Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Master in Data Science

Master Thesis

SPARQL-based Linking of Knowledge Graphs

Author: Wenqi Jiang
Tutor: Raul Garcia Castro and Andrea Cimmino Arriaga

Madrid, July, 2022

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Master Thesis
Master in Data Science

Title: SPARQL-based Linking of Knowledge Graphs
July, 2022

Author:	Wenqi Jiang
Supervisor:	Raul Garcia Castro
	Inteligencia Artificial
	ETSI Informáticos
	Universidad Politécnica de Madrid
Supervisor:	Andrea Cimmino Arriaga
	Sistemas Informáticos
	ETSI Sistemas Informáticos
	Universidad Politécnica de Madrid

Abstract

Nowadays, both traditional and emerging enterprises rely heavily on Internet data, yet these data are disorganized and scattered in different data sources all over the world. In most of such data, the models may be different, but their contents may be similar, to utilize these data more efficiently, it is imperative to link these isolated data to create a unified view. Link discovery has always been a trending problem, and many companies tried to solve it. However, many of the solutions from these companies are not based on a common standard and thus result in a steep learning curve.

This thesis focuses on the specification and development of link discovery proposal based on standard SPARQL, as well as, the implementation of several strings similarity metrics used for the linking. Specifically, it is a Resource Description Framework (RDF) linking task that produces connections between RDF resources from different or the same data sources, such as DBpedia, Wikidata and BNE datos. RDF linking is based on one or more link rules that specify the conditions that two RDF resources can be considered the same, such as text similarity or geographical position. SPARQL Protocol and RDF Query Language (SPARQL) is one of the most widely used and standard query language and protocol for Linked Open Data on the web or RDF triplestores. So, this TFM featured a SPARQL-based application to link knowledge graphs using Apache Jena as the SPARQL engine, a free and open-source Java framework for building Semantic Web and Linked Data applications. This TFM used multiple linking rules and evaluated the results with several data sets from the Ontology Alignment Evaluation Initiative (OAEI) competition in 2021 and achieved excellent results.

Keywords— RDF, SPARQL, Linking algorithm, Apache Jena, OAEI

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter is the introduction about this work, and focuses on the current situation and the problems encountered. It has two parts, as follows: section §1.1 is the context of this research work; section §1.2 presents the possible problems in linking these ontologies. This chapter presents a comprehensive grasp of the broad background of this project, why to link ontologies, and how to achieve the goal.

1.1 Research Context

This is the era of big data, any emerging firms rely on the availability and interoperability of appropriate online data sets. Huge amounts of data are generated every day, and most of them are heterogeneous in terms of datasets, formats, etc. So, it is hard to machine to read and analyze. That is the reason scientists and research institutions have come up with the concept of a Web of Data, whose goal is to make Internet data machine-readable. The Web of Data gives us with isolated data that may be exploited transparently as necessary by business models. The Web of Data bases on the Linked-Data principles, which hold that resources inside various data sets that relate to the same real-world entity must be linked to allow data interoperability. It helps to store, manage and share knowledge more efficiently so that computer can use such resources without human effort on parsing the data. On the Web of Data, the most common structure is The Resource Description Framework (RDF), which is the standard data model representation for knowledge on the Web of Data. RDF is a quite effective way to process and link data, even if the ontologies are different. It uses Internationalized Resource Identifier (IRI) to extend the graph, which is a directed, labelled graph, and its edge represent named links between two resources.

The number of data sets published on the web is constantly increasing (e.g., LinkedGeoData with more than 30 billion triples). Furthermore, the Linked Data paradigm bases on the unrestricted dissemination of information by many publishers, as well as the interlinking of Web resources across knowledge bases. It is easy to understand that there are many redundant and duplicate RDFs here; to make the Web of Data more efficient and succinct, it must link RDFs together according to certain rules, which is this project's purpose.

1.2 The Linking Task

The linking task contains two parts of the process of linking two data sets: the first step generates link rules that describe whether two resources should be linked, and the second phase focuses on applying these rules effectively. The contents of generating link rules include choosing transformation and similarity metrics, which are used in determining whether two resources should be linked based on their literal data attributes; if they are linked, the input resources will link together; otherwise, they remain separate. Actually, in most circumstances, linking RDFs in cross-data sets is quite difficult, and it must rely on Instance Matching (IM) and Link Discovery technologies to do so. Indeed, a few engines, such as Limes and Silk, have achieved this goal. However, because the vast majority of them rely on their own link rules languages, using the same link rules in two or more engines is impossible. As a result, users are finding it difficult to use these implementations at the same time, but RDF linking is a fairly common task that must be finished. Furthermore, a Cartesian product of the resources of two data sets will be performed in some situations during the comparison of the data under the specific link rules. It needs to compare all the resource pairings to determine if they should be linked. Thus, RDF linking is a computationally intensive job because it requires comparing all resources from one dataset with all resources from another. In other words, for each RDF resource, one or more data comparisons must be performed. So, the way in which link rules are expressed, built, and processed has a high impact on the performance results, just like the language and the strategy used, which can set restrictions to reduce the number of RDF resources to be compared.

For this reason, this project created a more general way to link ontologies based on SPARQL by implementing functions in Apache Jena, which is a SPARQL query engine. So, practitioners don't need to learn another language to link ontologies, and using the SPARQL query is enough.

Chapter 2

State of the Art

The focus of this chapter is to present modern technologies that are relevant to the mission objectives, but not to provide a comprehensive solution to the problem of implementation, which would give a clearer idea in advance of how the project will be implemented at this stage of its development. The section §2.1 RDF and RDF Schema presents a standard model for data interchange on the Web, the section §2.2 SPARQL presents a query language for such RDF models. The most important section is section §2.3 Apache Jena, which introduces a query engine that supports the SPARQL language and the custom functions. In addition, this section describes how to implement the project's ARQ functions, which can be used to link different ontologies.

This chapter will provide a clear image of how information is represented in the real world and how to execute simple operations on these ontologies, and show the basis for the project development in the next chapter.

2.1 RDF and RDF Schema

The Resource Description Framework (RDF) data model is discussed in this section of the state of the art. This section shall discuss its history, definition, constituent parts, and syntax. Here is an example to help illustrate how to interpret it. Similarly, it presents the idea of the knowledge graph, which will help to go further into the concept of the evaluation section.

RDF is a standard model for data transmission and a framework for describing information on the Web of data[1]. RDF was certified as a recommended standard by the World Wide Web Consortium (W3C) in 1999, and the most recent version of the specification, RDF 1.1, was released in 2014. While originally intended to be used for metadata, RDF has evolved into a generic machine-readable data format that can be used to describe and communicate graph data as a whole. RDF contains capabilities that make data merging easier even when the underlying schemas are different, and it permits schema evolution over time without needing all data consumers to be updated. Using this simple model to express descriptions of resources allows structured or semi-structured data to be mixed, exposed, and shared across different sources. Furthermore, because RDF is a simple data format that can be used to express almost all abstract ideas and resources, it is increasingly being used in some management systems that have nothing to do with the semantic web.

The subject, predicate, and object are all that are required in the RDF schema, which are also known as **Triples**. A "triple" in RDF refers to a linking structure that uses URIs to identify not only the link between entities, but also the two endpoints of the link. **IRIs**, **blank nodes**, or **datatyped literals** are used as elements in RDF graphs, which are sets of subject-predicate-object triples. RDF datasets organize collections of RDF graphs, where the nodes represent the resources and the edges represent the named links between two resources. The graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations. The visual representation of triples is an arc and node diagram. The resources are shown as oval nodes, the predicate as arcs, and the objects are depicted as rectangles. A simple example of a graph view is shown below:

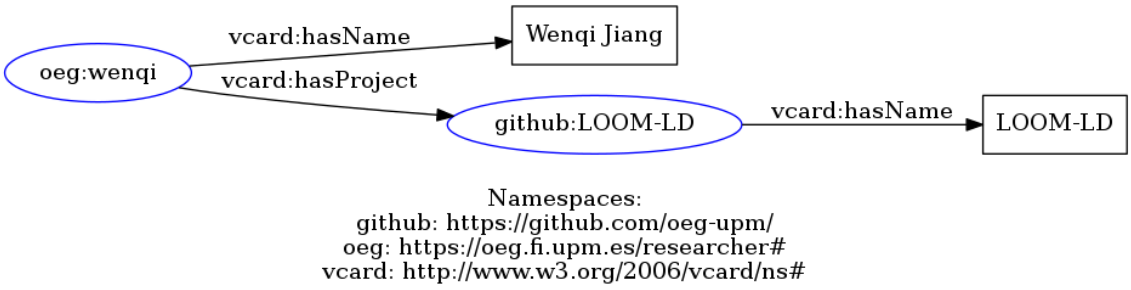

Figure 2.1: A Basic RDF Triples

Figure 2.1 shows a simple example of the statement "Wenqi has a GitHub project named LOOM-LD", where "Wenqi" is the subject, "hasProject" is the predicate, and "LOOM-LD" is the object. Arrows always go from subject to object, so the direction of the arrow is significant. The above example can also be read as "A GitHub project LOOM-LD belongs to Wenqi." There are many URLs in the picture above; these are really Internationalized Resource Identifiers (IRIs), which are a generalization of URIs that may describe almost all resources on the Internet. Notice that both the subject and the predicate must be IRIs; only the object of a statement can be a literal string or other data type. In addition to RDF/XML and N-Triples, there are also Turtle, JSON-LD and N-Quads and other data serialization forms provided by RDF. Turtle is now the most extensively used. So, the example above can be serialized to Turtle as follows:

Listing 2.1: A Basic RDF

```
@prefix github: <https://github.com/oeg-upm/> .
@prefix oeg:    <https://oeg.fi.upm.es/researcher#> .
@prefix vcard:  <http://www.w3.org/2006/vcard/ns#> .

oeg:wenqi  vcard:hasName  "Wenqi Jiang" ;
        vcard:hasProject  github:LOOM-LD .

github:LOOM-LD  vcard:hasName  "LOOM-LD" .
```

2.2 SPARQL Query Language for RDF

SPARQL is a query language for retrieving and processing RDF data. It is a basic technology of the semantic web and a W3C standard. SPARQL 1.0 was endorsed by the W3C on January 15, 2008, and SPARQL 1.1 was released in March 2013. This section gives a short summary of the most important SPARQL features that are important to the work that is talked about in this thesis.

Just like SQL allows users to retrieve and modify data in a relational database, SPARQL provides the same functionality for NoSQL graph databases like Ontotext's GraphDB. SPARQL allows users to write queries against what can loosely be called "key-value" data or, more specifically, data that follows the RDF specification of the W3C. SPARQL can be used to express queries across diverse data sources, whether the data stores natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by the source RDF graph. The results of SPARQL queries can be result sets or RDF graphs. The Listing 2.2 below shows a simple SPARQL query example that gets the names of all researchers and the GitHub projects they are working on.

Listing 2.2: A Basic SPARQL

```
PREFIX oeg: <https://oeg.fi.upm.es/researcher#>
PREFIX github: <https://github.com/oeg-upm/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>

SELECT ?name ?projectName WHERE {
  ?researcher vcard:hasProject ?project;
    vcard:hasName ?name.
  ?project vcard:hasName ?projectName.
}
```

This format is very similar to the well-known SQL. From a syntactic perspective, the general structure of the language is the same as SQL, with the same three main building blocks:

- A SELECT clause, which says how the results will be given back to the user. SPARQL, unlike SQL, return data in different ways, like the table result with SELECT, the graph result with DESCRIBE or CONSTRUCT, or the boolean answer with ASK.
- A WHERE clause is constructed from a graph pattern. Informally, this sentence is represented by a pattern corresponding to an RDF network in which certain resources have been substituted by variables. In addition, more complicated expressions (patterns) that are created using algebraic operators are also permitted. This pattern is used to filter the returned values of the dataset.
- A FROM clause, which specifies the sources or endpoints to be queried.

The table 2.1 shows the result of the above simple SPARQL query, which is all the researchers' names and project names.

name	projectName
"Wenqi Jiang"	"LOOM-LD"

Table 2.1: The Basic SPARQL Result

There is a wide variety of graph patterns that can be matched through SPARQL queries, which reflects the variety of the data that SPARQL was designed to query. So, SPARQL can find information quickly that is hidden in data that is not all the same and is stored in different ways and formats. For this reason, it can easily use SPARQL to query resources

from different sources and then link them according to different algorithms. In addition, SPARQL is a very general language that is easy to learn, and many people can get used to it in a very short time, so it is easier to make the application more accessible to more people. For all of these reasons, using SPARQL-based linking is reasonable.

2.3 Apache Jena

Apache Jena (or Jena in short) is a free and open-source Java framework for building semantic web and Linked Data applications. The framework is composed of different APIs interacting together to process RDF data [2].

2.3.1 Framework Architecture

Figure 2.2 show the architecture of the Apache Jena framework.

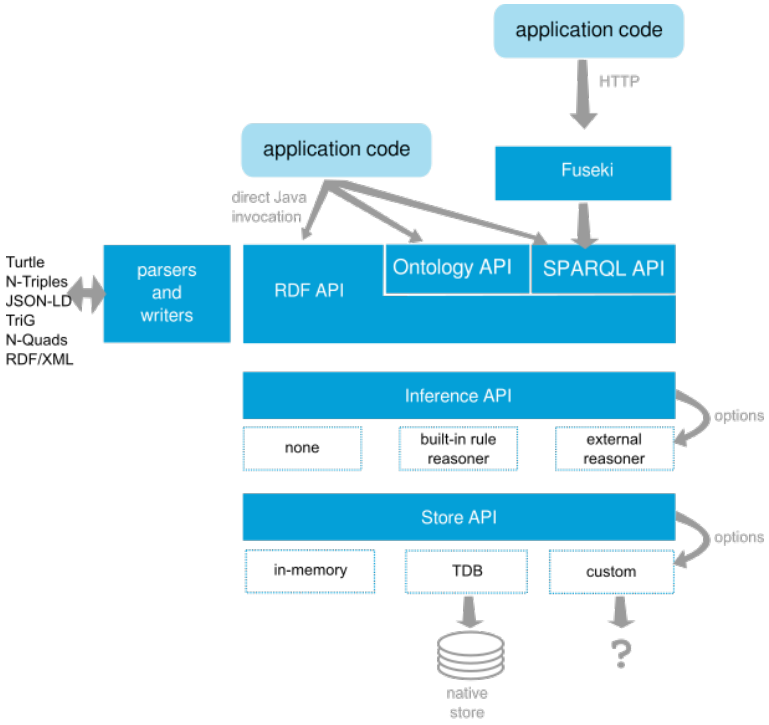

Figure 2.2: Jena Architecture

The Apache Jena framework is composed of three main components:

- The **RDF** API, which is used to create and manipulate RDF graphs [3].
- The **Ontology** API, which provides a consistent programming interface for ontology application development, independent of which ontology language which using in the programs [4].
- The **SPARQL** API, which is a query engine for Jena that supports the SPARQL RDF Query language [5].

In addition, the Apache Jena framework provides an **inference** API, which is designed to allow a range of inference engines or reasoners to be plugged into Jena. Such engines are used to derive additional RDF assertions, which are entailed by some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner [8]. Like the other frameworks, Apache Jena also provides a server – Fuseki [9]. Apache Jena Fuseki is a SPARQL server. It can run as an operating system service, as a Java web application (WAR file), or as a standalone server, which is tightly integrated with TDB [10] to provide a robust, transactional persistent storage layer and incorporates Jena text query.

The goal of this TFM is pretty clear so far. Choosing Apache Jena as the framework for SPARQL-based linking application, which use the SPARQL API of Apache Jena, called ARQ,

to handle SPARQL queries and also customize the linking functions. Moreover, the official documentation [5] and source code gave strong support to this project.

2.3.2 Jena ARQ: General Usage

ARQ is a query engine for Jena that supports the SPARQL RDF Query language [5]. SPARQL is a query language and a way to access RDF, and it is just a "data-oriented" query language, which means that it only asks about the information in the models and doesn't do any inference or do other operations. Of course, the Jena `model` is much "smarter" in the sense that it creates triples on demand, including OWL reasoning or other customize features shown below:

- Standard SPARQL
- Free text search via Lucene
- SPARQL/Update
- Access and extension of the SPARQL algebra
- Support for custom filter functions, including JavaScript functions
- Property functions for custom processing of semantic relationships
- Aggregation, GROUP BY and assignment as SPARQL extensions
- Support for federated query
- Support for extension to other storage systems
- Client-support for remote access to any SPARQL endpoint

The majority of the RDF operations that may be performed with ARQ are derived from the RDF model, which consists of a collection of statements. There are offered methods for the creation of resources, literals, and properties, as well as the statements that link these entities together. There are also methods for querying a model, adding and removing statements from a model, and set operations for merging models. The simple example shows below is about how to use ARQ to create an RDF model and run previous SPARQL of Listing 2.2 in Java.

Listing 2.3: Jena ARQ Example

```
1  public class TestDemo {
2      private static final String RDF = "@prefix github: <https://github.com/oeg-upm/> .\n" +
3              "@prefix oeg:    <https://oeg.fi.upm.es/researcher#> .\n" +
4              "@prefix vcard:  <http://www.w3.org/2006/vcard/ns#> .\n" +
5              "oeg:wenqi  vcard:hasName  \"Wenqi Jiang\" ;\n" +
6              "        vcard:hasProject  github:LOOM-LD .\n" +
7              "github:LOOM-LD  vcard:hasName  \"LOOM-LD\" .";
8
9      private static final String SPARQL = "PREFIX oeg: <https://oeg.fi.upm.es/researcher#>\n" +
10             "PREFIX github: <https://github.com/oeg-upm/>\n" +
11             "PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>\n" +
12             "SELECT ?name ?projectName\n" +
13             "WHERE {\n" +
14             "    ?researcher vcard:hasProject ?project .\n" +
15             "    ?researcher vcard:hasName ?name .\n" +
16             "    ?project vcard:hasName ?projectName .\n" +
17             "}";
18
19     public static void main(String[] args) throws IOException {
20         Model model = ModelFactory
21                 .createDefaultModel()
22                 .read(IOUtils.toInputStream(RDF, "UTF-8"), null, "TURTLE");
23         Query query = QueryFactory.create(SPARQL);
24
25         try (QueryExecution qexec = QueryExecutionFactory.create(query, model)) {
```

```
26              ResultSet results = qexec.execSelect();
27              ResultSetFormatter.out(System.out, results, query);
28          }
29      }
30  }
```

This `TestDemo` is basic and straightforward, mainly using the following APIs:

- `ModelFactory.createDefaultModel()` to create a new model, and it can load the RDF data from a string.
- The `Model` is a collection of statements.
- The `Query` a class that represents the application query. It is a container for all the details of the query. Objects of the class `Query` are normally created by calling one of the methods of `QueryFactory` which provides access to the various parsers.
- The `QueryExecution` represents one execution of a query.
- The `ResultSet` is an iterator, which contains the same query result in table 2.1.
- The `ResultSetFormatter` is a result set formatter, which can print the result in console, or turn it into a JSON, text, or as plain XML.

The application API [6] is in the package `org.apache.jena.query`, which is the main package for most applications.

2.3.3 Jena ARQ: Custom Function

Use Jena ARQ as the SPARQL engine to query local or online ontology databases because it is powerful. And it also can be used to implement the linking algorithms with custom functions, which is what the project care about the most because ARQ supports custom functions as allowed by the SPARQL 1.1 specification. Applications can add SPARQL functions to the query engine. This is done by writing a class implementing the right interface, then either registering it or using the fake `java:` URI scheme to dynamically call the function. [7]

Here, it shows how to customize a function to link scientists in different ontology databases according to the similarity of names.

2.3.3.1 Writing a SPARQL value function

A SPARQL value function is an extension point of the SPARQL query language that allows a **URI** to name a function in the query processor. In the Jena ARQ engine, the custom function's code must implement the interface `org.apache.jena.sparql.function.Function`, which provides two methods that must be implemented. In addition, to specify the number of arguments, it has to extend directly some abstract class. Functions do not have to have a fixed number of arguments, the abstract class `FunctionBase` has 4 subclasses from `FunctionBase1` to `FunctionBase4`.

In the example of ARQ custom function below, it extends the `org.apache.jena.sparql.function.FunctionBase2` that allows to deal with two parameters to develop the linking algorithm.

Listing 2.4: Jena ARQ Custom Function Example

```
1 public static class CustomFunction extends FunctionBase2 {
2     Cosine cosine = new Cosine();
3
4     @Override
5     public NodeValue exec(NodeValue v1, NodeValue v2) {
6         return NodeValue.makeDouble(cosine.similarity(v1.asString(), v2.asString()));
7     }
8 }
```

The function takes two arguments and returns a single value. The class `NodeValue` represents values and supports value-based operations. `NodeValue` value support includes the **XSD datatypes**, `xsd:decimal` and all its subtypes like `xsd:integer` and `xsd:byte`, `xsd:double`, `xsd:float`, `xsd:boolean`, `xsd:dateTime` and `xsd:date`. Literals with language tags are also treated as values in additional "value spaces" determined by the language tag without regard to case. Here, it uses `xsd:double` as the datatype of similarity result.

2.3.3.2 Registering the function

The query compiler finds functions based on the functions **URI**. For each function, there is a function factory associated with the **URI**. A new function instance is created for each use of a function in each query execution. A common case is registering a specific class for a function implementation, so there is an addition method that takes a class, wraps in a built-in function factory and registers the function implementation. The following code registers the `CustomFunction` class as the function implementation for the `http://oeg.upm.es/loom-ld/functions/linking#`.

Listing 2.5: Jena ARQ Custom Function Registry

```
FunctionRegistry ref = FunctionRegistry.get();
ref.put("http://oeg.upm.es/loom-ld/functions/linking#", CustomFunction.class);
```

2.3.3.3 Executing the function

After registering the function, the follow step just needs to add the URL of this function into the SPARQL query. The code below shows how to use the function, which is designed to link scientists in different ontology databases according to the similarity of names.

Listing 2.6: Jena ARQ Custom Function Execution

```
String queryString = "PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n"
    + "PREFIX foaf:     <http://xmlns.com/foaf/0.1/>\n"
    + "PREFIX dbo:      <http://dbpedia.org/ontology/>\n"
    + "PREFIX wd:       <http://www.wikidata.org/entity/>\n"
    + "PREFIX wdt:      <http://www.wikidata.org/prop/direct/>\n"
    + "PREFIX linking:    <http://oeg.upm.es/loom-ld/functions/linking#>\n"
    + "SELECT ?name1 ?name2 ?similarity\n"
    + "WHERE {\n"
    + "   SERVICE <http://live.dbpedia.org/sparql?timeout=300000> {\n"
    + "     ?scientist1 rdf:type dbo:Scientist.\n"
    + "     ?scientist1 foaf:name ?name1. \n"
    + "   }\n"
    + "   SERVICE <https://query.wikidata.org/sparql?timeout=300000> {\n"
    + "     ?scientist2 wdt:P106 wd:Q901.\n"
    + "     ?scientist2 wdt:P1559 ?name2.\n"
    + "   }\n"
    + "   BIND(linking:(?name1, ?name2) AS ?similarity)\n"
    + "   FILTER ( ?similarity >= 0.5)\n"
    + "}\n"
    + "Limit 10";

Model model = ModelFactory.createDefaultModel();
Query query = QueryFactory.create(queryString);

try (QueryExecution qexec = QueryExecutionFactory.create(query, model)) {
    ResultSet results = qexec.execSelect();
    ResultSetFormatter.out(System.out, results, query);
}
```

The following table shows the results of the query, the first column presents the name of the scientist in the **DBpedia**, the second column presents the name of the scientist in the

Wikidata, and the third column presents the cosine similarity of the two names.

name1	name2	similarity
"Grigory Margulis"@en	"Dan Margulis"@en	0.5916079783099616e0
"Grigory Margulis"@en	"Dan Margulis"@en	0.5916079783099616e0
"Grzegorz Rempała"@en	"Grzegorz Nowik"@pl	0.5400617248673217e0
"Grzegorz Rempała"@en	"Grzegorz Sosna"@pl	0.5400617248673217e0
"Grzegorz Rozenberg"@en	"Grzegorz Nowik"@pl	0.5051814855409226e0
"Grzegorz Rozenberg"@en	"Grzegorz Sosna"@pl	0.5051814855409226e0
"Guillaume Amontons"@en	"Guillaume Raoult"@fr	0.5345224838248488e0
"Guillaume Leduey"@en	"Guillaume Raoult"@fr	0.5714285714285714e0
"Gustav Flor"@en	"Gustav Jäger"@de	0.5270462766947299e0
"Gustav Mie"@en	"Gustav Jäger"@de	0.5590169943749475e0

Table 2.2: The Custom Function Result

Chapter 3

LOOM-LD

After giving a general background, this section will dive into the implementation details of the project, LOOM-LD, an RDF linking project that consists of generating relationships between RDF resources from different, or the same, datasets. In other words, the application generates links between RDF resources, generally `owl:sameAs`. RDF linking relies on one or more link rules to generate the links. Link rules specify the conditions under which two RDF resources can be considered the same. Due to this reason and the features of Apache Jena described in the previous chapter, it could easily implement the linking algorithms. This chapter focuses on two types of linking algorithms, one based on text similarity and one based on geometric relationships. So, the structure of this chapter is straightforward:

- Section §3.1 describes the text similarity linking discovery algorithms
- Section §3.2 describes the geometry linking discovery algorithm.

3.1 Text Linking

The most common use scenario for a graph linker is to link similar text content. For example, if two resources have the same title, they are likely to be the same resource. Because string-based similarity is the oldest, simplest, and most common measurement approach for text linking, it can be used in linking the resources. The way this measure works is based on string sequences and how characters are put together. It is based on the idea that two strings are similar if they have the same characters in the same index. For example, the strings `'a'` and `'a'` are similar because they have the same characters in the same index. The strings `'ac'` and `'ab'` are not similar because they have different characters in the same index. Two main types of string similarity algorithms are character-based similarity functions, and token-based similarity [11].

3.1.1 Character-based Similarity

Character-based Similarity is also called sequence-based or edit distance measurement, which is a way of quantifying how dissimilar two strings are to one another by counting the minimum number of operations required to transform one string into the other, the operations are including insertion, deletion, and substitution. In another word, two strings are similar if the edit distance minimum operation number is smaller than the given threshold. However, different edit distance algorithms have different definitions of the operations they need to do in different scenarios, and the most common algorithms are shown below:

- The Levenshtein Distance allows deletion, insertion, and substitution.
- The Longest Common Subsequence (LCS) distance allows only insertion and deletion, not substitution.
- The Jaro distance allows only transposition.

3.1.1.1 Levenshtein distance

The Levenshtein distance between two words is the number of single-character modifications (insertions, deletions, or substitutions) necessary to turn one word into the other. It is named after Soviet mathematician Vladimir Levenshtein, who researched it in 1965 [12]. Edit distance is another name for Levenshtein distance; however, it also refers to a wider family of distance measures known as edit distance. The Levenshtein distance between two strings a,b (of length $|a|$ and $|b|$ respectively) is given by $lev(a, b)$:

$$lev(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ lev(tail(a), tail(b)) & \text{if } a[0] = b[0], \\ 1 + min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & \text{Otherwise.} \end{cases} \tag{3.1}$$

Where the $tail$ of some string x is a string of all but the first character of x, and $x[n]$ is the nth character of the string x, counting from 0. In the minimum equation, the first element $lev(tail(a), b)$ corresponds to deletion (from a to b), the second $lev(a, tail(b))$ to insertion and the third $lev(tail(a), tail(b))$ to replacement. This formulation fits exactly onto the naïve recursive implementation of the algorithm.

After got the Levenshtein distance, it is normalized by dividing it by the length of the longest string. The resulting value will always be in the interval $[0.0, 1.0]$. The normalized formula is:

$$normalized\_lev(a,b) = 1.0 - \frac{lev(a,b)}{max(|a|, |b|)} \tag{3.2}$$

3.1.1.2 Longest common subsequence

The longest common subsequence (LCS) problem involves finding the longest subsequence that is shared by two or more sequences. Unlike longest common substring problems, LCS problems do not need subsequences to occupy consecutive indexes inside the original sequences. LCS distance equals Levenshtein distance when only insertion and deletion are permitted (no substitution) or when the cost of a substitution is twice that of an insertion or deletion. This method can use the dynamic programming approach, and the equation shows below:

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1})^\frown x_i & \text{if } i, j > 0 \text{ and } x_i = y_j \\ max\left\{LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\right\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases} \tag{3.3}$$

$X = (x_1 x_2 \cdots x_n)$ $Y = (y_1 y_2 \cdots y_n)$. X_i is the prefix of X and Y_j is the prefix of Y. Comparing x_i and y_j to calculate the result $LCS(X_i, Y_j)$, if they are equal, the result will extend the $LCS(X_{i-1}, Y_{j-1})$ by x_i, otherwise it will take the maximum of the $LCS(X_i, Y_{j-1})$ and $LCS(X_{i-1}, Y_j)$.

After got the LCS distance, it needs to be normalized by dividing by the length of the longest string. The resulting value will always be in the interval $[0.0, 1.0]$, here is the equation:

$$normalized\_LCS(X, Y) = 1.0 - \frac{LCS(X, Y)}{max(|X|, |Y|)} \tag{3.4}$$

3.1.1.3 Jaro-Winkler distance

Jaro Similarity is the measure of similarity between two strings [14]. The value of Jaro distance ranges from 0 to 1. where 1 means the strings are equal and 0 means no similarity between the two strings.

The Jaro Similarity is calculated using the following formula

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \tag{3.5}$$

Where:

- where $|s_1|$ and $|s_2|$ are the lengths of strings s_1 and, s_2 respectively.
- m is the number of matching characters
- t is the number of transpositions

Jaro-Winkler similarity [13] uses a prefix scale ρ which gives more favorable ratings to strings that match from the beginning for a set prefix length l. Given two strings s_1 and s_2, their Jaro-Winkler similarity sim_w is:

$$sim_w = sim_j + \ell\rho\,(1 - sim_j) \tag{3.6}$$

Where:

- sim_j is the Jaro similarity for strings s_1 and s_2
- ℓ is the length of common prefix at the start of the string up to a maximum of 4 characters
- ρ is a constant scaling factor for how much the score is adjusted upwards for having common prefixes. ρ should not exceed 0.25 (i.e., 1/4, with 4 being the maximum length of the prefix being considered). Otherwise, the similarity could become larger than 1. The standard value for this constant in Winkler's work is $\rho = 0.1$

3.1.1.4 Gestalt pattern matching

Gestalt Pattern Matching, also Ratcliff/Obershelp Pattern Recognition, is a string-matching algorithm for determining the similarity of two strings. It was developed in 1983 by John W. Ratcliff and John A. Obershelp and published in the Dr. Dobb's Journal in July 1988 [15].

The similarity of two strings S_1 and S_2 is determined by the formula, calculating twice the number of matching characters K_m divided by the total number of characters of both strings. The matching characters are defined as the longest common substring (LCS) plus recursively the number of matching characters in the non-matching regions on both sides of the LCS.

$$D_{ro} = \frac{2K_m}{|S_1| + |S_2|} \tag{3.7}$$

Note $0 \leq D_{ro} \leq 1$, where the result 1 indicates a perfect match between the two strings, while the value 0 indicates there is no match and not even a single letter in common.

3.1.2 Token-based Similarity

The term-based similarity is also called token-based because it treats each string as a set of tokens. Sets of tokens, like words, can be moved around to see how similar two strings are. The main idea behind this method is to measure how similar two strings are based on tokens that are common to both strings. If the similarity is marked, the pair of strings is marked as being the same or similar. The problem with character-based similarity is that it doesn't work well when the string is long. In other words, character-based becomes too expensive to run on a computer and less accurate for strings that are too long, like those in text documents. This part discusses some well-known similarity functions that are based on tokens: Jaccard similarity and Cosine similarity.

3.1.2.1 Jaccard similarity

Jaccard similarity [16] is computed as the number of shared terms over the union of all the terms in both strings.

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.8}$$

From figure 3.1, the principle of this algorithm is shown very straightforward.

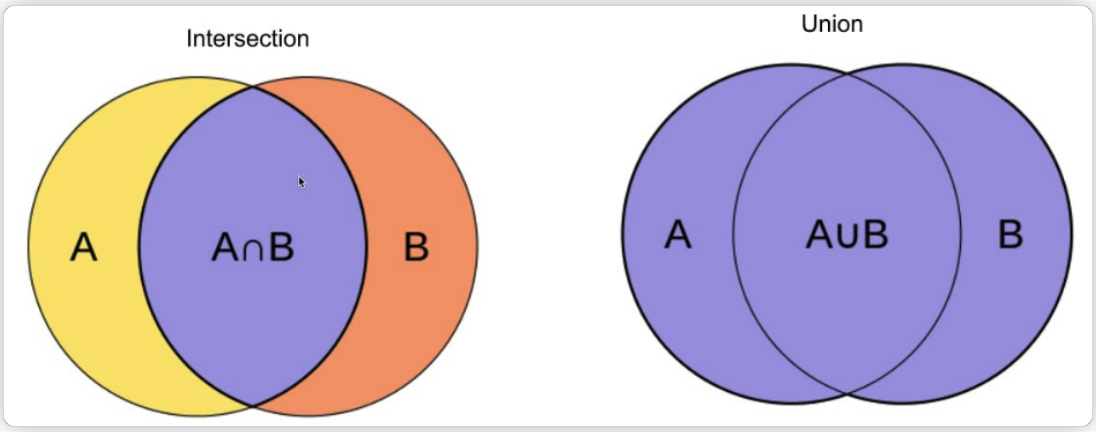

Figure 3.1: Jaccard

3.1.2.2 Cosine similarity

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, Cosine similarity measures the cosine of the angle between two vectors projected in a multidimensional space. In cosine similarity, data objects in a dataset are treated as a vector. The formula to find the cosine similarity between two vectors is

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} = \frac{\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{y}_i}{\sqrt{\sum_{i=1}^{n} (\boldsymbol{x}_i)^2}\sqrt{\sum_{i=1}^{n} (\boldsymbol{y}_i)^2}} \tag{3.9}$$

3.1.3 Implementation

After explaining the fundamental algorithms, the next part is the introduction of the project code. It needs to implement custom linking functions, the specific approach has been described in detail at the end of Chapter 2, State of the Art, where describes the code more clean and straightforward. The implementation of the text similarity algorithms is directly using the open-source project: java-string-similarity [17]. First, taking a brief look at the architecture of the text linking, as shown in the following example.

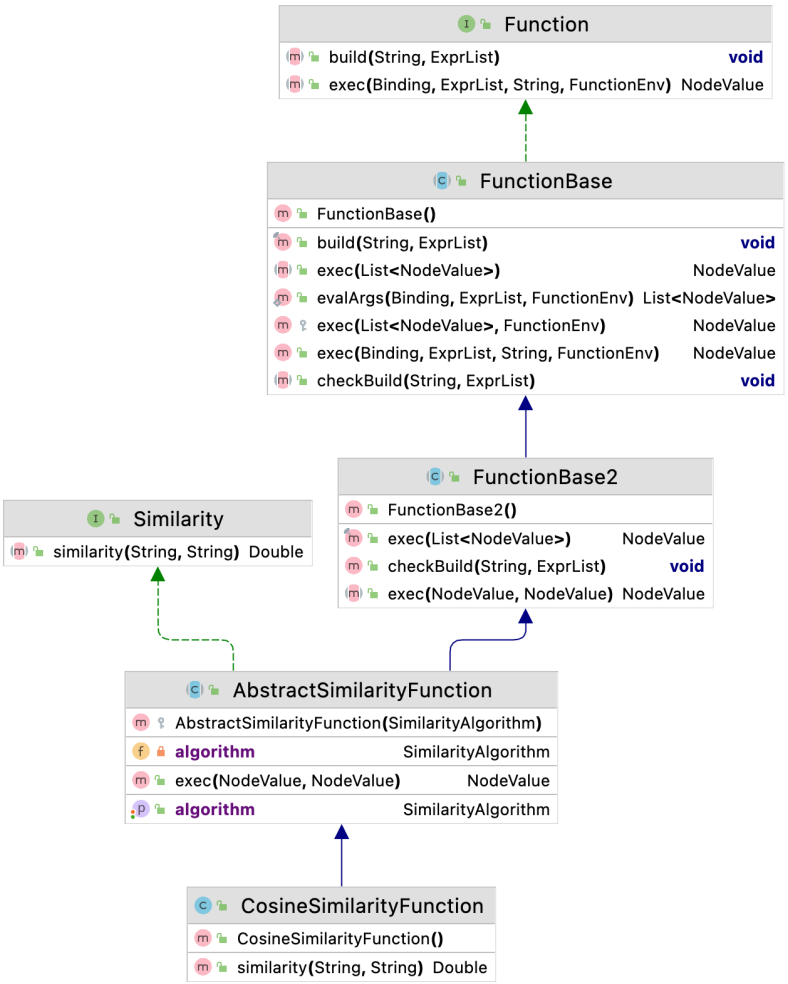

Figure 3.2: Cosine Similarity Function

Interfaces specify what a class must do rather than how to do, it is the blueprint of the class. So, this part of code has implemented a custom `Similarity` interface to calculate text

similarity, which is the main goal, and it can significantly reduce code redundancy.

Listing 3.1: Similarity Interface

```
public interface Similarity {
    /**
        * Get two strings' similarity
        *
        * @param element1 string 1
        * @param element2 string 2
        * @return Degree of similarity(0-1)
        */
    Double similarity(String element1, String element2);
}
```

Then it created an abstract class, `AbstractSimilarityFunction`, which does some common preparatory work, like extends `FunctionBase2` and implements `Similarity`. This abstract class can choose the specific similarity algorithm to be used, and build the common parts of the algorithm. Here is the implementation of the abstract class.

Listing 3.2: The abstract class of similarity

```
public abstract class AbstractSimilarityFunction extends FunctionBase2 implements Similarity {

    private SimilarityAlgorithm algorithm;

    protected AbstractSimilarityFunction(SimilarityAlgorithm algorithm) {
        this.algorithm = algorithm;
    }


    public SimilarityAlgorithm getAlgorithm() {
        return algorithm;
    }

    public void setAlgorithm(SimilarityAlgorithm algorithm) {
        this.algorithm = algorithm;
    }

    @Override
    public NodeValue exec(NodeValue v1, NodeValue v2) {
        String element1 = v1.asString();
        String element2 = v2.asString();
        Double score = similarity(element1, element2);
        return NodeValue.makeDouble(score);
    }
}
```

After implementing the abstract class, the next step is about implementing the concrete classes, which are the actual similarity functions. For instance, the `CosineSimilarityFunction` is shown below.

Listing 3.3: Cosine Similarity Function

```
public class CosineSimilarityFunction extends AbstractSimilarityFunction {

    private final Cosine cosine;

    public CosineSimilarityFunction() {
        super(SimilarityAlgorithm.COSINE);
        cosine = new Cosine();
    }


    @Override
```

```
12      public Double similarity(String element1, String element2) {
13          return cosine.similarity(element1, element2);
14      }
15  }
```

Once the algorithms are implemented, it is necessary to be registered, and the prefix of the registered URL used is `https://oeg.upm.es/loom-ld/functions/linking/text#`. So, the full URL for the function of Cosine Similarity algorithm above is `https://oeg.upm.es/loom-ld/functions/linking/text#cosine`. In addition, there are many other algorithms, so the detailed code is shown below.

Listing 3.4: Register Functions

```
1  public class CustomFunctions {
2      private static final String TEXT_PREFIX = CustomARQConstants.TEXT_PREFIX;
3      private static final String GEOMETRY_PREFIX = CustomARQConstants.GEOMETRY_PREFIX;
4      private static final FunctionRegistry FUNCTION_REGISTRY = FunctionRegistry.get();
5
6      private CustomFunctions() {
7      }
8
9      public static void loadTextFunctions() {
10         FUNCTION_REGISTRY.put(TEXT_PREFIX + SimilarityAlgorithm.COSINE,
               CosineSimilarityFunction.class);
11         FUNCTION_REGISTRY.put(TEXT_PREFIX + SimilarityAlgorithm.JACCARD,
               JaccardSimilarityFunction.class);
12         FUNCTION_REGISTRY.put(TEXT_PREFIX + SimilarityAlgorithm.JARO_WINKLER,
               JaroWinklerSimilarityFunction.class);
13         FUNCTION_REGISTRY.put(TEXT_PREFIX + SimilarityAlgorithm.LEVENSHTEIN,
               LevenshteinSimilarityFunction.class);
14         FUNCTION_REGISTRY.put(TEXT_PREFIX + SimilarityAlgorithm.LCS, LCSSimilarityFunction.
               class);
15         FUNCTION_REGISTRY.put(TEXT_PREFIX + SimilarityAlgorithm.RATCLIFF_OBERSHELP,
               RatcliffObershelpSimilarityFunction.class);
16     }
17 }
```

All need to do is load those functions before invoking them, `CustomFunctions.loadTextFunctions()`. After load them, it just needs to specify those functions in the SPARQL queries, like this:

Listing 3.5: Text Linking Sparql

```
1  PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX owl:      <http://www.w3.org/2002/07/owl#>
3  PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
4  PREFIX foaf:     <http://xmlns.com/foaf/0.1/>
5  PREFIX dbo:      <http://dbpedia.org/ontology/>
6  PREFIX wd:       <http://www.wikidata.org/entity/>
7  PREFIX wdt:      <http://www.wikidata.org/prop/direct/>
8  CONSTRUCT { ?scientist1 owl:sameAs ?scientist2. }
9  WHERE {
10   SERVICE <http://live.dbpedia.org/sparql?timeout=30000> {
11     ?scientist1 rdf:type dbo:Scientist;
12       foaf:name ?name1.
13   }
14   SERVICE <https://query.wikidata.org/sparql?timeout=30000> {
15     ?scientist2 wdt:P106 wd:Q901;
16       wdt:P1559 ?name2.
17   }
18   FILTER((loom:cosine(?name1, ?name2)) > "0.85"^^xsd:decimal)
19 }
20 LIMIT 8
```

3.2 Geometry Linking

There is another type of linking in the graph linker usage scenarios besides linking based on text similarity, which is to link two regions based on their spatial relationship. It is called geometry linking. Before implementing the set of linking algorithms, two basic concepts have to be clarified, the first one is how to represent the spatial data. The second one is what are the relationships between these spatial data.

3.2.1 Well-know Text

Well-known text (WKT) is a text markup language that is used to express vector geometry objects. A binary version, known as well-known binary (WKB), is used to transport and store the same information in a more compact, computer-readable format. The Open Geospatial Consortium (OGC) first specified the formats in their Simple Feature Access document. [18]. The WKT can represent the following distinct geometric objects:

- Point, MultiPoint
- LineString, MultiLineString
- Polygon, MultiPolygon, Triangle
- PolyhedralSurface
- TIN (Triangulated irregular network)
- GeometryCollection

Some examples are shown below. The first table, Table 3.1, shows some 2D examples, and the second table, Table 3.2, shows some 3D examples.

Types	Examples
Point	POINT (30 10)
LineString	LINESTRING (30 10, 10 30, 40 40)
Polygon	POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
	POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))

Table 3.1: Geometry primitives (2D)

Types	Examples
MultiPoint	MULTIPOINT ((10 40), (40 30), (20 20), (30 10))
	MULTIPOINT ((10 40), (40 30), (20 20), (30 10))
MultiLineString	MULTILINESTRING ((10 10, 20 20, 10 40), (40 40, 30 30, 40 20, 30 10))
MultiPolygon	MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))
	MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))

Table 3.2: Geometry primitives (3D)

3.2.2 DE-9IM

The Dimensionally Extended 9-Intersection Model (DE-9IM) is a topological model and a standard used to describe the spatial relations of two regions (two geometries in two-dimensions, R^2) in geometry, point-set topology, geospatial topology and fields related to computer spatial analysis. The model was developed by Clementini and others [19]. It has been used as a basis for standards for queries and assertions in GIS and spatial databases.

The DE-9IM model is based on the following 3×3 intersection matrix:

$$DE9IM(a,b) = \begin{bmatrix} dim(I(a) \cap I(b)) & dim(I(a) \cap B(b)) & dim(I(a) \cap E(b)) \\ dim(B(a) \cap I(b)) & dim(B(a) \cap B(b)) & dim(B(a) \cap E(b)) \\ dim(E(a) \cap I(b)) & dim(E(a) \cap B(b)) & dim(E(a) \cap E(b)) \end{bmatrix} \tag{3.10}$$

Where dim is the dimension of the intersection ($\cap$) of the interior (I), boundary (B), and exterior (E) of geometries a and b. When two polygonal shapes overlap, the result of the function **DE9IM(a, b)** looks like this:

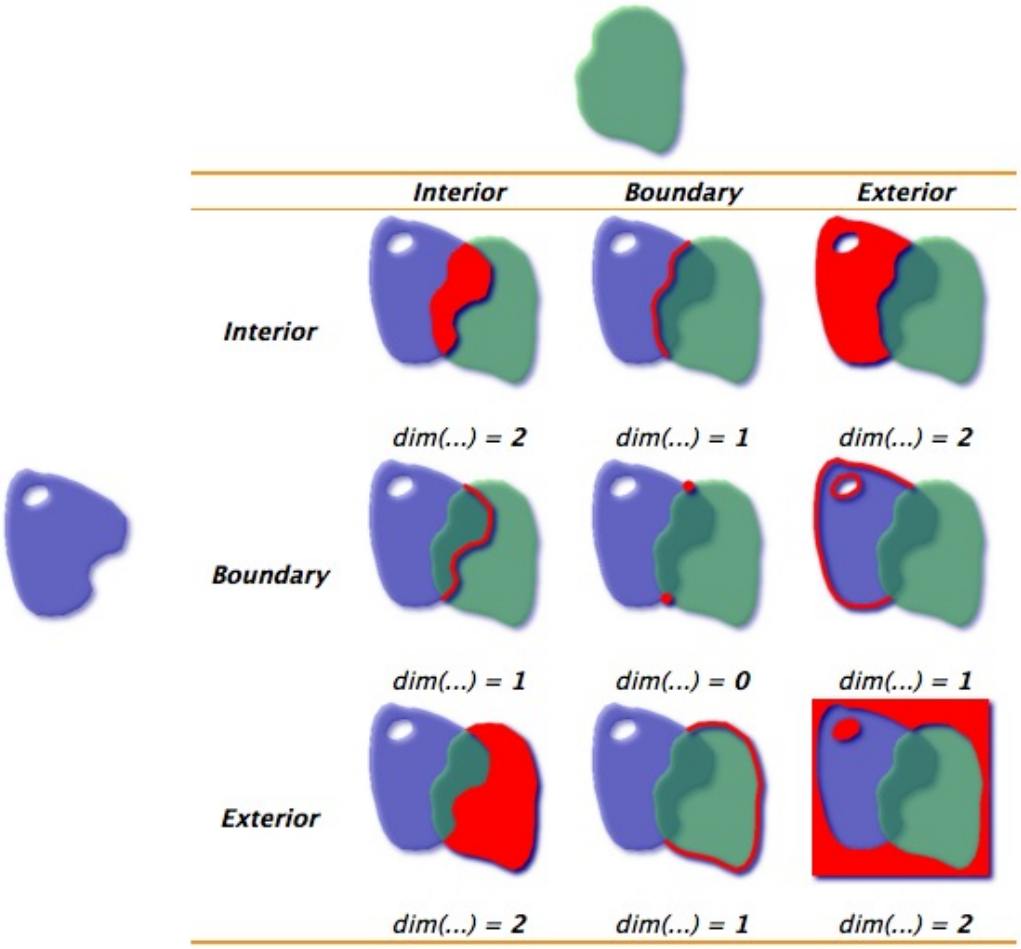

Figure 3.3: DE-9IM

Based on two spatial data, it's clear that their relationships can be predicated by using the function **DE9IM(a, b)**. The relationship predicates are `contains`, `coveredBy`, `covers`, `disjoint`,

`equals`,`intersects`, `overlaps`, `touches`,`crosses` and `within`. The project can use such relations to link these spatial graphs.

3.2.3 Implementation

Like text linking, after introducing the relationships of spatial data, this project implemented the function of determining 10 different relationships, and it used the JTS Topology Suite [20]. The diagram about the geometry equals function is shown below.

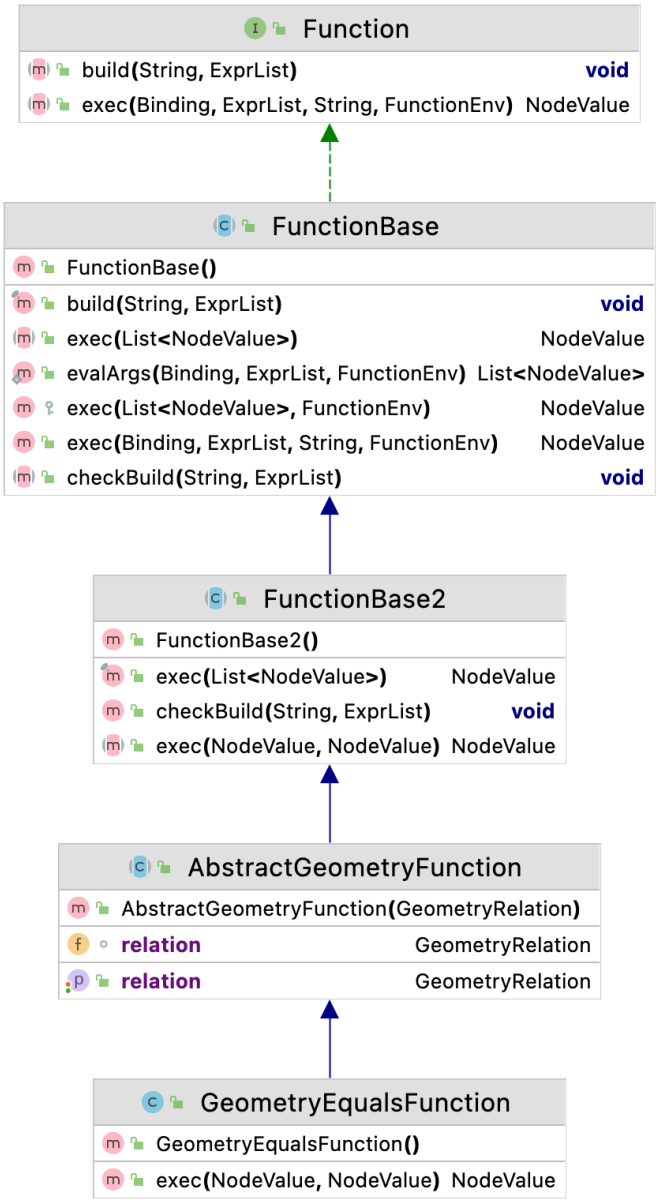

Figure 3.4: Geometry Equals Function

Similarly, it needs an abstract class to do some preparatory work, which can initialize the `WKTReader` to load the spatial data.

Listing 3.6: Abstract Geometry Function

```
public abstract class AbstractGeometryFunction extends FunctionBase2 {
    public static final Logger LOGGER = LoggerFactory.getLogger(SparqlExecutor.class);
    GeometryRelation relation;
    WKTReader reader;

    public AbstractGeometryFunction(GeometryRelation relation) {
        this.relation = relation;
        reader = new WKTReader();
    }

    public GeometryRelation getRelation() {
        return relation;
    }

    public void setRelation(GeometryRelation relation) {
        this.relation = relation;
    }
}
```

And then the project specified the implementation of the function, which is elementary, it just needs to use `WKTReader` to load the spatial data and use the relationship function in it. For example, the function **equals** are defined as follows:

Listing 3.7: Geometry Equals Function

```
public class GeometryEqualsFunction extends AbstractGeometryFunction {
    public GeometryEqualsFunction() {
        super(GeometryRelation.EQUALS);
    }

    @Override
    public NodeValue exec(NodeValue v1, NodeValue v2) {
        try {
            Geometry g1 = reader.read(v1.asString());
            Geometry g2 = reader.read(v2.asString());
            return NodeValue.makeBoolean(g1.equals(g2));
        } catch (ParseException e) {
            LOGGER.error(e.getMessage(), e);
            return NodeValue.FALSE;
        }
    }
}
```

At the end, the application can use these methods to determine the geometric relationships between spatial data by registering the functions. The registration address is prefixed with `https://oeg.upm.es/loom-ld/functions/linking/geometry#`, and there are different URLs for different functions, for example:

`https://oeg.upm.es/loom-ld/functions/linking/geometry#equals`.

Here is a simple example of how to use the `equals` functions in the below SPARQL query.

Listing 3.8: Geometry Linking Sparql

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:  <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT { ?source ?target "1.0"^^xsd:decimal. }
WHERE {
  ?source geometry:isLocatedAt "sourceEQUALS-0001.nt";
    strdf:hasGeometry ?sourceGeometry.
  ?target geometry:isLocatedAt "targetEQUALS-0001.nt";
    strdf:hasGeometry ?targetGeometry.
  BIND(geometry:equals(?sourceGeometry, ?targetGeometry) AS ?relation)
```

```
11   FILTER(?relation = "true"^^xsd:boolean)
12 }
```

So far, the kernel code has been introduced, and the next chapter will explore how to apply the application to real-world problems and evaluate the results in a comprehensive.

Chapter 4

Evaluation and Conclusion

After implementing the text linking algorithms and the geometric linking algorithms, it is time to apply them to the reality, and before that, the project needs to be checked the effectiveness and efficiency of some linking task results. So, this chapter focuses on applying the algorithms to the dataset of Ontology Alignment Evaluation Initiative (OAEI). It is organized as follows: Section §4.1 presents the OAEI contest and the experiment metrics, Section §4.2 presents the experiment datasets and each results.

4.1 Ontology Alignment Evaluation Initiative (OAEI)

When applications and agents use heterogeneous ontologies, it is important to reconcile these ontologies before they can interact; this is referred to as heterogeneity problems. Some heterogeneity problems on the semantic web can be resolved by aligning or matching the ontologies, which involves locating the corresponding entities. So, the application needs to consider the various parameters of the alignment task that must be controlled during the experiment and examine the measures that can be used for an evaluation. The increasing number of methods available for schema or ontology matching mandates consensus for evaluation of these methods. However, very few experimental comparisons of algorithms are available. One of the goals of the Ontology Alignment Evaluation Initiative (OAEI) [21], which was founded in 2004, is to conduct such an evaluation. OAEI presents a framework for arranging the evaluation based on some principles and previous efforts in the field of ontology alignment [23].

The goals of the OAEI are:

- assessing strengths and weaknesses of alignment/matching systems;
- comparing performance of techniques;
- increase communication among algorithm developers;
- improve evaluation techniques;
- most of all, helping to improve the work on ontology alignment/matching.

The OAEI's major purpose is to develop ontology alignment procedures, and it also helps to improve evaluation methodology. It outlined the types of tests to be processed, which contained two types of tests: competency and performance benchmarks; as well as the metrics for evaluating the results for this purpose.

4.1.1 Evaluation Metrics

4.1.1.1 Competency benchmarks

There are many ways to qualitatively evaluate returned results [24]. Precision and recall are currently the most well-understood evaluation measures. One possibility consists of proposing a reference alignment (R) that is the one that the participants must find (a gold standard). The result from the evaluated alignment algorithm (A) can then be compared to that reference alignment. In what follows, the alignments A and R are considered to be sets of pairs.

- **Precision**:
$$P(A,R) = \frac{|R \cap A|}{|A|} \tag{4.1}$$
- **Recall**:
$$P(A,R) = \frac{|R \cap A|}{|R|} \tag{4.2}$$
- **F-measure**:
$$M_\alpha(A,R) = \frac{P(A,R) \cdot R(A,R)}{(1-\alpha) \cdot P(A,R) + \alpha \cdot R(A,R)} \tag{4.3}$$
If $\alpha = 1$, then the F-measure is equal to precision and if $\alpha = 0$, the **F-measure** is equal to recall. In between, the higher α, the more importance is given to precision regarding recall. Very often, the value $\alpha = 0.5$ is used, i.e. $M_{0.5}(A,R) = \frac{2 \times P(A,R) \times R(A,R)}{P(A,R)+R(A,R)}$.

4.1.1.2 Performance benchmarks

Performance measures (or non-functional measures) quantify the number of resources required to link two ontologies. Unlike compliance measures, performance measures are dependent on the processing environment benchmark and the underlying ontology management system. Therefore, it is rather challenging to obtain objective assessments. The following two indicators are worth considering.

- **Speed** is measured in the amount of time taken by the algorithms to perform their linking tasks.
- **Memory** is one more way to measure performance when the alignment task actually takes.

4.2 Results and Conclusions of the OAEI Tasks

Each year the OAEI initiative organizes a contest in which one challenge is link discovery, and the datasets used were presented at the OAEI OM 2021 Workshop at ISWC 2021 [22]. The aim of this section is to test the performance of this linking discovery tool, LOOM-LD, that implement string-based as well as topological approaches for identifying matching instances and spatial entities, and the process and results are evaluated for accuracy (precision, recall and f-measure) and time/memory performance.

4.2.1 SPIMBENCH

SPIMBENCH dataset is composed of a Tbox (contains the ontology and the instances) and corresponding Abox (contains only the instances). The datasets share almost the same ontology (with some difference in the properties' level, due to the structure-based transformations). The goal of the SPIMBENCH task is to determine when two instances describe the same creative work. It needs to match instances in the source dataset (Tbox1) against the instances of the target dataset (Tbox2). In other words, this project needs to produce a set of mappings between the pairs of matching instances that are found to refer to the same real-world entity. An instance in the source (Tbox1) dataset can have none or one matching counterparts in the target dataset (Tbox2). The SPIMBENCH task needs to map only instances of creative works (`http://www.bbc.co.uk/ontologies/creativework/NewsItem`, `http://www.bbc.co.uk/ontologies/creativework/BlogPost` and `http://www.bbc.co.uk/ontologies/creativework/Programme`) and not the instances of the other classes.

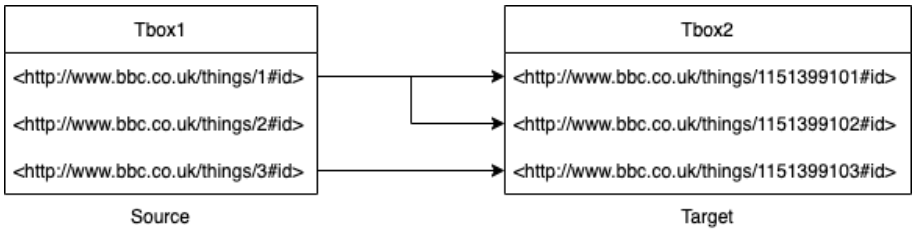

Figure 4.1: SPIMBENCH task

There are 3105 records in the source dataset (Tbox1) and 2802 records in the target dataset (Tbox2). Here is a pair of matching instances. It is very obvious that the properties are almost the same except for their IDs. Linking the creative entities from Tbox1 to Tbox2, which is all the task need to do.

Listing 4.1: An example of TBox1

```
1 <http://www.bbc.co.uk/things/107#id> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http
      ://www.bbc.co.uk/ontologies/creativework/BlogPost> .
2 <http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/title> "
      Montegaldella around both as conscientious gather page enactment display contribution
      secular." .
3 <http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/shortTitle>
       " accept early adopted necessarily laws last inform day be kinds." .
4 <http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/category> <
      http://dbpedia.org/ontology/Place> .
5 <http://www.bbc.co.uk/things/107#id>
      <http://www.bbc.co.uk/ontologies/creativework/description> " easier professional commerce
      as certain most modern land least policy all." .
6 <http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/about> <
      http://dbpedia.org/resource/Montegaldella> .
7 <http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/about> <
      http://dbpedia.org/resource/1966_South_African_Grand_Prix> .
8 <http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/about> <
      http://www.marymacleod.com/> .
```

```
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/mentions> <http://sws.geonames.org/7299903/> .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/audience> <http://www.bbc.co.uk/ontologies/creativework/InternationalAudience> .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/liveCoverage> "false"^^<http://www.w3.org/2001/XMLSchema#boolean> .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/primaryFormat> <http://www.bbc.co.uk/ontologies/creativework/TextualFormat> .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/dateCreated> "2011-08-06T15:56:35.003+03:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/dateModified> "2012-04-17T21:08:06.359+03:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/thumbnail> <http://www.bbc.co.uk/thumbnail/711152249> .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/creativework/altText> "thumbnail atlText for CW http://www.bbc.co.uk/context/107#id" .
<http://www.bbc.co.uk/things/107#id> <http://www.bbc.co.uk/ontologies/bbc/primaryContentOf> <http://www.bbc.co.uk/things/1448095947#id> .
```

Listing 4.2: An example of TBox2

```
<http://www.bbc.co.uk/things/646629526#id> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.bbc.co.uk/ontologies/creativework/BlogPost> .
<http://www.bbc.co.uk/things/646629526#id>
    <http://www.bbc.co.uk/ontologies/creativework/title> "Montegaldella around both as conscientious gather page enactment display contribution secular." .
<http://www.bbc.co.uk/things/646629526#id>
    <http://www.bbc.co.uk/ontologies/creativework/shortTitle> " accept early adopted necessarily laws last inform day be kinds." .
<http://www.bbc.co.uk/things/646629526#id>
    <http://www.bbc.co.uk/ontologies/creativework/description> " easier professional commerce as certain most modern land least policy all." .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/about> <http://dbpedia.org/resource/Montegaldella> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/about> <http://dbpedia.org/resource/1966_South_African_Grand_Prix> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/about> <http://www.marymacleod.com/> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/tag> <http://sws.geonames.org/7299903/> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/audience> <http://www.bbc.co.uk/ontologies/creativework/InternationalAudience> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/liveCoverage> "false"^^<http://www.w3.org/2001/XMLSchema#boolean> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/primaryFormat> <http://www.bbc.co.uk/ontologies/creativework/TextualFormat> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/dateCreated> "2011-08-06T15:56:35.003+03:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/dateModified> "2012-04-17T21:08:06.359+03:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/thumbnail> <http://www.bbc.co.uk/thumbnail/711152249> .
<http://www.bbc.co.uk/things/646629526#id> <http://www.bbc.co.uk/ontologies/creativework/altText> "thumbnail atlText for CW http://www.bbc.co.uk/context/646629526#id" .
```

Here is the result of the above pair, it just needs to use the `sameAs` to describe the relationship.

Listing 4.3: An result of the SPIMBENCH task

```
<http://www.bbc.co.uk/things/107#id>
    <http://www.w3.org/2002/07/owlsameAs>
        <http://www.bbc.co.uk/things/646629526#id> .
```

And the SPARQL used to get the above result is shown below.

Listing 4.4: A sparql of the SPIMBENCH task

```
PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>

CONSTRUCT { ?thing1 owl:sameAs ?thing2. }
WHERE {
    ?thing1 seals:isLocatedAt "tbox1.nt".
    OPTIONAL { ?thing1 cwork:title ?title1. }
    OPTIONAL { ?thing1 cwork:shortTitle ?shortTitle1. }
    OPTIONAL { ?thing1 cwork:description ?description1. }
    ?thing2 seals:isLocatedAt "tbox2.nt".
    OPTIONAL { ?thing2 cwork:title ?title2. }
    OPTIONAL { ?thing2 cwork:shortTitle ?shortTitle2. }
    OPTIONAL { ?thing2 cwork:description ?description2. }
    BIND(loom:lcs(?title1, ?title2) AS ?titleGrade)
    BIND(loom:lcs(?shortTitle1, ?shortTitle2) AS ?shortTitleGrade)
    BIND(loom:lcs(?description1, ?description2) AS ?descriptionGrade)
    FILTER(?thing1 != ?thing2)
    FILTER(((?titleGrade >= "0.7"^^xsd:decimal) || (?shortTitleGrade >= "0.7"^^xsd:decimal))
        || (?descriptionGrade >= "0.7"^^xsd:decimal))
}
```

The SPARQL shown above used the `loom:lcs` function to get similarity between two entities. The fields used are: `title`, `shortTitle`, `description`, and the similarity threshold is 0.7.

4.2.1.1 Results

The evaluation tried all the previous chapter's algorithms of text similarity to match the instances of **Tbox1** with the instances of **Tbox2**. Then calculating the evaluation metrics under the different thresholds from 0.2 to 1.0, including the precision, recall, f1 and running time. The fields used are **Title**, **ShortTitle** and **Description**. Here is the table of all results.

Algorithm	Similarity	Golden size	Result size	Precision	Recall	F1	Milliseconds
cosine	0.2	225	29608	0.007	0.947	0.014	16,704
cosine	0.3	225	3200	0.065	0.929	0.122	15,775
cosine	0.4	225	344	0.602	0.92	0.728	15,776
cosine	0.5	225	233	0.88	0.911	0.895	15,799
cosine	0.6	225	225	0.898	0.898	0.898	15,733
cosine	0.7	225	222	0.896	0.884	0.89	15,717
cosine	0.8	225	209	0.89	0.827	0.857	16,757
cosine	0.9	225	183	0.874	0.711	0.784	15,927
cosine	1.0	225	174	0.868	0.671	0.757	16,362
jaccard	0.2	225	356	0.587	0.929	0.719	17,415
jaccard	0.3	225	230	0.896	0.916	0.905	16,767
jaccard	0.4	225	225	0.898	0.898	0.898	16,809
jaccard	0.5	225	223	0.897	0.889	0.893	17,349
jaccard	0.6	225	216	0.894	0.858	0.875	21,176
jaccard	0.7	225	208	0.889	0.822	0.855	21,994
jaccard	0.8	225	182	0.874	0.707	0.781	20,371
jaccard	0.9	225	176	0.869	0.68	0.763	20,892
jaccard	1.0	225	174	0.868	0.671	0.757	23,224

jaro-winkler	0.2	225	118209	0.002	0.947	0.004	20,403
jaro-winkler	0.3	225	118209	0.002	0.947	0.004	22,296
jaro-winkler	0.4	225	118209	0.002	0.947	0.004	19,736
jaro-winkler	0.5	225	118209	0.002	0.947	0.004	16,073
jaro-winkler	0.6	225	117034	0.002	0.947	0.004	16,590
jaro-winkler	0.7	225	56900	0.004	0.938	0.007	15,931
jaro-winkler	0.8	225	232	0.828	0.853	0.84	15,407
jaro-winkler	0.9	225	189	0.873	0.733	0.797	15,518
jaro-winkler	1.0	225	174	0.868	0.671	0.757	15,139
levenshtein	0.2	225	115837	0.002	0.947	0.004	31,940
levenshtein	0.3	225	9164	0.023	0.947	0.045	27,970
levenshtein	0.4	225	276	0.772	0.947	0.85	24,341
levenshtein	**0.5**	225	237	**0.899**	0.947	**0.922**	24,110
levenshtein	0.6	225	234	0.897	0.933	0.915	24,240
levenshtein	0.7	225	219	0.89	0.867	0.878	24,231
levenshtein	0.8	225	210	0.89	0.831	0.86	25,670
levenshtein	0.9	225	201	0.886	0.791	0.836	25,073
levenshtein	1.0	225	174	0.868	0.671	0.757	24,340
lcs	0.2	225	118209	0.002	0.947	0.004	19,313
lcs	0.3	225	114834	0.002	0.947	0.004	16,799
lcs	0.4	225	52114	0.004	0.947	0.008	15,287
lcs	0.5	225	521	0.409	0.947	0.571	15,916
lcs	0.6	225	235	0.898	0.938	0.917	15,152
lcs	0.7	225	222	0.892	0.88	0.886	14,887
lcs	0.8	225	210	0.89	0.831	0.86	14,867
lcs	0.9	225	202	0.886	0.796	0.838	17,625
lcs	1.0	225	174	0.868	0.671	0.757	21,725
ratcliff-obershelp	0.2	225	115650	0.002	0.947	0.004	129,979
ratcliff-obershelp	0.3	225	71017	0.003	0.947	0.006	129,758
ratcliff-obershelp	0.4	225	6463	0.033	0.947	0.064	129,387
ratcliff-obershelp	0.5	225	287	0.742	0.947	0.832	129,901
ratcliff-obershelp	**0.6**	225	236	0.898	**0.942**	**0.92**	**130,843**
ratcliff-obershelp	0.7	225	232	0.897	0.924	0.91	144,001
ratcliff-obershelp	0.8	225	224	0.893	0.889	0.891	138,807
ratcliff-obershelp	0.9	225	208	0.889	0.822	0.855	140,794
ratcliff-obershelp	1.0	225	174	0.868	0.671	0.757	129,514

Table 4.1: The result of SPIMBENCH

4.2.1.2 Conclusion

Using the levenshtein algorithm with **0.5** as the threshold, the evaluation got the best precision 0.899 and the best f1 score 0.922. As for the recall rate, almost all algorithms used achieved the same highest value, 0.947. So, it turned out that the levenshtein algorithm is the best algorithm for the task of string similarity. As for the performance benchmark, the calculation time, LCS just needed around 15 seconds, and achieved over 0.9 on both recall and f1 score, which is significantly faster than other algorithms.

After getting above results, It should be noted that the recall rate of some results got the same value 0.947. It is very likely that some golden matching results cannot be matched by calculating their text similarity, so after carefully checked the results that do not match, and found the following several situations.

1. The filed names are not matched, including **title**, **shortTitle** and **description**. Here is an example which has description0 and description1, but the SPARQL did not count them into the linking task.

Listing 4.5: The first example of linking failure

```
1 <http://www.bbc.co.uk/things/348467808#id> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
      type> <http://www.bbc.co.uk/ontologies/creativework/BlogPost> .
2 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /category> <http://www.bbc.co.uk/category/PoliticsPersonsReference> .
3 <http://www.bbc.co.uk/things/348467808#id>
      <http://www.bbc.co.uk/ontologies/creativework/description0> " site's adoption final
      facts be pope usage about" .
4 <http://www.bbc.co.uk/things/348467808#id>
      <http://www.bbc.co.uk/ontologies/creativework/description1> " about olympics
      competed already partisan page." .
5 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://news.bbc.co.uk/democracylive/hi/representatives/profiles/38898.stm>
      .
6 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://rdf.freebase.com/ns/m.0by1vd5> .
7 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://dbpedia.org/resource/Castanheira,_Mato_Grosso> .
8 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://dbpedia.org/resource/1964_Daily_Mirror_Trophy> .
9 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /mentions> <http://sws.geonames.org/7301743/> .
10 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /audience> <http://www.bbc.co.uk/ontologies/creativework/InternationalAudience> .
11 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /liveCoverage> "false"^^<http://www.w3.org/2001/XMLSchema#boolean> .
12 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /primaryFormat> <http://www.bbc.co.uk/ontologies/creativework/TextualFormat> .
13 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /dateCreated> "2011-09-06T08:41:38.303+03:00"^^<http://www.w3.org/2001/XMLSchema#
      dateTime> .
14 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /dateModified> "2012-01-08T12:38:11.303+02:00"^^<http://www.w3.org/2001/XMLSchema#
      dateTime> .
15 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/creativework
      /thumbnail> <http://www.bbc.co.uk/thumbnail/905480047> .
16 <http://www.bbc.co.uk/things/348467808#id> <http://www.bbc.co.uk/ontologies/bbc/
      primaryContentOf> <http://www.bbc.co.uk/things/298047133#id> .
```

2. Another situation is there are not any text fields can be used to match. Here is an example which has no text fields.

Listing 4.6: The second example of linking failure

```
1 <http://www.bbc.co.uk/things/257417947#id> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
      type> <http://www.bbc.co.uk/ontologies/creativework/BlogPost> .
2 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /category> <http://dbpedia.org/ontology/Place> .
3 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://dbpedia.org/resource/Bedford_Township,_Meigs_County,_Ohio> .
4 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://dbpedia.org/resource/Colo-Colo_(women)> .
5 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://google.com/search?btnI&q=site:http://www.guardian.co.uk/politics/
      person/+Adrian+Sanders> .
6 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /about> <http://dbpedia.org/resource/Pasmilgys> .
7 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /mentions> <http://sws.geonames.org/2637861/> .
8 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /audience> <http://www.bbc.co.uk/ontologies/creativework/InternationalAudience> .
9 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /liveCoverage> "false"^^<http://www.w3.org/2001/XMLSchema#boolean> .
10 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /primaryFormat> <http://www.bbc.co.uk/ontologies/creativework/TextualFormat> .
11 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /dateCreated> "2011-08-16T14:10:22.755+03:00"^^<http://www.w3.org/2001/XMLSchema#
      dateTime> .
12 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /dateModified> "2012-06-08T23:43:14.059+03:00"^^<http://www.w3.org/2001/XMLSchema#
      dateTime> .
13 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/creativework
      /thumbnail> <http://www.bbc.co.uk/thumbnail/1210072079> .
14 <http://www.bbc.co.uk/things/257417947#id> <http://www.bbc.co.uk/ontologies/bbc/
      primaryContentOf> <http://www.bbc.co.uk/things/172297071#id> .
```

4.2.2 Spatial Data Matching

The spatial datasets used here are groups by two parts, which are TomTom and Spaten.

- **TomTom** has a Synthetic Trace Generator that makes it easy to create any amount of data from statistics about vehicle traffic. More specifically, it makes traces, which are a list of (longitude, latitude) pairs that one device (phone, car, etc.) records over the course of one day.
- **Spaten** is an open-source, programmable, spatio-temporal and textual dataset generator that can make a lot of data based on how users act in the real world. Using the Google Maps API, Spaten takes GPS tracks from real routes and combines them with real POIs and relevant user comments from TripAdvisor. Spaten creates GB-sized datasets with millions of check-ins and GPS tracks available to the public.

This round of the challenge is related to spatial data, which evaluates the ability to recognize DE-9IM (Dimensionally Extended nine-Intersection Model) topological relations. The supported spatial relations are the following: Equals, Disjoint, Touches, Contains/Within, Covers/CoveredBy, Intersects, Crosses, Overlaps and all traces are represented in Well-Known Text (WKT) format. Each relation will be assigned a unique pair of source and target datasets. This task is used to evaluate the performance of systems that deal with topological connections presented in the cutting-edge DE-9IM (Dimensionally Extended nine-Intersection Paradigm) model, where all topological relations between trajectories are in two-dimensional space. In this work, the task worked on the correct implementation of all the topological relationships of the DE-9IM topological model, as well as the pressure of big datasets in the test. [22].

Spatial task consists of two subtasks:

- **TomTom dataset**

  - Match LineStrings to LineString
  - Match LineStrings to Polygons
- **Spaten dataset**
  - Match LineStrings to LineString
  - Match LineStrings to Polygons

The namespaces used of the above datasets are:

- **TomTom dataset**
  - `http://www.tomtom.com/ontologies/traces#` for the Traces (LineStrings)
  - `http://www.tomtom.com/ontologies/regions#` for the Regions (Polygons)
- **Spaten dataset**
  - `http://www.spaten.com/ontologies/traces#` for the Traces (LineStrings)
  - `http://www.spaten.com/ontologies/regions#`for the Regions (Polygons)

Here is an example of a pair of spatial data.

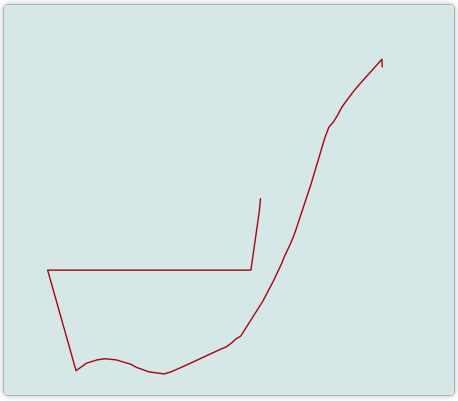

Figure 4.2: The source spatial data

Figure 4.3: The target spatial data

Listing 4.7: An example of the source spatial data

```
1 <http://www.spaten.com/trace-data#490030> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
      http://www.spaten.com/ontologies/traces#Trace> .
2 <http://www.spaten.com/trace-data#490030> <http://strdf.di.uoa.gr/ontology#hasGeometry> "
      LINESTRING (40.79844 -73.4127, 40.79843 -73.41263, 40.79818 -73.41271, 40.79774 -73.41285,
       40.79753 -73.41292, 40.79719 -73.41305, 40.7971 -73.41309, 40.79701 -73.41314, 40.79686
      -73.41321, 40.79671 -73.41326, 40.79665 -73.41331, 40.79658 -73.41336, 40.7965 -73.41344,
      40.79645 -73.41348, 40.79635 -73.41358, 40.79628 -73.41365, 40.79613 -73.41379, 40.79592
      -73.41397, 40.79562 -73.41423, 40.79548 -73.41433, 40.79541 -73.41437, 40.79535 -73.41441,
       40.79525 -73.41447, 40.7952 -73.41451, 40.79505 -73.4146, 40.79492 -73.41468, 40.79472
      -73.41479, 40.79457 -73.41487, 40.79424 -73.41502, 40.79386 -73.41519, 40.79369 -73.41522,
       40.79358 -73.41525, 40.79339 -73.41529, 40.79322 -73.41531, 40.79254 -73.4154, 40.79201
      -73.41547, 40.79161 -73.41552, 40.7915 -73.41553, 40.79138 -73.41554, 40.79088 -73.41552,
      40.79049 -73.41548, 40.7903 -73.41545, 40.78983 -73.41541, 40.78947 -73.4154, 40.78921
      -73.41541, 40.78887 -73.41544, 40.78853 -73.41551, 40.78818 -73.41516, 40.78808 -73.41505,
       40.78798 -73.41495, 40.78774 -73.41471, 40.78761 -73.41458, 40.79419 -73.41458, 40.79427
      -73.41441, 40.79437 -73.41423, 40.79442 -73.41412, 40.79446 -73.41404, 40.79447 -73.414,
      40.79449 -73.41396, 40.7945 -73.41392)"^^<http://strdf.di.uoa.gr/ontology#WKT> .
```

Listing 4.8: An example of the target spatial data

```
<http://www.hobbit.eb53c0825-2aec-42dc-9b72-6863a6144a25> <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#type> <http://www.spaten.com/ontologies/traces#Trace> .
<http://www.hobbit.eb53c0825-2aec-42dc-9b72-6863a6144a25> <http://strdf.di.uoa.gr/ontology#
    hasGeometry> "LINESTRING (40.79457 -73.41487, 40.79424 -73.41502, 40.79386 -73.41519,
    40.79369 -73.41522, 40.79358 -73.41525, 40.79339 -73.41529, 40.79322 -73.41531, 40.79254
    -73.4154, 40.79201 -73.41547, 40.79161 -73.41552, 40.7915 -73.41553, 40.79138 -73.41554,
    40.79088 -73.41552, 40.79049 -73.41548, 40.7903 -73.41545, 40.78983 -73.41541, 40.78947
    -73.4154, 40.78921 -73.41541, 40.78887 -73.41544, 40.78853 -73.41551, 40.78818 -73.41516,
    40.78808 -73.41505, 40.78798 -73.41495, 40.78774 -73.41471, 40.78761 -73.41458, 40.79419
    -73.41458, 40.79427 -73.41441, 40.79437 -73.41423)"^^<http://strdf.di.uoa.gr/ontology#WKT>
     .
```

The relation between the above pair of spatial data is **CONTAINS**. It can be clearly seen from the figures below, the figure on the left contains the figure on the right, the right geometry is party of the left one. The SPARQL used to check the above relationships is shown below:

Listing 4.9: A sparql of the Spatial task

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>

CONSTRUCT { ?source ?target "1.0"^^xsd:decimal. }
WHERE {
  ?source geometry:isLocatedAt "sourceCONTAINS-0001.nt";
    strdf:hasGeometry ?sourceGeometry.
  ?target geometry:isLocatedAt "targetCONTAINS-0001.nt";
    strdf:hasGeometry ?targetGeometry.
  BIND(geometry:contains(?sourceGeometry, ?targetGeometry) AS ?relation)
  FILTER(?relation = "true"^^xsd:boolean)
}
```

4.2.2.1 Results

This section presents the results of all spatial data matching.

Dataset	Type	Relation	Golden size	Result size	Precision	Recall	F1	Milliseconds
TomTom	LinesTOPolygons	covers	256	256	1.0	1.0	1.0	428
TomTom	LinesTOPolygons	coveredBy	256	256	1.0	1.0	1.0	167
TomTom	LinesTOPolygons	disjoint	400	400	1.0	1.0	1.0	225
TomTom	LinesTOPolygons	contains	256	256	1.0	1.0	1.0	127
TomTom	LinesTOPolygons	touches	20	20	1.0	1.0	1.0	59
TomTom	LinesTOPolygons	crosses	23	23	1.0	1.0	1.0	124
TomTom	LinesTOPolygons	intersects	22	22	1.0	1.0	1.0	123
TomTom	LinesTOPolygons	within	256	256	1.0	1.0	1.0	121
Spaten	LinesTOLines	covers	20	20	1.0	1.0	1.0	25
Spaten	LinesTOLines	coveredBy	61	61	1.0	1.0	1.0	97
Spaten	LinesTOLines	overlaps	19	19	1.0	1.0	1.0	82
Spaten	LinesTOLines	disjoint	400	400	1.0	1.0	1.0	50
Spaten	LinesTOLines	contains	20	20	1.0	1.0	1.0	23
Spaten	LinesTOLines	equals	20	20	1.0	1.0	1.0	32
Spaten	LinesTOLines	touches	19	19	1.0	1.0	1.0	66
Spaten	LinesTOLines	crosses	20	20	1.0	1.0	1.0	48
Spaten	LinesTOLines	intersects	20	20	1.0	1.0	1.0	58
Spaten	LinesTOLines	within	61	61	1.0	1.0	1.0	94
TomTom	LinesTOLines	covers	20	20	1.0	1.0	1.0	64
TomTom	LinesTOLines	coveredBy	60	60	1.0	1.0	1.0	229

TomTom	LinesTOLines	overlaps	20	20	1.0	1.0	1.0	181
TomTom	LinesTOLines	disjoint	400	400	1.0	1.0	1.0	149
TomTom	LinesTOLines	contains	20	20	1.0	1.0	1.0	72
TomTom	LinesTOLines	equals	20	20	1.0	1.0	1.0	88
TomTom	LinesTOLines	touches	20	20	1.0	1.0	1.0	568
TomTom	LinesTOLines	crosses	21	21	1.0	1.0	1.0	146
TomTom	LinesTOLines	intersects	29	29	1.0	1.0	1.0	386
TomTom	LinesTOLines	within	60	60	1.0	1.0	1.0	219
Spaten	LinesTOPolygons	covers	24	24	1.0	1.0	1.0	39
Spaten	LinesTOPolygons	coveredBy	24	24	1.0	1.0	1.0	39
Spaten	LinesTOPolygons	disjoint	400	400	1.0	1.0	1.0	47
Spaten	LinesTOPolygons	contains	24	24	1.0	1.0	1.0	40
Spaten	LinesTOPolygons	touches	20	20	1.0	1.0	1.0	20
Spaten	LinesTOPolygons	crosses	19	19	1.0	1.0	1.0	42
Spaten	LinesTOPolygons	intersects	20	20	1.0	1.0	1.0	41

Table 4.2: The result of spatial data matching

4.2.2.2 Conclusion

After checking the results, it is obviously seeing those spatial data were perfectly mapped to each other, and the running time also fast. This shows that the algorithm performs very well when faced with data matching tasks with complete data formats.

Chapter 5

Future Work

The chapter is about future work. It shows the future work that will be done in the future. As previously described, there are strong interests in this type of tool for linking graphs in the market or in research institutions, so there is a lot of work to be performed in the future for this project so that more people can benefit and increase their efficiency and effectiveness.

5.1 Resolving Problems in the SPIMBENCH

As mentioned in the previous chapter, the result shown that many algorithms achieved the same recall rate during the SPIMBENCH task, and after researching the author of this TFM found it was because the field names in the target dataset did not match or there was no literal text data at all. So based on these two different problem scenarios, the possible solutions for the future work are the following ways:

1. Adding more fields to match;
2. Comparing the similarity of each field names;
3. Looking up each linking objects, and comparing their similarity;
4. Using NLP techniques etc.

5.2 Uploading Artifacts to The Maven Central Repository

LOOM-LD is an application using Maven as a build tool, so this application can upload the artifacts to the Maven Central Repository (MCR) to allow other developers to use it. Besides, it is also an open-source project on GitHub, more programmers who share the same interests in development can work together.

5.3 Deploying as A Service

Using Apache Jena to build an online service that allows other researchers to easily link ontologies with the standard SPARQL queries.

5.4 Implementing A SPARQL Generator

Unfortunately, for most people without a scientific background or industry experience, the general SPARQL syntax is still not cheap to learn, developing a tool that automatically creates or aids in the generation of SPARQL queries for linking ontologies by using these algorithms is quite necessary because these queries do not have to be written manually.

Bibliography

[1] *RDF*. [Online]. Available: https://www.w3.org/RDF/

[2] *Apache Jena – Getting started with Apache Jena*. [Online]. Available: https://jena.apache.org/getting_started/index.html

[3] *An Introduction to RDF and the Jena RDF API*. [Online]. Available: https://jena.apache.org/tutorials/rdf_api.html

[4] *Jena Ontology API*. [Online]. Available: https://jena.apache.org/documentation/ontology/

[5] *ARQ – A SPARQL Processor for Jena*. [Online]. Available: https://jena.apache.org/documentation/query/

[6] *ARQ – Application API*. [Online]. Available: https://jena.apache.org/documentation/query/app_api.html

[7] *ARQ – Writing Filter Functions*. [Online]. Available: https://jena.apache.org/documentation/query/writing_functions.html

[8] *Reasoners and rule engines: Jena inference support*. [Online]. Available: https://jena.apache.org/documentation/inference/

[9] *Apache Jena Fuseki*. [Online]. Available: https://jena.apache.org/documentation/fuseki2/

[10] *TDB*. [Online]. Available: https://jena.apache.org/documentation/tdb/index.html

[11] Prasetya, D., Wibawa, A. & Hirashima, T. The performance of text similarity algorithms. *International Journal Of Advances In Intelligent Informatics*. **4** (2018,5)

[12] Levenshtein, V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*. **10** pp. 707 (1966,2)

[13] Winkler, W. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. *Proceedings Of The Section On Survey Research Methods*. (1990,1)

[14] Jaro, M. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal Of The American Statistical Association*. **84**, 414-420 (1989), https://www.tandfonline.com/doi/abs/10.1080/01621459.1989.10478785

[15] John W. Ratcliff and David E. Metzener, Pattern Matching: The Gestalt Approach. *Dr. Dobb's Journal*. 46, (1988, 7)

[16] Murphy, A. The Finley Affair: A Signal Event in the History of Forecast Verification. *Weather And Forecasting*. **11**, 3-20 (1996,3)

[17] Debatty, T., *Implementation of various string similarity and distance algorithms*. [online] GitHub. Available: https://github.com/tdebatty/java-string-similarity

[18] J.Herring (2011, May. 28). *Simple Feature Access – Part 1: Common Architecture* [Online]. Available: https://www.ogc.org/standards/sfa

[19] Clementini, E., Di Felice, P. & Oosterom, P. A small set of formal topological relationships suitable for end-user interaction. *Advances In Spatial Databases*. pp. 277-295 (1993)

[20] LocationTech. *JTS Topology Suite*. [online] GitHub. Available: https://github.com/locationtech/jts

[21] *Ontology Alignment Evaluation Initiative*. [online]. Available: http://oaei.ontologymatching.org/

[22] *OAEI 2021 Instance Matching or Link Discovery*. [online]. Available: https://hobbit-project.github.io/OAEI_2021.html/

[23] Karlruhe. U, “Ontology Alignment Evaluation Initiative 1", (2005)

[24] Do. H, Melnik. S, & Rahm. E, “Comparison of Schema Matching Evaluations,” in *Web, Web-Services, And Database Systems*, 2003, pp. 221-237

Annex

.1 The SPARQLs for the SPIMBENCH Task

Listing 1: A Sparql of getting the source data

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
  ?thing rdf:type ?work .
  ?thing cwork:title ?title .
  ?thing cwork:shortTitle ?shortTitle .
  ?thing cwork:description ?description .
  ?thing seals:isLocatedAt "tbox1.nt"
}
WHERE {
  ?thing rdf:type ?work .
  OPTIONAL {?thing cwork:title ?title .}
  OPTIONAL {?thing cwork:shortTitle ?shortTitle .}
  OPTIONAL {?thing cwork:description ?description .}
}
```

Listing 2: A Sparql of getting the target data

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?thing rdf:type ?work .
    ?thing cwork:title ?title .
    ?thing cwork:shortTitle ?shortTitle .
    ?thing cwork:description ?description .
    ?thing seals:isLocatedAt "tbox2.nt"
}
WHERE {
    ?thing rdf:type ?work .
    OPTIONAL {?thing cwork:title ?title .}
    OPTIONAL {?thing cwork:shortTitle ?shortTitle .}
    OPTIONAL {?thing cwork:description ?description .}
}
```

Listing 3: A Sparql of getting the golden data

```
PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
```

```
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?entity1 owl:sameAs ?entity2 .
}
WHERE {
    ?alignment heterogeneity:alignmententity1 ?entity1.
    ?alignment heterogeneity:alignmententity2 ?entity2.
    FILTER ( ?entity1 != ?entity2)
}
```

Listing 4: A Sparql of the LCS similarity function

```
PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?thing1 owl:sameAs ?thing2 .
}
WHERE {
    ?thing1 seals:isLocatedAt "tbox1.nt" .
    OPTIONAL {?thing1 cwork:title ?title1 .}
    OPTIONAL {?thing1 cwork:shortTitle ?shortTitle1 .}
    OPTIONAL {?thing1 cwork:description ?description1 .}
    ?thing2 seals:isLocatedAt "tbox2.nt" .
    OPTIONAL {?thing2 cwork:title ?title2 .}
    OPTIONAL {?thing2 cwork:shortTitle ?shortTitle2 .}
    OPTIONAL {?thing2 cwork:description ?description2 .}
    BIND(loom:lcs(?title1, ?title2 ) AS ?titleGrade)
    BIND(loom:lcs(?shortTitle1, ?shortTitle2 ) AS ?shortTitleGrade)
    BIND(loom:lcs(?description1, ?description2 ) AS ?descriptionGrade)
    FILTER ( ?thing1 != ?thing2 )
    FILTER ( ?titleGrade >= 0.2 || ?shortTitleGrade >=  0.2 || ?descriptionGrade >=  0.2 )
}
```

Listing 5: A Sparql of the Ratcliff Obershelp similarity function

```

PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?thing1 owl:sameAs ?thing2 .
}
WHERE {
    ?thing1 seals:isLocatedAt "tbox1.nt" .
    OPTIONAL {?thing1 cwork:title ?title1 .}
    OPTIONAL {?thing1 cwork:shortTitle ?shortTitle1 .}
    OPTIONAL {?thing1 cwork:description ?description1 .}
    ?thing2 seals:isLocatedAt "tbox2.nt" .
    OPTIONAL {?thing2 cwork:title ?title2 .}
    OPTIONAL {?thing2 cwork:shortTitle ?shortTitle2 .}
    OPTIONAL {?thing2 cwork:description ?description2 .}
    BIND(loom:ratcliff-obershelp(?title1, ?title2 ) AS ?titleGrade)
    BIND(loom:ratcliff-obershelp(?shortTitle1, ?shortTitle2 ) AS ?shortTitleGrade)
    BIND(loom:ratcliff-obershelp(?description1, ?description2 ) AS ?descriptionGrade)
    FILTER ( ?thing1 != ?thing2 )
    FILTER ( ?titleGrade >= 0.2 || ?shortTitleGrade >=  0.2 || ?descriptionGrade >=  0.2 )
```

```
}
```

Listing 6: A Sparql of the Cosine similarity function

```
PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?thing1 owl:sameAs ?thing2 .
}
WHERE {
    ?thing1 seals:isLocatedAt "tbox1.nt" .
    OPTIONAL {?thing1 cwork:title ?title1 .}
    OPTIONAL {?thing1 cwork:shortTitle ?shortTitle1 .}
    OPTIONAL {?thing1 cwork:description ?description1 .}
    ?thing2 seals:isLocatedAt "tbox2.nt" .
    OPTIONAL {?thing2 cwork:title ?title2 .}
    OPTIONAL {?thing2 cwork:shortTitle ?shortTitle2 .}
    OPTIONAL {?thing2 cwork:description ?description2 .}
    BIND(loom:cosine(?title1, ?title2 ) AS ?titleGrade)
    BIND(loom:cosine(?shortTitle1, ?shortTitle2 ) AS ?shortTitleGrade)
    BIND(loom:cosine(?description1, ?description2 ) AS ?descriptionGrade)
    FILTER ( ?thing1 != ?thing2 )
    FILTER ( ?titleGrade >= 0.3 || ?shortTitleGrade >=  0.3 || ?descriptionGrade >=  0.3 )
}
```

Listing 7: A Sparql of the Jaccard similarity function

```
PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?thing1 owl:sameAs ?thing2 .
}
WHERE {
    ?thing1 seals:isLocatedAt "tbox1.nt" .
    OPTIONAL {?thing1 cwork:title ?title1 .}
    OPTIONAL {?thing1 cwork:shortTitle ?shortTitle1 .}
    OPTIONAL {?thing1 cwork:description ?description1 .}
    ?thing2 seals:isLocatedAt "tbox2.nt" .
    OPTIONAL {?thing2 cwork:title ?title2 .}
    OPTIONAL {?thing2 cwork:shortTitle ?shortTitle2 .}
    OPTIONAL {?thing2 cwork:description ?description2 .}
    BIND(loom:jaccard(?title1, ?title2 ) AS ?titleGrade)
    BIND(loom:jaccard(?shortTitle1, ?shortTitle2 ) AS ?shortTitleGrade)
    BIND(loom:jaccard(?description1, ?description2 ) AS ?descriptionGrade)
    FILTER ( ?thing1 != ?thing2 )
    FILTER ( ?titleGrade >= 0.2 || ?shortTitleGrade >=  0.2 || ?descriptionGrade >=  0.2 )
}
```

Listing 8: A Sparql of the Jaro Winkler similarity function

```
PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
```

```
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?thing1 owl:sameAs ?thing2 .
}
WHERE {
    ?thing1 seals:isLocatedAt "tbox1.nt" .
    OPTIONAL {?thing1 cwork:title ?title1 .}
    OPTIONAL {?thing1 cwork:shortTitle ?shortTitle1 .}
    OPTIONAL {?thing1 cwork:description ?description1 .}
    ?thing2 seals:isLocatedAt "tbox2.nt" .
    OPTIONAL {?thing2 cwork:title ?title2 .}
    OPTIONAL {?thing2 cwork:shortTitle ?shortTitle2 .}
    OPTIONAL {?thing2 cwork:description ?description2 .}
    BIND(loom:jaro-winkler(?title1, ?title2 ) AS ?titleGrade)
    BIND(loom:jaro-winkler(?shortTitle1, ?shortTitle2 ) AS ?shortTitleGrade)
    BIND(loom:jaro-winkler(?description1, ?description2 ) AS ?descriptionGrade)
    FILTER ( ?thing1 != ?thing2 )
    FILTER ( ?titleGrade >= 0.3 || ?shortTitleGrade >=  0.3 || ?descriptionGrade >=  0.3 )
}
```

Listing 9: A Sparql of the Levenshtein similarity function

```
PREFIX heterogeneity: <http://knowledgeweb.semanticweb.org/heterogeneity/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
PREFIX loom:     <https://oeg.upm.es/loom-ld/functions/linking/text#>
PREFIX cwork:    <http://www.bbc.co.uk/ontologies/creativework/>
PREFIX seals:    <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
CONSTRUCT {
    ?thing1 owl:sameAs ?thing2 .
}
WHERE {
    ?thing1 seals:isLocatedAt "tbox1.nt" .
    OPTIONAL {?thing1 cwork:title ?title1 .}
    OPTIONAL {?thing1 cwork:shortTitle ?shortTitle1 .}
    OPTIONAL {?thing1 cwork:description ?description1 .}
    ?thing2 seals:isLocatedAt "tbox2.nt" .
    OPTIONAL {?thing2 cwork:title ?title2 .}
    OPTIONAL {?thing2 cwork:shortTitle ?shortTitle2 .}
    OPTIONAL {?thing2 cwork:description ?description2 .}
    BIND(loom:levenshtein(?title1, ?title2 ) AS ?titleGrade)
    BIND(loom:levenshtein(?shortTitle1, ?shortTitle2 ) AS ?shortTitleGrade)
    BIND(loom:levenshtein(?description1, ?description2 ) AS ?descriptionGrade)
    FILTER ( ?thing1 != ?thing2 )
    FILTER ( ?titleGrade >= 0.2 || ?shortTitleGrade >=  0.2 || ?descriptionGrade >=  0.2 )
}
```

.2 The SPARQLs for the Spatial Data Matching

Listing 10: A Sparql of getting source data

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?source strdf:hasGeometry ?sourceGeometry .
    ?source geometry:isLocatedAt "sourceCOVERS-0001.nt"
}
WHERE {
    ?source strdf:hasGeometry ?sourceGeometry .
}
```

Listing 11: A Sparql of getting target data

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?target strdf:hasGeometry ?targetGeometry .
    ?target geometry:isLocatedAt "targetCOVERS-0001.nt"
}
WHERE {
    ?target strdf:hasGeometry ?targetGeometry .
}
```

Listing 12: A Sparql of checking the covers relation

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?source ?target 1.0
}
WHERE {
    ?source geometry:isLocatedAt "sourceCOVERS-0001.nt" .
    ?source strdf:hasGeometry ?sourceGeometry .
    ?target geometry:isLocatedAt "targetCOVERS-0001.nt" .
    ?target strdf:hasGeometry ?targetGeometry .
    BIND ( geometry:covers (?sourceGeometry, ?targetGeometry ) AS ?relation )
    FILTER ( ?relation = TRUE )
}
```

Listing 13: A Sparql of checking the coveredBy relation

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?source ?target 1.0
}
WHERE {
    ?source geometry:isLocatedAt "sourceCOVERED_BY-0001.nt" .
    ?source strdf:hasGeometry ?sourceGeometry .
    ?target geometry:isLocatedAt "targetCOVERED_BY-0001.nt" .
    ?target strdf:hasGeometry ?targetGeometry .
    BIND ( geometry:coveredBy (?sourceGeometry, ?targetGeometry ) AS ?relation )
    FILTER ( ?relation = TRUE )
}
```

Listing 14: A Sparql of checking the disjoint relation

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?source ?target 1.0
}
WHERE {
    ?source geometry:isLocatedAt "sourceDISJOINT-0001.nt" .
    ?source strdf:hasGeometry ?sourceGeometry .
    ?target geometry:isLocatedAt "targetDISJOINT-0001.nt" .
    ?target strdf:hasGeometry ?targetGeometry .
    BIND ( geometry:disjoint (?sourceGeometry, ?targetGeometry ) AS ?relation )
    FILTER ( ?relation = TRUE )
}
```

Listing 15: A Sparql of checking the contains relation

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?source ?target 1.0
}
WHERE {
    ?source geometry:isLocatedAt "sourceCONTAINS-0001.nt" .
    ?source strdf:hasGeometry ?sourceGeometry .
    ?target geometry:isLocatedAt "targetCONTAINS-0001.nt" .
    ?target strdf:hasGeometry ?targetGeometry .
    BIND ( geometry:contains (?sourceGeometry, ?targetGeometry ) AS ?relation )
    FILTER ( ?relation = TRUE )
}
```

Listing 16: A Sparql of checking the equals relation

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?source ?target 1.0
}
WHERE {
    ?source geometry:isLocatedAt "sourceEQUALS-0001.nt" .
    ?source strdf:hasGeometry ?sourceGeometry .
    ?target geometry:isLocatedAt "targetEQUALS-0001.nt" .
    ?target strdf:hasGeometry ?targetGeometry .
    BIND ( geometry:equals (?sourceGeometry, ?targetGeometry ) AS ?relation )
    FILTER ( ?relation = TRUE )
}
```

Listing 17: A Sparql of checking the intersects relation

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
CONSTRUCT {
    ?source ?target 1.0
}
WHERE {
    ?source geometry:isLocatedAt "sourceINTERSECTS-0001.nt" .
    ?source strdf:hasGeometry ?sourceGeometry .
    ?target geometry:isLocatedAt "targetINTERSECTS-0001.nt" .
    ?target strdf:hasGeometry ?targetGeometry .
    BIND ( geometry:intersects (?sourceGeometry, ?targetGeometry ) AS ?relation )
    FILTER ( ?relation = TRUE )
```

```
14 }
```

Listing 18: A Sparql of checking the overlaps relation

```
1 PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
3 PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
4 CONSTRUCT {
5     ?source ?target 1.0
6 }
7 WHERE {
8     ?source geometry:isLocatedAt "sourceOVERLAPS-0001.nt" .
9     ?source strdf:hasGeometry ?sourceGeometry .
10    ?target geometry:isLocatedAt "targetOVERLAPS-0001.nt" .
11    ?target strdf:hasGeometry ?targetGeometry .
12    BIND ( geometry:overlaps (?sourceGeometry, ?targetGeometry ) AS ?relation )
13    FILTER ( ?relation = TRUE )
14 }
```

Listing 19: A Sparql of checking the touches relation

```
1 PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
3 PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
4 CONSTRUCT {
5     ?source ?target 1.0
6 }
7 WHERE {
8     ?source geometry:isLocatedAt "sourceTOUCHES-0001.nt" .
9     ?source strdf:hasGeometry ?sourceGeometry .
10    ?target geometry:isLocatedAt "targetTOUCHES-0001.nt" .
11    ?target strdf:hasGeometry ?targetGeometry .
12    BIND ( geometry:touches (?sourceGeometry, ?targetGeometry ) AS ?relation )
13    FILTER ( ?relation = TRUE )
14 }
```

Listing 20: A Sparql of checking the crosses relation

```
1     PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2     PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
3     PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
4     CONSTRUCT {
5        ?source ?target 1.0
6     }
7     WHERE {
8       ?source geometry:isLocatedAt "sourceCROSSES-0001.nt" .
9       ?source strdf:hasGeometry ?sourceGeometry .
10      ?target geometry:isLocatedAt "targetCROSSES-0001.nt" .
11      ?target strdf:hasGeometry ?targetGeometry .
12      BIND ( geometry:crosses (?sourceGeometry, ?targetGeometry ) AS ?relation )
13      FILTER ( ?relation = TRUE )
14    }
```

Listing 21: A Sparql of checking the within relation

```
1 PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX geometry:    <https://oeg.upm.es/loom-ld/functions/linking/geometry#>
3 PREFIX strdf:     <http://strdf.di.uoa.gr/ontology#>
4 CONSTRUCT {
5     ?source ?target 1.0
6 }
7 WHERE {
8     ?source geometry:isLocatedAt "sourceWITHIN-0001.nt" .
```

```
9       ?source strdf:hasGeometry ?sourceGeometry .
10      ?target geometry:isLocatedAt "targetWITHIN-0001.nt" .
11      ?target strdf:hasGeometry ?targetGeometry .
12      BIND ( geometry:within (?sourceGeometry, ?targetGeometry ) AS ?relation )
13      FILTER ( ?relation = TRUE )
14  }
```