Similar to LIMES, SILK is evaluated using an in-house benchmark that consists of real datasets: 3100 drug instances of the DBPedia dataset and 4700 drug instances from the DrugBank Knowledge Base. Silk also computes statistical measures pertaining to completeness and correctness of the generated links. Towards this objective, the metrics used for this purpose were *precision*, *recall* and *F-measure*.

LINDA (LINked Data Alignment) [11] performs entity matching in the Web of Data. It is evaluated using the OAEI 2012 Instance Matching Benchmark as well as the augmented Billion Triple Challenge Dataset (BTC). The original BTC 2010 dataset [79] contains approximately 3.1 billion RDF quadruples. The dataset used for the evaluation was enhanced by including all triples from DBpedia dataset (BTC+) resulting to an approximate additional 350 million unique triples, plus Freebase, YAGO and Geonames data (BTC++) resulting to an additional 560 million triples. LINDA was able to identify approximately 2.5 million owl:sameAs links. These datasets are the largest ones used in the evaluation of an instance matching system.

6.3 ETL Benchmarks

Extract **T**ransform and **L**oad (**ETL**) is a database process used in *data warehousing* applications that involves:

- *extracting* data from input sources
- *transforming* this data into a form that can be used by the target system
- *loading* the transformed data into the target system that can be a database or a data warehouse.

Despite the plethora of ETL tools available in the market, there have been only few benchmarking proposals. Currently, there has not been any effort in developing standard benchmarks for ETL processes that take into consideration all the intricacies of RDF data (e.g., absence of schema constraints) as well as some (even simple) form of reasoning. In the following we present briefly TPC-DS [128, 58] and the Linked Open Data Integration Benchmark (LODIB) [65].

6.3.1 TPC-DS

TPC-DS is the decision support benchmark that has been proposed by TPC with main objective to provide a workload that is scalable and considers a wide range of dataset sizes, methodologies and metrics to compare different aspects of ETL systems. This benchmark illustrates decision support systems that (a) consider large volumes of data, (b) answer real-world business questions (c) execute queries of different operational requirements and complexities (d) are characterized by high CPU and IO load and (e) are periodically synchronized with source OLTP databases through database maintenance functions.

TPC-DS is based on the TPC-H [134] and the obsolete now TPC-R [135] and creates a new decision support (DSS) benchmark. More specifically it uses multiple star schemas with shared dimensions and

supports ETL-like data maintenance. The schema is an aggregate of multiple star schemas that contain essential business information. The schemas, when possible, are populated with real data. Although the emphasis is on information analysis, the benchmark recognizes the need to periodically refresh the database a process that corresponds to an ETL task.

To realistically scale the benchmark from small to large datasets, the fact tables of the star schema scale linearly while dimensions scale sub linearly. In a production system environment, the data extraction step may be comprised of numerous separate extract operations, executed against multiple OLTP databases and ancillary data sources. As it is unlikely that the data resides in decision support servers, the data extraction step of the ETL process (E) is assumed and represented in the benchmark in the form of generated flat files from the source databases. The data transformations are expressed as SQL 99 queries (TPC-DS has defined 99 such SQL queries). TPC-DS can be used in a straightforward manner for benchmarking ETL processes for RDF data that conform to the database schema used by the TPC-DS benchmark.

6.3.2 The Linked Open Data Integration Benchmark (LODIB)

The Linked Open Data Integration Benchmark (LODIB) [65] aims to compare both the expressiveness and the runtime performance of data translation systems. It tests data translation systems in the context of Linked Data, by providing a catalogue of 15 data translation patterns, each of which is a data translation problem based on statistics that were derived from 84 examples from the Linked Open Data Cloud. The different translation patterns can be found in [65] and include different types of heterogeneities that can be found in the target models in relation with the source model. The metrics used in LODIB to measure the performance of data translation systems are:

- *Expressivity:* this refers to the number of mapping patterns that can be expressed in a specific data translation system.
- *Performance:* this corresponds to the time needed to translate all source data sets to the target representation.
- *Recall:* this refers to the percentage of matching triples that are present in the expected output.

LODIB consists of three different source datasets that need to be translated by the system under test into a single target vocabulary. These datasets are taken from the e-commerce domain and describe products, reviews, people etc., as in the case of BSBM Benchmarks (see Section 5.2). Each of the translation tasks of these datasets comprises a number of the translation patterns of the benchmark. The benchmark also used a data generator for creating datasets with millions of triples, in order to scale source data.

The strength of the LODI Benchmark is that its datasets are based on a real-world distribution of data translation patterns in the LOD Cloud, which makes the data very close to real data. Moreover, it is strictly use-case driven, based on an e-commerce scenario. On the other hand, LODIB only tests SPARQL construct queries, in contrast to other benchmarks, like the BSBM benchmark, which contains query mixes with various SPARQL select queries.

7 Benchmarking Principles

A benchmark is, generally speaking, a set of tests against which the performance of a system should be measured. Benchmarks or workloads essentially provide a *quantitative* and not a *qualitative* comparison of systems. Benchmarks are used not only to inform users of the strengths and weaknesses of systems, but also to encourage the technology vendors to deal with the drawbacks of their systems and to ameliorate their performance and functionality. In general, each benchmark is defined by [49]:

- the *datasets* to be used
- the *workload* (i.e., *queries*, *test cases*) to run
- the *object* of the benchmark (i.e., whole system or components thereof)
- a set of *metrics* that specify *what* to measure (system performance, energy consumption etc.)
- a set of guidelines on *how* to compare the results (i.e., auditing)

In this report we focus on the *purpose* and *scope* of benchmarks, discuss the *principles* that have been introduced in the literature, and based on the existing ones we propose an extended set of principles that can be used for designing new benchmarks. Three categories of benchmarks have been reported in the literature: *generic* benchmarks and *micro-benchmarks* and *application* benchmarks.

Generic benchmarks [33] are used to provide an *estimate* of the relative performance and price/performance of a system since the cost of implementing a specific application for testing several systems can be prohibitive, since we should consider the intricacies of *every* system to be tested. The results obtained guarantee that the system will never exceed the quoted performance.

No single benchmark can measure the performance of systems for *all applications* since the requirements vary not only by application but also by domain. Each system is designed to solve problems in a single or in related domains and possibly cannot perform other tasks. For instance, database systems and transaction processing software cannot perform data analytics, numerical computations or handle problems that appear in business scenarios. Even, some systems may be strong in handling decision support tasks, other may be strong in transactions. In a different level, widely accepted relational database systems can handle efficiently relational workloads but do not perform equally well for XML or RDF data since each native relational database system is designed to handle the intricacies of the relational data model and the respective query workload.

Such problems are handled by proposing *application-specific* benchmarks. Each such benchmark defines a workload that is characteristic of known and widely used applications in a specific domain of interest. The results obtained by running the application-specific workload on different systems provides an idea of the relative performance of those systems for this domain.

Several application-specific and standard benchmarks have been developed for testing the performance of systems. Standard benchmarks for database applications mimic *real-life scenarios*, are *publicly available* and hence can be used freely by a number of systems, provide a well defined *set of queries* and a set of *metrics* and define *scalable datasets* and workloads [49]. It takes a very long time to create standard benchmarks since wide acceptance is required for standardization. Often, the benchmark is often very large and complicated to run and has a limited dataset and workload variation.

TPC benchmarks [129] fall in this category. They are widely used by vendors of transaction processing and database systems. Systems are judged on the basis of these benchmarks. Several such benchmarks exist in the domain of XML, RDF and graph databases but have not been standardized as in the case of TPC. OO7 [16] is a graph database benchmark. For XML query processing, the MBench [66], XBench [146], XMach-1 [12], XMark [70], X007 [67] and TPoX [136] benchmarks have been defined, with XMark being the one that is the most widely used. In the case of RDF, several benchmarks exist (DBPSB [87], UniProtKB [64], YAGO [143], Barton Library Dataset [75], LUBM [107], UOBM [48], SP2Bench [69], BSBM [10, 76], SIB [121, 61])

but most of them have been proposed and used by academic papers and not by vendors of Semantic Web technologies.

Looking at the TPC and XMark benchmarks, that have been widely used for testing the performance of relational and XML (native or SQL-based) systems, we see that both have a small schema that does not evolve over time to incorporate more interesting use cases. They also define a specific set of queries that can be used to test the systems' performance. TPC has evolved over the years by adding tests for different functionalities, while on the other hand XMark has not evolved to include more complex queries as specified in later versions of XQuery [142] or XML updates. In addition, since the workloads rarely evolve, most of the systems are optimized to perform well for those benchmarks, thereby obtaining results that are not representative of all the possible workloads in the domain.

Micro-benchmarks, as opposed to the generic and application specific benchmarks are used to test a particular aspect of the evaluation process. These can be thought of as a special case of a domain-specific benchmark. Micro-benchmarks are specialized, stand-alone pieces of software that isolate and test one database operator. Such benchmarks are useful for a detailed analysis of the system. The advantages of micro-benchmarks are several [49]:

- they focus on the problem at hand, providing essential information for testing and therefore debugging a specific aspect of a system. The results can then be used to improve this aspect of the system
- they provide a controllable workload and data characteristics. Datasets can be synthetic and/or real and they can scale
- the synthetic data generators consider parameters such as value ranges, data distribution and value correlations
- the set of benchmark queries are of different complexity. For instance, in the case of relational database systems, join queries can be of different forms: path, star queries and even combinations thereof, and involve a different number of tables to be joined.

Micro-benchmarks come also with a number of disadvantages [49] since they cannot be used to understand the impact of local operations to global (system-wide) costs. In the same line, such benchmarks neglect how the small piece of software to be tested is integrated into the system. Micro-benchmarks do not take into consideration real-life applications and consequently the insights that one obtains from them are not directly applicable for real-life applications. Contrary to the standard benchmarks, a micro-benchmark does not come with a set of well defined metrics. It is exactly the lack of such standard metrics that make comparisons across systems unfair, and not representative of the system's performance. In addition, micro-benchmarks do not come with rules or guidelines of how the benchmark should be executed, how the results are interpreted and if they are correct.

There are no standard principles that can be used for defining benchmarks. Jim Gray in his book "The Benchmark Handbook for Database and Transaction Systems" [33] discusses the principles that the domain-specific benchmarks should follow. The benchmark should be:

- *relevant*: it should measure the *peak performance* of systems when performing the typical operations in the problem domain
- *portable*, it should be easy to implement the benchmark on different systems and architectures
- *scalable*: it should test both small and large computer systems. Datasets should scale in order to test larger systems and to answer issues that are raised by new applications in the domain of interest and
- *simple*: meaning that it must be understandable, otherwise it will lack credibility.

Euzenat and Shvaiko in their book "Ontology Matching" [40] present the principles that should guide the evaluation process for an ontology matching system:

- *systematic procedure*
- *continuity*
- *quality* and *equity*
- *dissemination*
- *intelligibility*.

Gray's *portable* and *simple* principles are reflected to the Euzenat and Shvaiko *equity* and *intelligibility* principles respectively.

We adopt and extend the aforementioned guidelines in order to propose a set of principles that benchmarks should follow:

Continuity This principle dictates that the benchmark should evolve in order to incorporate the progress made in the field to include new challenges. For instance, in the case of *query processing*, the query workload should evolve to reflect the features that are added to the (preferably standard) query language used to express the queries.

Quality The datasets and test cases (i.e., queries, workloads) should be of good *quality*. Depending on the problem in hand, good quality may refer to different things:

- *Data* should be expressed using a universal encoding to avoid parsing errors when loading the data in a data or a knowledge base (e.g., UTF-8). For *data integration* tasks, the ontologies that will be used must be large enough and have interesting characteristics in order to reflect real world scenarios.
- *Queries* and *mappings* for *query processing* and *data integration* tasks respectively,
  - should be expressed in a standard query language (SPARQL 1.1 or 1.0 for RDF)
  - should not contain errors
  - be accompanied by a natural language description
  - the formal semantics of the query should agree with the accompanied description
  - should be *concise* and *focused*. Afanasiev and Marx [2], after an analysis of XQuery benchmarks, came to the conclusion that a benchmark query does not add any value to if there is another one that yields the same execution time on all documents and for all engines.

Equity The datasets and workloads to be used *should not be biased* towards a specific system (i.e., query engine, ETL tool, instance matching algorithm etc.). but should be driven only by the tasks to be solved.

Scalability Datasets (either real or synthetic) should scale in order to test different systems. For query processing, queries should return results of different sizes thereby testing the limitations of the query engine. Moreover, the query response times of all queries in the workload as data scales should evolve in the same way. In general, systems should not exhibit a behavior where for different data sizes, different queries of a workload dominate over others, hence having different benchmarks for different data sizes.

Representativity The selected (real) or generated (synthetic) datasets should mimic real world data from the domain of interest. In the case of RDF data, generated datasets should consider characteristics regarding in-degree and out-degree (e.g., power law distributions for RDF data [28]), data and value distributions as well as value correlations.

Benchmark queries must be *representative* of the query language. More specifically, they should cover a large fragment of the *functionality* supported by the language [2]. Features of the language that are

not supported by existing engines should also be included in the benchmark in order to push the system vendors to implement the full functionality of the language. It is only in this manner that vendors will be able to develop systems that advance the state of the art in the field. The benchmark should include queries of different complexity.

Dissemination The benchmark along with the results from different engines, should be *publicly* and *freely available* and *disseminated* in order to be used by the different stakeholders from both academia and industry. In this way the benchmark will be more easily adopted and become a standard.

Intelligibility dictates that the benchmark must specify how the results are represented so that they could be easily analyzed and understood by everyone. The benchmark should also provide *reference* results that can be used to verify the *correctness* of the system. In the case of *instance matching* and *ontology matching* systems reference alignments are provided with the benchmarks. The queries that are considered in a workload should have fixed and predictable characteristics, even if those are parameterized queries. Otherwise, the same query with different parameters many mean different things and need different evaluation strategies, making the interpretation of results impossible.

8 Conclusions

Well-defined and good quality benchmarks are important for comparing the performance of products and systems along different dimensions and for different application needs. They provide vendors and users with an objective and reliable assessment of the behaviour of such products/systems in real-life situations and uncover useful insights related to their limitations. The present deliverable discusses state of the art benchmarks for (a) core database functionalities, (b) graph databases, (c) RDF databases, and (d) RDF data integration approaches (the latter covering both instance matching and ETL techniques).

For core database functionalities, we present the TPC family of benchmarks that deal with transaction processing, querying, decision support, data maintenance and other functionalities of a relational database. Even though TPC benchmarks have been widely used for testing the performance of relational systems, they are generally not suitable for RDF and graph databases due to the different nature of data. Thus, our survey included also specialized benchmarks for RDF and graph databases, and identified the strengths and limitations thereof. From our study, we concluded that benchmarks for RDF and graph databases mainly spur from efforts from researchers and academia and constitute isolated efforts; moreover, they are at their early stage since there is no standard graph data model or query language. RDF benchmarks are more mature than graph database benchmarks. Regarding instance matching benchmarks, our survey showed that the majority have been developed in the context of the Ontology Alignment Evaluation Initiative (OAEI). Even though data integration benchmarks are generally mature and of good quality, many of them still have shortcomings, as described in Chapter 6 of this deliverable.

In the last part of the deliverable we discussed a number of benchmarking principles that should be followed in the development of new benchmarks. In particular, we identified and studied various types of benchmarks, such as generic benchmarks or micro-benchmarks, explained their purpose and scope, and outlined the principles that have been proposed in the literature.

The survey performed in the context of this deliverable uncovered ideas, practices, techniques and solutions that can be reused for the envisioned suite of LDBC benchmarks. Moreover, it allowed the identification of shortcomings and limitations of existing benchmarks, so that the benchmarks proposed by the LDBC consortium can properly focus on the challenges imposed by the new applications. A preliminary outcome of this effort is the set of principles to which new benchmarking efforts should adhere, which was outlined in Chapter 7.

REFERENCES

[1] Daniel J. Abadi, Adam Marcus, and Barton Data. Scalable semantic web data management using vertical partitioning. In *VLDB*, 2007.

[2] L. Afanasiev and M. Marx. An analysis of XQuery benchmarks. *Information Systems*, 33, 2008.

[3] B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: Towards a benchmark for mapping systems. In *PVLDB*, 2008.

[4] B. Amann and M. Scholl. Gram: a graph data model and query language. In *ACM Hypertext*, 1992.

[5] R. Angles Rojas and C. Gutiérrez. Survey of graph database models. *ACM Computing Surveys*, 40(1), 2008.

[6] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[7] D. Bader, J. Feo, J. Gilbert, J. Kepner, D. Koetser, E. Loh, K. Madduri, B. Mann, T. Meuse, and E. Robinson. Hpc scalable graph analysis benchmark v1.0. `http://www.graphanalysis.org/benchmark/GraphAnalysisBenchmark-v1.0.pdf`, February 2009.

[8] D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. Lyons. ToXgene: an extensible template-based data generator for XML. In *WebDB*, 2002.

[9] I. Bhattacharya and L. Getoor. *Entity resolution in graphs. Mining Graph Data*. Wiley and Sons, 2006.

[10] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web and Inf. Sy.*, 5(2), 2009.

[11] C. Boehm, G. de Melo, F. Naumann, and G. Weikum. LINDA: distributed web-of-data-scale entity matching. In *CIKM*, 2012.

[12] T. Bohme and E. Rahm. XMach-1: A Multi-User Benchmark for XML Data Management. In *Efficiency and Effectiveness of XML Tools, and Techniques Workshop (EEXT)*, 2003. In conjunction with VLDB.

[13] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. `www.w3.org/TR/2004/REC-rdf-schema-20040210`, 2004.

[14] J. Broekstra and A. Kampman. SeRQL: An RDF Query and Transformation Language. http://www.cs.vu.nl/ jbroeks/papers/SeRQL.pdf, 2004.

[15] M. Carey, D. DeWitt, and J. Naughton. The OO7 benchmark. In *SIGMOD*, 1993.

[16] M. J. Carey, D. J. DeWitt, and J. E. Naughton. The 007 Benchmark. In *SIGMOD*, 1993.

[17] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named Graphs. *Journal of Web Semantics*, 3(3), 2005.

[18] R. Cattell and J. Skeen. Object operations benchmark. *ACM TODS*, 17(1), 1992.

[19] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *SDM*, 2004.

[20] M. Ciglan, A. Averbuch, and L. Hluchy. Benchmarking Traversal Operations over Graph Databases. In *ICDEW*, 2012.

[21] Marek Ciglan and Kjetil Nørvåg. Sgdb–simple graph database optimized for activation spreading computation. In *Database Systems for Advanced Applications*, pages 45–56. Springer, 2010.

[22] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, and Florian Waas. The mixed workload CH-benCHmark. In *DBTest*, 2011.

[23] M.P. Consens and A.O. Mendelzon. Expressing structural hypertext queries in graphlog. In *ACM Hypertext*, 1989.

[24] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[25] Miyuru Dayarathna and Toyotaro Suzumura. Xgdbench: A benchmarking platform for graph stores in exascale clouds. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 363–370. IEEE, 2012.

[26] D. Dominguez-Sal, N. MartÃČÂŋnez-BazÃČÂąn, P. Baleta V. MuntÃČÂľs-Mulero, and J.L. Larriba-Pey. A Discussion on the Design of Graph Database Benchmarks. In *TPC Technology Conference (TPCTC)*, 2010.

[27] D. Dominguez-Sal, P. UrbÃČÂşn-Bayes, A. GimÃČÂľnez-VaÃČÂśÃČÂş, S. GÃČÂşmez-Villamor, N. MartÃČÂŋnez-BazÃČÂąn, and J.L. Larriba-Pey. Survey of graph database performance on the HPC scalable graph analysis benchmark. In *IWGD*, 2010.

[28] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets. In *SIGMOD*, 2011.

[29] A. K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1), 2007.

[30] J. Euzenat, A. Ferrara, L. Hollink, A. Isaac, C. Joslyn, V. Malaise, C. Meilicken, A. Nikolov, J. Pane, M. Sabou, F. Scharffe, P. Shvaiko, V. Spiliopoulos H., Stuckenschmidt, O. Svab-Zamazal, V. Svatek, , C. Trojahn, G. Vouros, and S. Wang. Results of the Ontology Alignment Evaluation Initiative 2009. Technical report, Universita Degli Studi di Trento, 2010.

[31] B. McBride F. Manola, E. Miller. RDF Primer. `www.w3.org/TR/rdf-primer`, February 2004.

[32] A. Ferrara, D. Lorusso, S. Montanelli, and G. Varese. Towards a Benchmark for Instance Matching. In *OM*, 2008.

[33] J. Gray, editor. *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann, 1993.

[34] R.H. Güting. Graphdb: Modeling and querying graphs in databases. In *VLDB*, 1994.

[35] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. `http://www.w3.org/TR/sparql11-query/`, November 2012. W3C Proposed Recommendation.

[36] H. He and A.K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *SIGMOD*, 2008.

[37] J. Hidders and J. Paredaens. GOAL, A graph-based object and association language. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.6465, 1993.

[38] R. Isele, A. Jentzsch, and C. Bizer. Silk Server - Adding missing Links while consuming Linked Data. In *COLD*, 2010.

[39] A. Jentzsch, R. Isele, and C. Bizer. Silk: Generating RDF Links while publishing or consuming Linked Data. In *ISWC*, 2010. Poster.

[40] J.Euzenat and P. Shvaiko, editors. *Ontology Matching*. Springer-Verlag, 2007.

[41] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *WWW*, 2002.

[42] Myunghwan Kim and Jure Leskovec. Multiplicative attribute graph model of real-world networks. *Internet Mathematics*, 8(1-2):113–160, 2012.

[43] D. Kolas. A benchmark for spatial semantic web systems. Available at `http://filebox.vt.edu/users/dkolas/public/ssbm/`.

[44] H. KÃűpcke, A. Thor, and E. Rahm. Comparative evaluation of entity resolution approaches with FEVER. In *VLDB*, 2009. Demo Track.

[45] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[46] M. Levene and G. Loizou. A graph-based data model and its ramifications. *IEEE Transactions on Knowledge and Data Engineering*, 7(5), 1995.

[47] C. Li, L. Jin, and S. Mehrotra. Supporting efficient record linkage for large data sets using mapping techniques. In *WWW*, 2006.

[48] L. Ma, Y. Yang, Z. Qiu, G, Xie, Y. Pan, and S. Liu. Towards a Complete OWL Ontology Benchmark. In *ESWC*, 2006.

[49] Stefan Manegold and Ioana Manolescu. Performance evaluation in database research: principles and experience. In *EDBT*, 2009. Tutorial.

[50] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language. `http://www.w3.org/TR/owl-features/`, 2004.

[51] M. Morsey, J. Lehmann, S. Auer, and A-C. N. Ngomo. DBpedia SPARQL Benchmark - Performance assessment with real queries on real data. In *ISWC*, 2011.

[52] R.C. Murphy, K.B. Wheeler, B.W. Barrett, and J.A. Ang. Introducing the graph 500. cray user's group (cug), 2010.

[53] M. Neiling, S. Jurk, H.-J. Lenz, and F. F. Naumann. Object identification quality. In *DQCIS*, 2003.

[54] Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1), 2010.

[55] A.-C. Ngonga Ngomo and Soren Auer. LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. *IJCAI*, 2011.

[56] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt. Leveraging Terminological Structure for Object Reconciliation. In *ESWC*, 2010.

[57] P. O'Neil, B. O'Neil, and X. Chen. Star Schema Benchmark (Revision 3, June 5, 2009). Technical report, UMass/Boston, 2009.

[58] R. Othayoth and M. Poess. The Making of TPC-DS. In *VLDB*, 2006.

[59] J. Paredaens, P. Peelman, and L. Tanca. G-log: A graph-based query language. *IEEE Transactions on Knowledge and Data Engineering*, 7(3), 1995.

[60] H. Patni, C. Henson, and A. Sheth. Linked sensor data. In *CTS*, 2010.

[61] M-D. Pham, P.A. Boncz, and O. Erling. S3G2: a Scalable Structure-correlated Social Graph Generator. In *TPCTC*, 2012.

[62] A. Poulovassilis and S.G. Hild. Hyperlog: A graph-based system for database browsing, querying, and update. *IEEE Transactions on Knowledge and Data Engineering*, 13(2), 2001.

[63] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. `www.w3.org/TR/rdf-sparql-query`, January 2008.

[64] N. Redaschi and UniProt Consortium. UniProt in RDF: Tackling Data Integration and Distributed Annotation with the Semantic Web. In *Biocuration Conference*, 2009.

[65] C. R. Rivero, A. Schultz, and C. Bizer. Benchmarking the Performance of Linked Data Translation Systems. In *Linked Data On the Web (LDOW)*, 2012.

[66] K. Runapongsa, J. M. Patel, H. V. Jagadish, Y. Chen, and S. Al-Khalifa. The Michigan Benchmark: Towards XML Query Performance Diagnostics. In *VLDB*, 2003.

[67] S. Bressan and M. Li Lee and Y. Guang Li and Z. Lacroix and U. Nambiar. The XOO7 Benchmark. In *Efficiency and Effectiveness of XML Tools, and Techniques Workshop (EEXT)*, 2002. In conjunction with VLDB.

[68] S. Sakr, S. Elnikety, and Y. He. G-SPARQL: A Hybrid Engine for Querying Large Attributed Graphs. In *CIKM*, 2012.

[69] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. XMARK: A benchmark for xml data management. In *VLDB*, 2002.

[70] A. Schmidt, F. Wass, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *VLDB*, 2002.

[71] A. Seaborne. RDQL - A query Language for RDF. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/, January 2004. W3C Member Submission 9.

[72] M. Sintek and S. Decker. Triple: A Query, Inference and Transformation Language for the Semantic Web. In *Proc. of Deductive Databases and Knowledge Management (DDLP)*, 2001.

[73] AKT Project. `http://www.aktors.org/akt/`.

[74] AllegroGraph. http://www.franz.com/agraph/allegrograph/.

[75] Barton Dataset. `http://simile.mit.edu/wiki/Dataset:_Barton`.

[76] Berlin SPARQL Benchmark (BSBM). `http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/`.

[77] Berlin SPARQL Benchmark (BSBM) Specification - V3.1. `http://wifo5-03.informatik.unimannheim.de/bizer/berlinsparqlbenchmark/spec/index.html`.

[78] TinkerPop Blueprints. https://github.com/tinkerpop/blueprints/wiki.

[79] Billion Triple Challenge Dataset. `http://km.aifb.kit.edu/projects/btc-2010/`.

[80] Neo4j cypher documentation. `http://docs.neo4j.org/chunked/milestone/cypher-query-lang.html`.

[81] Dailymed. `http://datahub.io/dataset/fu-berlin-dailymed`.

[82] The DBLP Computer Science Bibliography. http://www.informatik.uni-trier.de/~ley/db/.

[83] Dbpedia 3.2. http://blog.dbpedia.org/2008/11/17/dbpedia-version-32-released-including-thenew-

[84] Dbpedia. http://dbpedia.org/sparql.

[85] Dbpedia. http://oaei.ontologymatching.org/2009/vlcr/#dbpedia.

[86] The DBPedia Ontology. http://wiki.dbpedia.org/Ontology?v=194q.

[87] DBSB. http://aksw.org/Projects/DBPSB.

[88] DEX. http://www.sparsity-technologies.com/dex.

[89] Diseasome. http://diseasome.kobic.re.kr/.

[90] Drugbank. http://www.drugbank.ca/.

[91] Dublin Core. http://dublincore.org/.

[92] Englishwordnet. http://oaei.ontologymatching.org/2009/vlcr/#wordnet.

[93] Eprints. http://eprints.aktors.org/.

[94] Freebase. http://www.freebase.com/.

[95] The Friend of a Friend Project. http://www.foaf-project.org/.

[96] Apache Fuseki. http://jena.apache.org/documentation/serving data.

[97] Geonames. http://www.geonames.org/.

[98] GeoRSS. http://georss.org/Main_Page.

[99] Gtaa. http://datahub.io/dataset/gemeenschappelijke-thesaurus-audiovisuele-archieven.

[100] HypergraphDB. http://www.hypergraphdb.org.

[101] Infinitegraph. http://objectivity.com/INFINITEGRAPH.

[102] ISLAB, Instance Matching Benchmark. http://islab.dico.unimi.it/iimb/.

[103] Jena. http://jena.apache.org/.

[104] Linkedct. http://linkedct.org/about/.

[105] Linkedmdb. http://datahub.io/dataset/linkedmdb.

[106] LOD2: Creating Knowledge out of Linked Data. http://lod2.eu/.

[107] LUBM. http://swat.cse.lehigh.edu/projects/lubm.

[108] Medical Subject Headings. http://www.nlm.nih.gov/mesh/.

[109] Neo4J. http://neo4j.org.

[110] Oaei instance matching. http://oaei.ontologymatching.org/2010/, 2010.

[111] Oaei instance matching. http://www.instancematching.org/oaei/imei2011.html, 2011.

[112] OKKAM Project. http://www.okkam.org/.

[113] Ontology Alignment Evaluation Initiative. http://oaei.ontologymatching.org/.

[114] Open Directory Project. http://www.dmoz.org/.

[115] OpenRDF. http://www.openrdf.org.

[116] OrientDB. http://www.orientechnologies.com/orient-db.htm.

[117] Region Connection Calculus vocabulary. http://en.wikipedia.org/wiki/Region_connection_calculus.

[118] Rexa. http://www.rexa.info/about.

[119] RSS. http://www.rssboard.org/.

[120] Sider. http://datahub.io/dataset/fu-berlin-sider.

[121] Social Network Intelligence BenchMark. http://www.w3.org/wiki/Social_Network_Intelligence_BenchMark.

[122] SQL.org. http://www.sql.org/.

[123] Sweto-dblp. http://datahub.io/dataset/sweto-dblp.

[124] Sweto-dblp ontology. Available at http://lsdis.cs.uga.edu/projects/semdis/swetodblp/august2007/opus_august2007.rdf.

[125] SWETO-testbed. http://lsdis.cs.uga.edu/projects/semdis/sweto/.

[126] Tap. http://tap.stanford.edu/.

[127] tinkerpop / tinkubator. https://github.com/tinkerpop/tinkubator/tree/master/graphdb-bench.

[128] TPC BenchmarkâĎćDS (TPC-DS): The New Decision Support Benchmark Standard. http://www.tpc.org/tpcds/.

[129] Transcation Processing Performance Council (TPC): TPC Benchmark. http://www.tpc.org.

[130] Tpc-c. http://www.tpc.org/tpcc/default.asp.

[131] Tpc-d. http://www.tpc.org/tpcd/default.asp.

[132] Tpc-ds. http://www.tpc.org/tpcds/default.asp.

[133] Tpc-e. http://www.tpc.org/tpce/default.asp.

[134] Tpc-h. http://www.tpc.org/tpch/default.asp.

[135] Tpc-r. http://www.tpc.org/tpcr/default.asp.

[136] Transaction Processing over XML (TPoX). http://tpox.sourceforge.net/.

[137] UniProtKB Queries. urlhttp://www.uniprot.org/help/query-fields.

[138] Uri. http://www.w3.org/TR/uri-clarification/.

[139] Vlcr. http://oaei.ontologymatching.org/2009/vlcr/.

[140] Wikipedia. The Free Encyclopedia. http://en.wikipedia.org/wiki/Wikipedia.

[141] Wordnet 3.0 in RDF. `http://semanticweb.cs.vu.nl/lod/wn30/`.

[142] XQuery 1.0: An XML Query Language (Second Edition). http://www.w3.org/TR/xquery/, 2010. W3C Recommendation.

[143] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[144] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and Maintaining Links on the Web of Data. In *ISWC*, 2009.

[145] M. Weis, F. Naumann, and F. Brosy. A Duplicate Detection Benchmark for XML and Relational Data. In *IQIS*, 2006.

[146] B. Bin Yao, M. T. ÃŰzsu, and N. Khandelwal. XBench Benchmark and Performance Testing of XML DBMSs. In *ICDE*, 2004.

[147] K. Zaiss, S. Conrad, and S. Vater. A Benchmark for Testing Instance-Based Ontology Matching Methods. In *KMIS*, 2010.

[148] Katrin Simone Zaiss. *Instance-Based Ontology Matching and the Evaluation of Matching Systems*. PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultaet der Heinrich-Heine-Universitaet Dusseldorf, 2010.