HOBBIT

Collaborative Project

Holistic Benchmarking of Big Linked Data

Project Number: 688227 **Start Date of Project:** 2015/12/01 **Duration:** 36 months

Deliverable 4.1.1
First version of the Linking Benchmark

Dissemination Level	Public
Due Date of Deliverable	Month 18, 31/05/2017
Actual Submission Date	Month 18, 31/05/2017
Work Package	WP4 - Linking
Task	T4.1
Type	Report
Approval Status	Final
Version	1.0
Number of Pages	22

Abstract:This deliverable presents the first version of the linking benchmark for HOBBIT Platform.

Project funded by the European Commission's Horizon 2020 Program.

History

Version	Date	Reason	Revised by
0.0	24/04/2017	First draft created	Tzanina Saveta
0.1	05/05/2017	Draft revised	Giorgos Flouris
0.2	05/05/2017	Draft revised	Tzanina Saveta
0.3	05/05/2017	Draft revised	Irini Fundulaki
0.3	12/05/2017	Peer Review Comments	Axel-Cyrille Ngonga Ngomo
0.4	19/05/2017	Draft revised	Tzanina Saveta
0.5	22/05/2017	Final Draft	Irini Fundulaki

Author List

Organization	Name	Contact Information
FORTH	Tzanina Saveta	jsaveta@ics.forth.gr
FORTH	Irini Fundulaki	fundul@ics.forth.gr
FORTH	Giorgos Flouris	fgeo@ics.forth.gr

Executive Summary

The number of datasets published in the Web of Data as part of the Linked Data Cloud is constantly increasing. The Linked Data paradigm is based on the unconstrained publication of information by different publishers, and the inter-linking of Web resources across knowledge bases. In most cases, the cross-dataset links are not explicit in the dataset and must be automatically determined using *link discovery* tools amongst others [13]. The large variety of link discovery techniques requires their comparative evaluation to determine which one is best suited for a given context. Performing such an assessment generally requires well-defined and widely accepted *benchmarks* to determine the weak and strong points of the proposed techniques and/or tools.

A number of real and synthetic benchmarks that address different data challenges have been proposed for evaluating the performance of such systems. So far, only a limited number of *link discovery benchmarks* target the problem of linking geo-spatial entities. However, some of the largest knowledge bases on the Linked Open Data Web are geo-spatial knowledge bases (e.g., LinkedGeoData). We believe that due to the large amount of available geo-spatial datasets employed in Linked Data and in several domains, it is critical that benchmarks for geo-spatial link discovery are developed. The present deliverable proposes two *benchmark generators* that deal with link discovery for spatial data: the *linking Benchmark* generator that is based on SPIMBENCH, to test the performance of Instance Matching tools that implement string-based approaches for identifying matching entities and the *Spatial Benchmark* that can be used to test the performance of systems that deal with topological relations proposed in the state of the art DE-9IM (Dimensionally Extended nine-Intersection Model) model. Both benchmarks are generic in the sense that *schema agnostic*: they can operate with any datasets that contain *trajectories*, a trajectory being a *set of points* or *set of longitude, latitude pairs*. In this deliverable we use the TomTom datasets provided in the context of the HOBBIT project to present both benchmarks.

Contents

1 Introduction

The number of datasets published in the Web of Data as part of the Linked Data Cloud is constantly increasing. The Linked Data paradigm is based on the unconstrained publication of information by different publishers, and the inter-linking of Web resources across knowledge bases. In most cases, the cross-dataset links are not explicit in the dataset and must be automatically determined using *link discovery* tools amongst others [13]. A special case of link discovery is *instance matching*, also known as record linkage [12], duplicate detection [7], entity resolution [19, 5, 4], deduplication [15], merge-purge [10], entity-identification [2], object identification [14], and data fusion [3]) where systems are called to create `owl:sameAs` links between resources that refer to the same real world object. The large variety of techniques requires their comparative evaluation to determine which one is best suited for a given context. Performing such an assessment generally requires well-defined and widely accepted *benchmarks* to determine the weak and strong points of the proposed techniques and/or tools.

A number of real and synthetic benchmarks that address different data challenges have been proposed for evaluating the performance of such systems [16]. So far, only a limited number of *link discovery benchmarks* target the problem of linking geo-spatial entities. However, some of the largest knowledge bases on the Linked Open Data Web are geo-spatial knowledge bases (e.g., LinkedGeoData,[1] with more than 30 billion triples). Linking spatial resources requires techniques that differ from the classical mostly string-based approaches. In particular, considering the topology of the spatial resources and the topological relations between them is of central importance to systems that manage spatial data.

We believe that due to the large amount of available geo-spatial datasets employed in Linked Data and in several domains, it is critical that benchmarks for geo-spatial link discovery are developed. The present deliverable proposes two *benchmark generators* that deal with link discovery for spatial data:

- the *Linking Benchmark* generator that is based on SPIMBENCH[16], to test the performance of Instance Matching tools that implement string-based approaches for identifying matching entities
- the *Spatial Benchmark* that can be used to test the performance of systems that deal with topological relations proposed in the state of the art DE-9IM (Dimensionally Extended nine-Intersection Model) model [18].

The first benchmark is simple and can be used not only by instance matching tools, but also by SPARQL engines that deal with query answering over geo-spatial data such as STRABON [11][2]. The second is more complex and implements all topological relations of DE-9IM between *trajectories* in the two dimensional space. To the best of our knowledge such a generic benchmark, that takes as input trajectories (in its first version) and checks the performance of linking systems for spatial data does not exist. Both benchmarks are generic in the sense that *schema agnostic*: they can operate with any datasets that contain *trajectories*, a trajectory being a *set of points* or *set of longitude, latitude pairs*. In this deliverable we use the TomTom datasets[3] provided in the context of the HOBBIT project to present both benchmarks.

[1] http://linkedgeodata.org/About
[2] http://www.strabon.di.uoa.gr/
[3] https://www.tomtom.com/en_gr/

2 Dataset and Ontology

In this Section we present the ontology and datasets used for testing both the *linking* and *spatial* benchmarks. As noted in Section 1 both benchmarks are *schema agnostic* and can work with trajectories i.e., sets of pairs of longitude, latitude.

TomTom's traffic data archive contains approximately 15 trillion ($1.5 \cdot 10^{13}$) speed measurements from hundreds of millions of road segments all over the world, mostly based on anonymized GPS positioning data collected with user's consent. This data has been used to extract the recurrent traffic patterns and adds this to digital navigation maps as a map layer. Statistical traffic data also form an important input set for TomTom's live traffic fusion engine. Probe data is being used for custom-made traffic and demand analysis. Furthermore, archived positioning data is used to improve digital maps, road geometry and map attributes.

The ontology we use to represent TomTom's data is shown in Figure 1 and its code presented in Listing 1. The main class is `Trace` that contains one or more points (class `Point`). Class `Point` is a subclass of `wgs84_pos:Point` class of the WGS84 Geo Positioning (geo)[4] vocabulary that represents latitude, longitude and altitude information in the WGS84 geodetic reference datum. Each point is associated with a *velocity* (class `Velocity`), instances of which have properties `velocityMetric` and `velocityValue`, the former taking values from a predefined set (*km_per_hour*, *miles_per_hour*) and the later from class `xsd:Float`. A point also has attribute `hasTimeStamp` that takes its values in class `xsd:TimeStamp` which designates the time an object was at this specific point. Listing 2 shows an example of a Trace consisting of 4 Points.

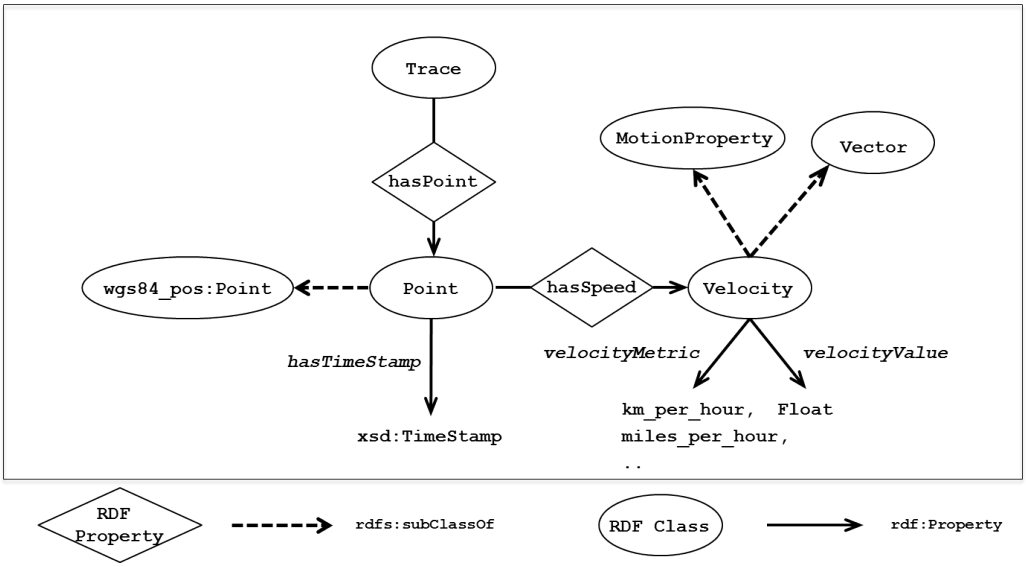


Figure 1: TomTom Ontology

[4] http://lov.okfn.org/dataset/lov/vocabs/geo

```
@prefix : <http://www.tomtom.com/ontologies/traces#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://www.tomtom.com/ontologies/traces> .

<http://www.tomtom.com/ontologies/traces> rdf:type owl:Ontology ;
owl:imports <http://sweet.jpl.nasa.gov/2.0/physDynamics.owl> ,
<https://www.w3.org/2003/01/geo/wgs84_pos#> .

:Trace rdf:type owl:Class .

:Point rdf:type owl:Class ;
    rdfs:subClassOf <http://www.w3.org/2003/01/geo/wgs84_pos#Point> .

:hasTimestamp rdf:type owl:AnnotationProperty ;
    rdfs:range xsd:dateTimeStamp ;
    rdfs:domain :Point .

:hasPoint rdf:type owl:ObjectProperty ;
    rdfs:range :Point ;
    rdfs:domain :Trace .

:hasSpeed rdf:type owl:ObjectProperty ;
    rdfs:range <http://sweet.jpl.nasa.gov/2.0/physDynamics.owl#Velocity> ;
    rdfs:domain :Point .

:velocityValue rdf:type owl:DataTypeProperty;
    rdfs:range xsd:Float ;
    rdfs:domain :Velocity .

:velocityMetric rdf:type owl:ObjectProperty;
    owl:oneOf (:killimeters_perHour :miles_perHour) .
```

Listing 1: TomTom Ontology

```
@base <http://www.tomtom.com/trace-data/0000000001.ttl> .
@prefix : <http://www.tomtom.com/ontologies/traces#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<#trace> a :Trace .
<#trace> :hasPoint <#point0> .
<#point0> :hasTimestamp "2010-03-12T10:19:00.000000"^^xsd:dateTime ;
    :lat 40.898320 ;
    :long 14.185080 ;
    :hasSpeed <#speed0> .
<#speed0> :velocityValue 7.22 ;
    :velocityMetric :kilometers_perHour .
<#trace> :hasPoint <#point1> .
<#point1> :hasTimestamp "2010-03-12T10:19:10.000000"^^xsd:dateTime ;
    :lat 40.898560 ;
    :long 14.184440 ;
    :hasSpeed <#speed1> .
<#speed1> :velocityValue 6.67 ;
    :velocityMetric :kilometers_perHour .
<#trace> :hasPoint <#point2> .
<#point2> :hasTimestamp "2010-03-12T10:19:20.000000"^^xsd:dateTime ;
    :lat 40.898950 ;
    :long 14.184130 ;
    :hasSpeed <#speed2> .
<#speed2> :velocityValue 5.28 ;
    :velocityMetric :kilometers_perHour .
<#trace> :hasPoint <#point3> .
<#point3> :hasTimestamp "2010-03-12T10:19:30.000000"^^xsd:dateTime ;
    :lat 40.899410 ;
    :long 14.184080 ;
    :hasSpeed <#speed3> .
<#speed3> :velocityValue 5.28 ;
    :velocityMetric :kilometers_perHour .
```

Listing 2: TomTom Data

3 Linking Benchmark

3.1 Overview

We built upon SPIMBENCH [16], the instance matching benchmark developed in the context of the LDBC project[5], HOBBIT's Linking Benchmark[6].

The *choke points* for this benchmark are a subset of the ones that were used for the development of SPIMBENCH. SPIMBENCH, in addition to complex *value-based*, *structure-based* and *semantics-aware* transformations. The ontologies used to represent trajectories are fairly simple, and do not consider complex RDF or OWL schema constructs already supported by SPIMBENCH. Nevertheless, since Linking Benchmark is based on SPIMBENCH, in the case in which the ontology is complex, introducing semantics-aware modifications is straightforward.

The test cases implemented in the benchmark focus on *string-based transformations* with different

- *levels*
- *types of spatial object representations* and
- *types of date representations*.

Furthermore, the benchmark supports *addition* and *deletion* of ontology (schema) properties, known also as *schema transformations*[6]. The datasets that implement those test cases can be used by Instance Matching tools to identify matching entities.

In a nutshell, the benchmark can be used to check whether two traces with their *points annotated with place names* designate the same trajectory. The Linking Benchmark gets an *input dataset* that consists of various traces, a *trace being a sequence of points*. The points are expressed using standard longitude/latitude coordinates, and the linking benchmark produces the source and target datasets from the input dataset one as follows:

1. certain percentage of the points is replaced by their associated placenames retrieved from appropriate datasets using respective APIs
2. the placenames are altered using string-based transformations
3. a persentage of longitude/latitude points that were not modified in the source dataset are again transformed in placenames or in a different coordinate system format
4. a percentage of dates are transformed in different date formats

Each trace from the source dataset along with the transformed trace from the target dataset is stored in the gold standard.

In the case that the input dataset already contains points annotated with placenames, steps (1) and (3) above are not necessary. As shown in Figure 2, the benchmark generator gets as input a set of traces and produces a *source* dataset, a *target* dataset that implements the test cases discussed previously according to the specified configuration parameters and a *gold standard* as discussed before. The benchmark generator makes use of the *Initialization*, *Resource Generator* and the *Resource Transformation* modules:

- The *Initialization Module* reads the test case generation parameters and retrieves by means of SPARQL queries each trace instance.
- The *Resource Generator* uses the trace instances retrieved from the Initialization Module the generation of the *source dataset*.

[5] http://ldbcouncil.org/
[6] https://github.com/hobbit-project/LinkingBenchmark

- The *Resource Transformation Module* gets as input a source instance s_i and stores in the *target dataset* the transformed instance t_i. This module is also responsible for producing an entry in the gold standard.

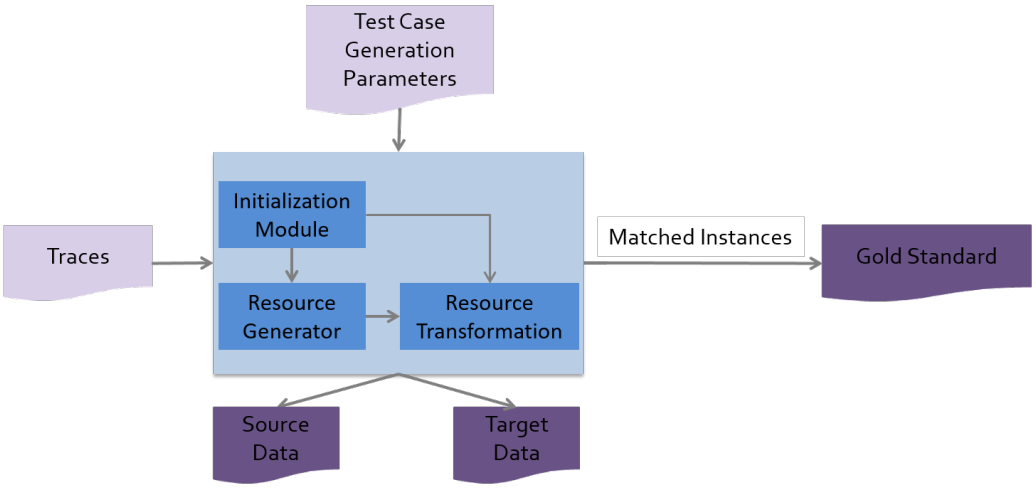


Figure 2: Linking benchmark architecture

3.2 Source, Target and Gold Standard Generation

Below we describe in detail the Linking Benchmark (a) parameters (b) source, target and gold standard generation.

Benchmark Parameters: For the generation of the *source dataset*, the user has to configure the benchmark generator appropriately. More specifically, the user must set values to the following parameters:

- Number of instances to retrieve from the input dataset
- Percentage of points to keep for each Trace (one can restrict the number of points to consider due to the possibly very large number of points per Trace)
- Percentage of the number of pairs of longitude/latitude to replace with address labels. To be able to apply value-based transformations on the Traces we need *literal type properties*. Thus, we replace the longitude, latitude values of the selected points with *place names*, using the Google Maps API [7], the Foursquare API [8] and the Nominatim API [9].
- Other parameters determine the allotment of each different types of transformation used for the generation of the *target dataset*.

Target Dataset Generation: In order to produce the *target dataset*, we retrieve all *source* dataset instances in a sequential manner, and apply the appropriate transformation(s) to obtain from source instance s_i, the target instance t_i. We repeat until the total number of retrieved instances reaches the size specified in the configuration file.

Table 1 presents the transformations implemented for the linking benchmark. In more detail:

- Transformations T1 - T4 have been proposed and implemented in SWING [8] and in OAEI [1] can be perceived as different cases of misspellings. Each transformation takes as input a *data*

[7] https://developers.google.com/maps/
[8] https://developer.foursquare.com/
[9] https://developer.mapquest.com/documentation/open/nominatim-search/

type property and a *difficulty level* that determines how difficult this modification is.

- T5 addresses the use of *different date formats*, while the given date is represented in yyyy-MM-dd'T'HH:mm:ss.SSSSSS format; We consider four types of date transformations, namely *short*, *medium*, *long* and *full*. For example, date "2017-03-20T10:19:00.000000", will be transformed to "20/03/17" for short, "Mar 20, 2017" for medium, "March 20, 2017" for long and "Monday, March 20, 2017" for a full transformation.
- T6 addresses the use of different *coordinate system formats*. The points are represented in pair (Longitude, Latitude) and we consider three type of transformations, namely: *Universal Transverse Mercator (UTM)*, *Degrees Minutes Seconds (DMS)* and *Military Grid Reference Universal Transverse Mercator (MGRUTM)*.
- T7 address the *change of longitude, latitude* to addresses using Google Maps, Foursquare and Nominatim APIs as referred above. For example, a point with longitude 14.184100 and latitude 40.899790 will be transformed to "Via Padreterno, 12, 80016 Marano di Napoli NA, Italy".
- T8 tackles the *addition* and *deletion* of *intermediate points* in a trace. Given two points, the computation of the new point to be added is done using the Haversine formula[10]. Additions and deletions are done on intermediate points in order to maintain departure and arrival points of a trace.

T1	Blank Character Addition/Deletion
T2	Random Character Addition/Deletion/Modification
T3	Token Addition/Deletion/Shuffle
T4	Abbreviation
T5	Change Date Format
T6	Change coordinate system format
T7	long/lat to address label
T8	Intermediate point Addition/Deletion

Table 1: Transformations

GOLD STANDARD: In order to test the performance of the systems, while applying the transformations to generate the target instance, we also generate a *gold standard*. Precisely, we generate a simple gold standard where we record the source and target instances that match ($s_i = t_i$) during the generation of the target instances. In addition, we propose a more detailed gold standard, which, apart from the source and target URIs that match, it also records the *exact transformations* applied on the source instance (including the property on which these are applied) in order to generate the target instance. Listing 3 shows a small example of source, target and gold standard files. More information about benchmark Key Performance Indicators (KPIs) can be found in Section 5.

[10] https://rosettacode.org/wiki/Haversine_formula

```
1
2 --------- SOURCE ---------
3
4 t1 a Trace .
5 t1 hasPoint p1 .
6 p1 hasTimestamp "2010-03-12T10:19:00.000000"^^<http://www.w3.org
    /2001/XMLSchema#dateTime> .
7 p1 lat "40.898560"^^<http://www.w3.org/2001/XMLSchema#decimal> .
8 p1 long "14.184440"^^<http://www.w3.org/2001/XMLSchema#decimal> .
9 p1 hasSpeed s1 .
10 s1 velocityValue "6.67"^^<http://www.w3.org/2001/XMLSchema#decimal>
    .
11 s1 velocityMetric <http://www.tomtom.com/ontologies/traces#
    kilometers_perHour> .
12
13 --------- TARGET ---------
14
15 t1' a Trace .
16 t1' hasPoint p1' .
17 p1' hasTimestamp "Mar 12, 2010" .
18 p1' label "Via Padreterno, 12, 80016 Marano di Napoli NA, Italy" .
19 p1' hasSpeed s1' .
20 s1' velocityValue "6.67"^^<http://www.w3.org/2001/XMLSchema#decimal>
    .
21 s1' velocityMetric <http://www.tomtom.com/ontologies/traces#
    kilometers_perHour>  .
22
23 --------- GOLD STANDARD ---------
24
25 t1 exactMatch t1' .
26
27 --------- DETAILED GOLD STANDARD ---------
28
29 t1 exactMatch t1' .
30 p1 exactMatch p1' .
31 p1 isTransformed transformationId .
32 transformationId type "CoordinatesToAddress" .
33 transformationId onProperty "lat/long" .
```

Listing 3: Linking Benchmark Example

4 Spatial Benchmark

4.1 Overview

For the Spatial Benchmark[11], we focus on transformations that follow the DE-9IM (Dimensionally Extended nine-Intersection Model) topological relations and determine whether the systems are able to identify such relations.

For the design of this benchmark, we focused on *(a)* on the correct implementation of all the topological relations of the DE-9IM topological model and *(b)* on producing large datasets large enough to stress the systems under test. To the best of our knowledge, there exist few systems that implement all the topological relations of DE-9IM, hence the benchmark already addresses the first choke point set. Moreover, we produced very large synthetic datasets using TomTom's original data, and hence we are able to challenge the systems regarding scalabity. Unfortunately, and as we discuss in Section 5.2, the restrictions imposed by HOBBIT Platform did not allow us to experiment with very large datasets.

Our benchmark gets as input a set of traces, which consists of various traces, each trace being a sequence of points. The points are expressed using *standard longitude/latitude coordinates*. In the Spatial Benchmark, we consider that the traces are represented Well-known text (WKT) format[12] [13]. Appropriate transformations are applied to the input set of traces in order to obtain the target dataset that can be used to test the ability of systems to identify DE-9IM topological relations. The gold standard is produced after the generation of the source and target datasets.

More specifically, as shown in Figure 3, the benchmark gets as input traces represented as *LineStrings*[14] and produces a *source* dataset and a *target* dataset that implements one of the DE-9IM relations according to the benchmark configuration parameters. Similar to the Linking benchmark, the Spatial Benchark uses the *Initialization*, *Resource Generator* and the *Resource Transformation modules*. The workflow between the different modules for the generation of the *source* and *target* datasets is the same as the one for the Linking Benchmark (see Section 3.1. The difference here is that the *Resource ransformation Module* uses the JTS Topology Suite[15] to return for each source instance s_i the transformed instance t_i that has a certain predecided relation (from the DE-9IM set) with s_i. In contrast with the Linking Benchmark, the Spatial Benchmark uses RADON [17] to produce the *gold standard*.

4.2 Dimensionally Extended nine-Intersection Model (DE-9IM)

The Dimensionally Extended nine-Intersection Model (DE-9IM) [18] is a topological model and a standard, used to describe the spatial relations of two geometries in two-dimensional space. The DE-9IM model is based on a 3×3 *Intersection Matrix* of the form:

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

[11] https://github.com/hobbit-project/SpatialBenchmark

[12] https://en.wikipedia.org/wiki/Well-known_text

[13] Nevertheless, if the input dataset comes in a different format, then we can easily, through a pre-processing phase, transform the data in the required format.

[14] A linestring is a one-dimensional object representing a sequence of points and the line segments connecting them.

[15] https://en.wikipedia.org/wiki/JTS_Topology_Suite

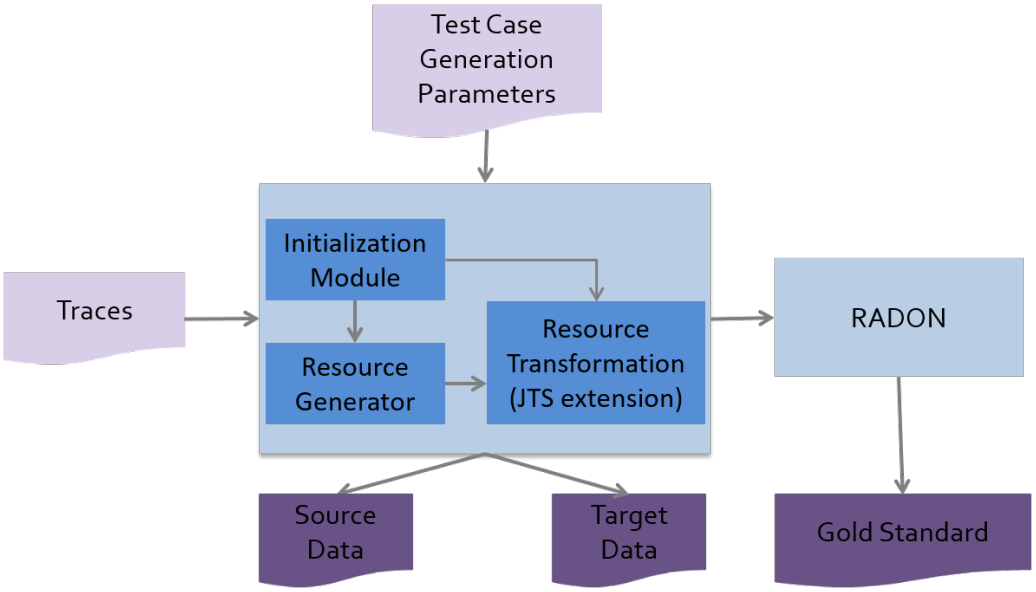


Figure 3: Spatial benchmark architecture

where dim is the maximum number of dimensions of the intersection ($\cap$) of the interior (I), boundary (B), and exterior (E) of geometries g_1 and g_2. Dimension ($\dim(x)$) values are obtained mapping the value 1 (for lines) to T (true), so using the boolean domain $\{T, F\}$. The supported spatial relations of DE-9IM are formally described below (see also Figure 4):

- **Equal**: Two geometries g_1 and g_2 are **topologically equal** if their interiors intersect and no part of the interior or boundary of one geometry intersects the exterior of the other. *Topologically equal* is equivalent to *Within* and *Contains* DE-9IM relations. Formally:

$$(I(g_1)I(g_2)) \wedge \neg(I(g_1)E(g_2)) \wedge \neg(B(g_1)E(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2))$$

- **Disjoint**: Two geometries g_1 and g_2 are **disjoint** if they have no point in common. They form a set of disconnected geometries. *Disjoint* is equivalent to *not Intersects*. Formally:

$$\neg(I(g_1)I(g_2)) \wedge \neg(I(g_1)B(g_2)) \wedge \neg(B(g_1)I(g_2)) \wedge \neg(B(g_1)B(g_2))$$

- **Touches**: A geometry g_1 **touches(meets)** a geometry g_2 if they have *at least one boundary point in common, but no interior points*. Formally:

$$(\neg(I(g_1)I(g_2)) \wedge I(g_1)B(g_2)) \vee (\neg(I(g_1)I(g_2)) \wedge B(g_1)I(g_2)) \vee (\neg(I(g_1)I(g_2)) \wedge B(g_1)B(g_2))$$

- **Contains:** A geometry g_1 **contains** a geometry g_2 if g_2 lies in g_1, and the interiors intersect. Another definition: "g_1 contains g_2 if no points of b lie in the exterior of g_1, and at least one point of the interior of g_2 lies in the interior of g_1". *Contains* is equivalent to *Within*(g_2, g_1). Formally:

$$(I(g_1)I(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2))$$

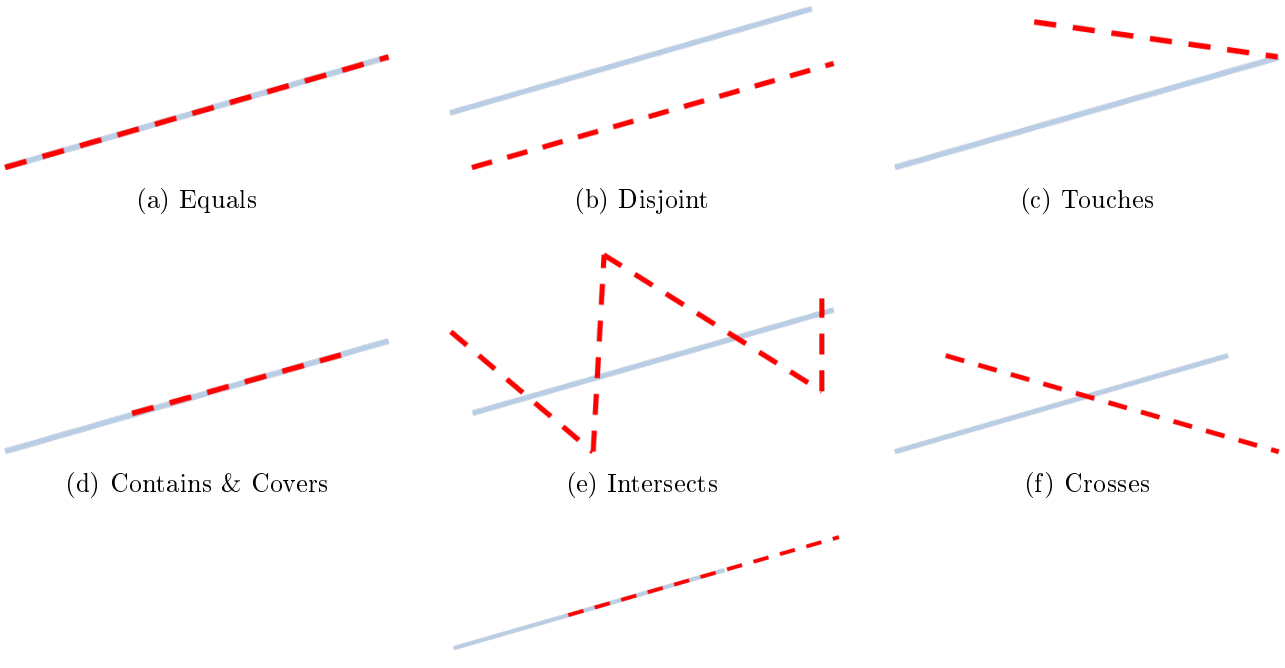
- **Covers:** A geometry g_1 **covers** a geometry g_2 if geometry g_2 lies in g_1. Other definitions: "no points of g_2 lie in the exterior of g_1", or "Every point of g_2 is a point of (the interior or boundary

of g_1". *Covers* is equivalent to *CoveredBy*(g_2, g_1). Formally:

$$\begin{aligned}((I(g_1)I(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2)))\vee \\ ((I(g_1)B(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2)))\vee \\ ((B(g_1)I(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2)))\vee \\ ((B(g_1)B(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2))) \quad (1)\end{aligned}$$

- **Intersects:** A geometry g_1 **intersects** g_2 if they have at least one point in common. *Intersects* is equivalent to *not Disjoint*.
- **Within:** A geometry g_1 is **within(inside)** g_2 if g_1 lies in the interior of g_2. *Within* is equivalent to *Contains*(g_2, g_1).
- **Covered by:** A geometry g_1 is **covered by** g_2 (extends *Within*) if every point of g_1 is a point of g_2, and the interiors of the two geometries have at least one point in common. *Covered by* is equivalent to *Covers*(g_2, g_1).
- **Crosses:** A geometry g_1 is **crosses** g_2 if they have some but not all interior points in common, and the dimension of the intersection is less than that of at least one of them. Mask selection rules are checked only when $\dim(g_1) \neq \dim(g_2)\dim(g_1) \neq \dim(g_2)$, except by line/line inputs, otherwise is false: $((I(g_1)I(g_2)) = 0)$for lines,$(I(g_1)I(g_2) \wedge I(g_1)E(g_2))$ when $\dim(g_1) < \dim(g_2)\dim(g_1) < \dim(g_2), (I(g_1)I(g_2) \wedge E(g_1)I(g_2))$ when $\dim(g_1) > \dim(g_2)\dim(g_1) > \dim(g_2)$
- **Overlaps:** A geometry g_1 is **overlaps** g_2 if they have some but not all points in common, they have the same dimension, and the intersection of the interiors of the two geometries has the same dimension as the geometries themselves. Mask selection rules are checked only when $\dim(g_1) = \dim(g_2)\dim(g_1) = \dim(g_2)$, otherwise is false: $((I(g_1)I(g_2)) \wedge (I(g_1)E(g_2)) \wedge (E(g_1)I(g_2)))$ for points or surfaces, $((I(g_1)I(g_2)) = 1 \wedge (I(g_1)E(g_2)) \wedge (E(g_1)I(g_2)))$ for lines.

Examples of the relations are shown in Figure 4.



(a) Equals (b) Disjoint (c) Touches

(d) Contains & Covers (e) Intersects (f) Crosses

(g) Overlaps

Figure 4: Examples of topological relations

Benchmark Parameters

For the generation of the source and target datasets, the user has to provide values for the following benchmark parameters:

- Number of instances to retrieve from the input dataset
- Percentage of points to keep for each Trace in the input dataset (this is due to the large number of points in each trace)
- DE-9IM topological relation that exists between the source and target trace. All the pairs of traces generated will follow the selected relation.

4.3 Source and Target Data Generation

In order to generate the *source* dataset we proceed as follows: from the input dataset, we only keep longitude and latitude information from each point. With those points, we create, for each trace, a LineString that is represented in **Well-known text (WKT)** representation format. WKT is a text markup language for representing vector geometry objects on a map, spatial reference systems of spatial objects and transformations between spatial reference systems. WKT offers a compact machine and human readable representation of geometric objects. A **LineString** is a WKT geometric object and consists of a sequence of two or more vertices, along with all points along the linearly interpolated curves (line segments) between each pair of consecutive vertices. The line segments in the line may intersect each other (in other words, the linestring may "curl back" in itself and self-intersect. A linestring must have either zero or two or more points.

Target Dataset Generation In order to produce the *target* dataset, we retrieve source instances in a sequential manner, and using the Extension of the JTS Topology Suite, we generate a target instance for the DE-9IM relation as specified in the benchmark configuration parameters. This process is continued until the requested number of instances is produced.

The **JTS Topology Suite**[16] is a Java API that implements a core set of spatial data operations using an explicit precision model and robust geometric algorithms. It provides a complete model for specifying 2-D linear geometry. Many common operations in computational geometry and spatial data processing are exposed in a clear, consistent and integrated API. JTS is intended to be used in the development of applications that support the validation, cleaning, integration and querying of spatial datasets. JTS is based on the notion of the *bounding box (bbox)*, which is an area defined by two longitudes (in the range $-180 \dots 180$) and two latitudes (in the range $-90 \dots 90$), such that the resulting box (included within these coordinates) is the minimum box that contains the geometry under study.

For generating the target data in the Spatial Benchmark, we decided to implement an extension[17] to the JTS Topology Suite that allows one to generate a target geometry, given a source geometry and the intended Intersection Matrix (i.e., DE-9IM relation) between the source and target geometries. Below, we describe the algorithm behind the implemented extension , for each target relation of DE-9IM. As we will see, all cases are essentially based on the algorithms that generate *disjoint LineStrings*, along with some random selection of points and/or series of points. In more details:

- **equal:** Given a source LineString, an equal LineString is the exact same.
- **disjoint:** Given a source LineString s we determine its bbox ($b(s)$), and randomly define coordinates for a bbox (say b') that does not intersect $b(s)$. This is done by just taking sufficiently

[16] https://svn.code.sf.net/p/jts-topo-suite/code/tags/Version_1.14
[17] https://github.com/jsaveta/jtsExtension

large (or sufficiently small) coordinates for the minimum (maximum) longitude or latitude coordinates. Finally, we generate a random LineString that entirely falls inside b', thereby guaranteeing disjointness.
In rare cases, it could happen that $b(s)$ covers the entire plane, so no b' can be defined. In these cases, we randomly break the original LineString into several smaller LineStrings (say $s_1, \ldots, s_k$, and compute their bboxes $(b(s_1), \ldots, b(s_k))$. Then, we use the above process to identify a bbox b' that does not intersect with any of them and create a random target LineString as above.
If, despite the partitioning of s, no appropriate b' can be found, we define an alternative, more fine-grained partition and repeat the process which ends when an appropriate b' is found, or when each pair of consecutive points of s is a partition; if even this fine-grained partition does not allow the definition of an appropriate bbox, then the original LineString covers the entire plane and no disjoint line can be created.

- **touches:** Given a source LineString s we first pick (randomly) a point (say p) that target LineString t will touch (this could be a boundary, or an intermediate point). Then, we create a LineString t' that is disjoint with s and create t by just adding p to t'. When adding p, we also make sure that it still follows the definition of "touches".
- **contains & covers:** Given a source LineString s, we randomly partition it into $s_1, \ldots, s_n$, and randomly pick one s_i to be the target LineString t.
- **intersects:** Given a source LineString s, we randomly pick n points $p_1, \ldots, p_n$ $(n > 1)$. Then, we create $n + 1$ disjoint LineStrings $t_1, \ldots, t_{n+1}$. Finally, we create the final LineString t by adjoining these LineStrings and points.
- **within & covered by:** Within and covered by are obtained from contains and covers respectively (they are their inverse relations).
- **crosses:** This is the same as intersects, except that $n = 1$, and that the chosen point must be an intermediate one.
- **overlaps:** Given a source LineString s, we pick an intermediate point, say p, essentially partitioning s into two segments, say s_1, s_2. Then, we create two LineStrings t_1, t_2. The LineString t_1 is randomly chosen to be equal to either s_1 or s_2. Without loss of generality, we assume that s_1 is chosen. The LineString t_2 is a LineString that is disjoint from s. The final target LineString t is defined by adjoining t_1 with t_2.

Gold Standard For the Spatial benchmark, contrary to the Linking Benchmark, the gold standard, could not be produced during the generation of the target dataset because it would be neither complete nor correct. Indeed, to determine this, the benchmark should check each generated target LineString t_i against all source LineStrings s_i for all possible relations that essentially amounts to implementing a system for the computation of the topological relations.

To avoid this problem, after all the source and target datasets are generated, we compute the gold standard using RADON [17]. RADON, is a novel approach for rapid discovery of topological relations between geo-spatial resources. RADON combines space tiling, minimum bounding box approximation and a sparse index to achieve a high scalability. RADON was evaluated with real datasets of various sizes and showed that in addition to being complete and correct, it also outperforms the state of the art by up to three orders of magnitude. Listing 4 shows a small example of source, target and gold standard. More information about benchmark Key Performance Indicators (KPIs) can be found in 5.

```
1
2 --------- SOURCE ---------
3
4 t1 a Trace .
5 t1 hasGeometry "LINESTRING (8.79232 48.92266, 8.79186 48.92246,
6 8.79135 48.92229, 8.79079 48.92219, 8.79012 48.92216, 8.78957
     48.92207,
7 8.78903 48.92185, 8.78848 48.92162, 8.78807 48.92133, 8.78782
     48.92097,
8 8.78771 48.92068, 8.78737 48.92031, 8.78695 48.92004)"
9 ^^<http://strdf.di.uoa.gr/ontology#WKT> .
10
11 --------- TARGET ---------
12
13 t1' a Trace .
14 t1' hasGeometry "LINESTRING (8.78903 48.92185, 8.78848 48.92162,
15 8.78807 48.92133, 8.78782 48.92097, 8.78771 48.92068, 8.78737
     48.92031)"
16 ^^<http://strdf.di.uoa.gr/ontology#WKT> .
17
18 --------- GOLD STANDARD (e.g. for COVERS) ---------
19
20 t1  t1'.
```

Listing 4: Spatial Benchmark Example

5 Benchmark in HOBBIT platform

Both the Linking[18] and Spatial Benchmarks[19] are integrated in the HOBBIT platform[20]. We describe the components of the platform below that are common to both benchmarks and we demonstrate an experimental set up only for the Spatial Benchmark only since it is more complex than the Linking Benchmark.

5.1 Benchmark Modules

The platform consists of 5 modules, the *Benchmark Controller*, the *Data Generator*, the *Task Generator*, the *System Adapter* and the *Evaluation Module*.

- Benchmark Controller is used to create and orchestrate all other components needed to execute the benchmark. It receives from the platform the *benchmark parameters* and creates the Data Generator, the Task Generator as well as the Evaluation Module. Those subcomponents initialize themselves and the Benchmark Controller sends a message on the platform when it is ready. When the platform receives the message, the Benchmark Controller starts the benchmarking. The *Data Generator* starts its tasks while the *Task Generator* waits for the termination of the former and starts after it. All generated data from Data and Task Generators are sent to the *System Adapter* and the *Evaluation Module*. The Evaluation Module waits until the System Adapter generates the response of the task and then computes the performance results. Those results are sent from the Benchmark Controller to the platform.
- Data Generator creates the data that is needed for the benchmark and sends it to the *Task Generator* and the *System Adapter*. In our benchmark, the Data Generator reads the benchmark parameters (number of generators, number on instances, serialization format, spatial relation and percentage of points to keep for each trace) and based on those parameters, generates the source data sent to the System Adapter, the target data, the topological relation and the gold standard that are send as a Task to the *Task Generator*.
- Task Generator reads the Task and sends it to the *System Adapter*. It also sends the expected answers for the Task along with the current timestamp to the *Evaluation Module*.
- System Adapter receives the data from the Data Generator and the Task from the Task generator. A Task has a unique ID and the system should generate a single response for every Task. This response is sent to the *Evaluation Module* together with the task ID it belongs to. In our case, we implemented a system adapter using LIMES [13] as a baseline system. For each topological relation we have created a configuration file[21]. We use the proper configuration file regarding the relation and send the results to the Evaluation Module when the system terminates.
- Evaluation Module is responsible for the evaluation of the Tasks. After the Data and Task Generators in addition to the System Adapter terminate, the Benchmark Controller starts the evaluation. The Evaluation Module evaluates the system responses by comparing them with the expected results. This module also computes the performance time of the system. The results are enhanced with additional information by the platform controller before they are stored in the platform storage.

Key Performance Indicators (KPIs): The performance metric(s) in a benchmark determine the

[18] https://github.com/hobbit-project/LinkingBenchmark
[19] https://github.com/hobbit-project/SpatialBenchmark/blob/master/src/main/java/org/hobbit/spatialbenchmark/platformConnection/
[20] http://master.project-hobbit.eu
[21] https://github.com/hobbit-project/SpatialBenchmark/tree/master/configs/topologicalConfigs

effectiveness and efficiency of the systems and tools. In this benchmark, we focus on the quality of the output in terms of standard metrics such as precision, recall and f-measure [9]. We also aim to quantify the performance (in ms) of the systems measuring the time needed to return all the results. We will report the average answer delay between the timestamp that the Task was executed and the timestamp that the results are sent to the Evaluation Module. The first time stamp is generated by the Task generator and the second time stamp is generated by the Evaluation Module.

5.2 Experimental Results

Figures 5, 6, 7 and 8 are screenshots of HOBBIT platform running the Spatial Benchmark with the baseline system. The screenshots are shown as sample, while more experiments will be presented below.

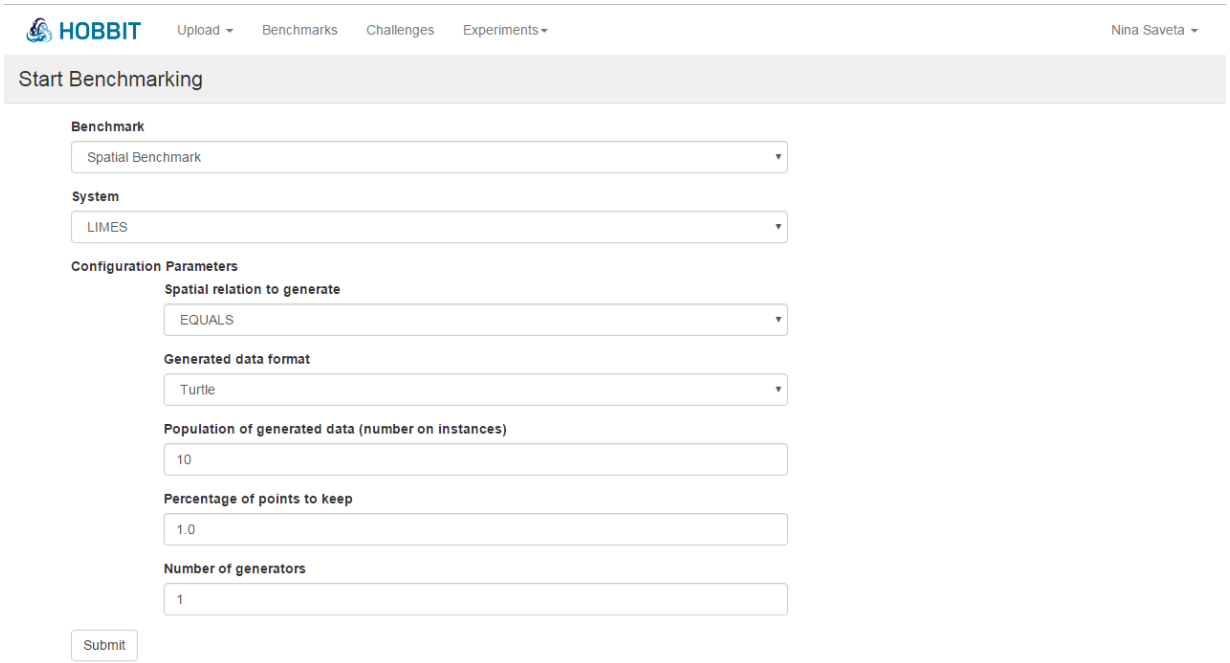


Figure 5: Benchmark in HOBBIT platform

Specifically, Figure 5 presents the page where someone can start a benchmark. The user selects the benchmark and the system to evaluate and fills the configuration parameters. The relation parameter and the data format can be selected from a drop-down list. As soon as the experiment is submitted, details are shown as in Figure 6. The details consist of the benchmark and the system URIs, the configuration parameters, the experiment ID and the time that the experiment started.

While the experiment is running, the user has the ability to view the status of the experiment along with other information, (as shown in Figure 7) just by clicking the experiment ID.

Finally, the results of the experiment are shown in a table as in Figure 8. The results consist of the benchmark, the system, the challenge task if the experiment was part of a challenge, the errors, if any, and the values for the KPIs that were defined in the benchmark.

To show the rate between the time performance and the population (10, 100 and 1000) of the traces but also the time difference between the topological relations we executed a set of experiments for all the DE-9IM relations using the HOBBIT Platform. We should mention here that we are only

Figure 6: Experiment submission example

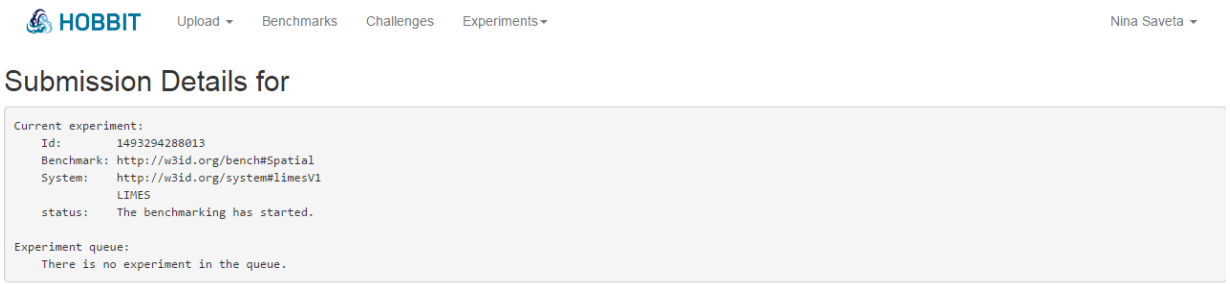


Figure 7: Experiment messages while running example

presenting the time performance and not precision, recall and f-measure as all were equal to 1.0.

Table 2 shows the results of the executed experiments. Relation *disjoint*, needs the less time for the system to return the results. *Overlaps*, seems to be one of the "difficult" relations as it needs more time than *Equals*, *Crosses*, *Covers/Covered By* and *Contains/Within* that need approximately the same time. For two of the topological relations, *Touches* and *Intersects* we were only able to run experiments for source data consisting of 10 traces (and 10 traces in the target data) as in the other cases the experimenys stopped from the platform due to the time limit set by the platform. The reason that those relations need much more time is that our data contain LineStrings that have a large number of points in a rather small area. LIMES first computes minimal bounding boxes (MBBs) for all geometries, which is very fast. Then, LIMES employs space tiling to index those MBBs, which is also quite fast, and finally, for all MBBs that share at least one tile, carries out the actual "intersects" computation using com.vividsolutions.jts.geom. In our use case, almost all geometries MBBs do intersect and therefore a lot of computation tasks are issued to com.vividsolutions.jts.geom package, which is slow and also memory intensive.

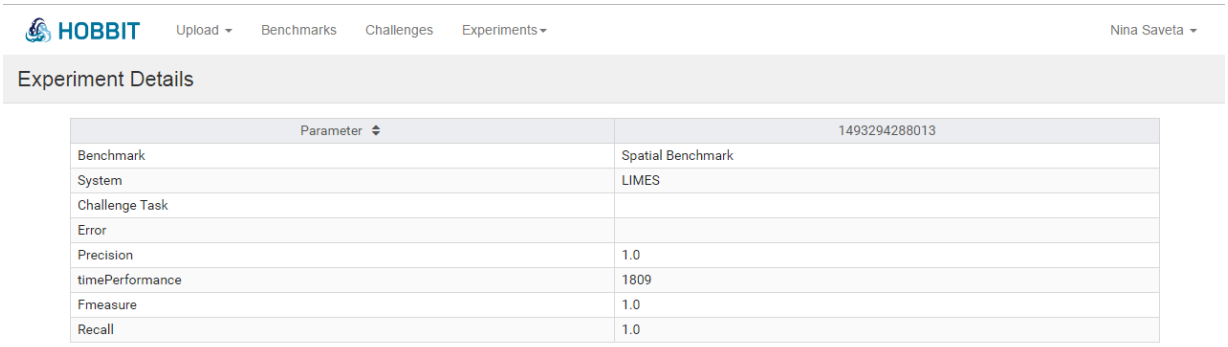


Figure 8: Experiment results example

RELATION	# 10	# 100	# 1000
EQUALS	1604	3711	13253
CROSSES	1400	3605	17424
COVERED BY	1253	3203	11375
COVERS	1272	2922	10177
DISJOINT	800	2865	8057
OVERLAPS	2072	9208	247466
WITHIN	1450	3141	11024
CONTAINS	1268	2895	9748
INTERSECTS	141526	Platform Time Limit	Platform Time Limit
TOUCHES	231046	Platform Time Limit	Platform Time Limit

Table 2: Time rate while increasing population for each topological relation.

6 Conclusions

We presented two benchmarks defined in the context of T4.1 of the project during the first phase of WP4. The first is the Linking benchmark, which is based on SPIMBENCH, to be used by Instance Matching tools that implement string-based approaches for identifying matching entities. The second is the Spatial benchmark, which checks whether the systems can identify DE-9IM (Dimensionally Extended nine-Intersection Model) topological relations.

For our experiments with the Spatial Benchmark we saw that there do not exist link discovery systems (except RADON and LIMES) that identify links for spatial datasets. Moreover, we witnessed that LIMES, that is one of the most prominent systems for link discovery does not perform well when the input datasets are very large. In the experiments that the developers of LIMES conducted, they used "large" traces that had few points. In our case, our traces contain millions of points, a fact that makes our datasets exteremely large for the systems under test. As a consequence, performing link discovery for several of the DE-9IM relations takes prohibitive time. In order to deal with this, and from our side, we are going improve the implementation of the JTS extension . We are also going to work with different classes of object geometries in addition to LineString, and last, explore the possibility to include new evaluation metrics.

Both benchmarks will be used for the Upcoming challenge at the Ontology Matching Workshop (OM) at ISWC 2017[22]. The aim of the instance matching benchmark for spatial data challenge is to test the performance of link discovery tools that implement string-based as well as topological approaches for identifying matching spatial entities. The Challenge will include 2 Tasks: *Task 1 (Linking)* will measure how well the systems can match traces that have been altered using string-based approaches along with addition and deletion of intermediate points. For this task, Linking Benchmark 3 will be used. *Task 2 (Spatial)* will measure how well the systems can identify DE-9IM (Dimensionally Extended nine-Intersection Model) topological relations. The supported spatial relations are the following: Disjoint, Touches, Contains/Within, Covers/CoveredBy, Intersects, Crosses, Overlaps and the traces are represented in Well-known text (WKT) format. For this task, Spatial Benchmark 4 will be used.

References

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

[2] Y. Velegrakis A. Morris and P. Bouquet. Identification on the Semantic Web. In *SWAP*, 2008.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[4] P. Christen. *Data Matching - Concepts and Techniques for Record Linkage,Entity Resolution, and Duplicate Detection, Data- Centric Systems and Applications*. Springer, 2012.

[5] V. Christophides, V. Efthymiou, and K. Stefanidis. *Synthesis Lectures on the Semantic Web: Theory and Technology*, chapter Entity Resolution in the Web of Data. Morgan and Claypool Publishers, 2015.

[6] E. Daskalaki, I. Fundulaki G. Flouris, and T. Saveta. Instance Matching in the era of Linked Data. *Journal of Web Semantics*, 39, 2016.

[22] https://project-hobbit.eu/challenges/om2017/

[7] Z. Dragisic, K. Eckert, J. Euzenat, A. Ferrara D. Faria, R. Granada, V. Ivanova, E. Jimenez-Ruiz, A. Oskar Kempf, S. Montanelli P. Lambrix, H. Paulheim, D. Ritze, P. Shvaiko, A. Solimando, C. Trojahn, O. Zamaza, and B. Cuenca Grau. Results of the Ontology Alignment Evaluation Initiative. In *OM*, 2014.

[8] A. Ferrara, S. Montanelli, J. Noessner, and H. Stuckenschmidt. Benchmarking Matching Applications on the Semantic Web. In *ESWC*, 2011.

[9] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[10] M. HernÃąndez and S. Stolfo. Real-world data is dirty: data cleansing and the merge/purge problem. *Data Mining Knowledge Discovery*, 2:9–37, 1998.

[11] Manolis Koubarakis and Kostis Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: the Model stRDF and the Query Language stSPARQL. In *ESWC*, 2010.

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

[13] Axel-Cyrille Ngonga Ngomo. On link discovery using a hybrid approach. *Journal on Data Semantics*, 1(4):203–217, 2012.

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

[15] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *ACM SIGKDD*, pages 269–278, 2002.

[16] T. Saveta, E. Daskalaki, G. Flouris, I Fundulaki, M. Herschel, and A.-C. Ngonga Ngomo. Pushing the limits of instance matching systems: A semantics-aware benchmark for linked data. In *WWW*, pages 105–106. ACM, 2015. Poster.

[17] Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. RADON - Rapid Discovery of Topological Relations. In *Proceedings of The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017.

[18] Christian Strobl. *Encyclopedia of GIS*, chapter Dimensionally Extended Nine-Intersection Model (DE-9IM), pages 240–245. Springer, 2008.

[19] S. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *ACM SIGMOD*, pages 219–232, 2009.