SPgen: A Benchmark Generator for Spatial Link Discovery Tools

T. Saveta[1], I. Fundulaki[1], G. Flouris[1], and A.-C. Ngonga-Ngomo[2]

[1] Institute of Computer Science - FORTH, Greece
[2] University of Paderborn, Germany

Abstract. A number of real and synthetic benchmarks have been proposed for evaluating the performance of link discovery 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 of the Linked Open Data Web, such as LinkedGeoData contain vast amounts of spatial information. Several systems that manage spatial data and consider the topology of the spatial resources and the topological relations between them have been developed. In order to assess the ability of these systems to handle the vast amount of spatial data and perform the much needed data integration in the Linked Geo Data Cloud, it is imperative to develop benchmarks for geo-spatial link discovery. In this paper we propose the *Spatial Benchmark Generator SPgen* that can be used to test the performance of link discovery systems which deal with topological relations as proposed in the state of the art DE-9IM (Dimensionally Extended nine-Intersection Model). *SPgen* implements all topological relations of DE-9IM between *LineStrings* and *Polygons* in the two-dimensional space. A comparative analysis with benchmarks produced using *SPgen* to assess and identify the capabilities of AML, OntoIdea, RADON and Silk spatial link discovery systems is provided.

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 publication of information by different publishers, and the interlinking of Web resources across knowledge bases. In most cases, the cross-dataset links are not integral to newly created datasets and must be determined automatically, using *link discovery* tools amongst others [1]. The large variety of techniques demands the availability of comparative evaluations to determine which one is best suited for a given use case. Performing such an assessment 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 have been proposed for evaluating the performance of such systems [2].

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

in the Linked Open Data Web are geo-spatial knowledge bases (e.g., LinkedGeoData,[3] with more than 30 billion triples). 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 various domains, it is critical that benchmarks for geo-spatial link discovery are developed.

In this paper we discuss the *Spatial Benchmark Generator SPgen* that can be used to test the performance of systems that deal with topological relations proposed by the state of the art DE-9IM (Dimensionally Extended nine-Intersection Model) [3]. *SPgen* is developed in the context of the H2020 European project HOBBIT.[4] This benchmark generator implements all topological relations of DE-9IM between *LineStrings* and *Polygons* in the two-dimensional space. *SPgen* follows the choke point-based approach [4] for benchmark design, i.e., it focuses on the technical difficulties of existing systems and implements tests that address those difficulties and "push" systems to resolve them. More specifically we focus on the following choke-points in *SPgen*:

- **Scalability:** produce datasets large enough to stress the systems under test
- **Output quality:** compute *precision*, *recall* and *f-measure*
- **Time performance:** measure the time the systems need to return the results

To the best of our knowledge such a *generic benchmark generator*, that checks the performance of linking systems for spatial data, does not exist. We also provide a comparative analysis with benchmarks produced using *SPgen* to assess and identify the capabilities of AML [5,6], OntoIdea [7], RADON [8] and Silk [9] spatial link discovery systems.

The outline of the paper is as follows: Section 2 discusses related work. We present the Dimensionally Extended nine-Intersection Model and the datasets employed in *SPgen* in Sections 3 and 4 respectively. *SPgen* is described in detail in Section 5 and the experiments we conducted in Section 6. We conclude and present future work in Section 7.

2 Related Work

SPgen is a generic, schema agnostic and choke-point based [4] benchmark generator that takes as input *trajectories* and checks the performance of linking systems for spatial data. To the best of our knowledge this is the first link discovery benchmark for spatial data. In this section we will discuss the most relevant benchmarks to *SPgen* and more specifically *benchmarks for spatial RDF stores* and *benchmarks for spatial relational databases*.

Benchmarks for spatial RDF stores The most relevant benchmark to *SPgen* is Geographica [10] that evaluates RDF stores and consists of micro and macro

[3] `http://linkedgeodata.org/About`
[4] `http://www.project-hobbit.eu`

benchmarks following the approach of Jackpine [11]. Geographica's micro benchmark tests the spatial components of RDF stores using queries that consist of spatial selections, joins and aggregations but it does not address topological relations. Geographica's macro benchmark tests the performance of the RDF stores using reverse geocoding, map search and browsing and a real-world use case from the Earth Observation domain. Kolas [12] proposed a benchmark that extends the LUBM benchmark for RDF stores [13] in order to include spatial entities. In this case, LUBM queries were extended to cover basic types of spatial queries namely location, range, join and nearest neighbor.

Benchmarks for spatial relational databases The most recent benchmark for spatial databases and the most relevant benchmark to *SPgen* is Jackpine [11] and consists of a micro and a macro benchmark. Jackpine's micro benchmark includes queries based on DE-9IM with queries that focus on spatial analysis. Jackpine's macro benchmark includes queries based on spatial data applications (search and browsing, geocoding, flood risk analysis, etc.). VESPA[14] is a vector-based spatial benchmark that tests the functionality and performance of spatial database systems and includes a set of query and update tasks over synthetic datasets composed of points, lines and polygons. The Á la Carte [15] benchmark produces a synthetic dataset composed only of rectangles and is used to test the performance of different spatial join techniques. Last but not least, one of the first benchmarks that uses real datasets and real queries representative of Earth Science tasks is SEQUOIA [16] and its extension [17]. The queries are related to data loading, raster data management, filtering, spatial joins and path computation over graphs.

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

The Dimensionally Extended nine-Intersection Model (DE-9IM) [3] or Clementini-Matrix is used for computing the spatial relationships between geometries. It is a topological model, based on the Nine-Intersection Model (9IM), used to describe the spatial relations of geometries in two-dimensional space. The model considers the objects' *interiors*, *boundaries* and *exteriors* and analyzes the intersections of these nine objects parts to determine their relationship.

Spatial relations are *boolean* functions that are used to test the relationships between two geometry objects. The spatial relationships described by DE-9IM are *equals*, *disjoint*, *touches*, *contains*, *within*, *intersects*, *covers*, *covered by*, *crosses* and *overlaps* including relations among *LineStrings* and *Polygons*. A *LineString* is a one-dimensional 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. A *Polygon* is a two-dimensional surface stored as a sequence of points where the first point is connected to the last point defining its exterior bounding ring and zero or more interior rings. In order to better

understand the topological relations of DE-9IM it is necessary to define the boundary, interior and exterior of the geometric types. For instance, in the case of *LineString*, the *boundary* (B) are the two end points, the *interior* (I) consists of *points* that are left when the boundary points are removed and the *exterior* (E) are the points not in the interior or boundary. In the case of *Polygon*, the *interior* are the points within the rings, the *boundary* is a set of rings and finally the *exterior* are points not in the interior or boundary.

Given that each geometry is represented by the aforementioned 3 dimensions, all possible relationships between two geometries are represented by a 3×3 matrix of the form:

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

where dim is the maximum number of dimensions of the intersection ($\cap$) of the interior (I), boundary (B), and exterior (E) of geometries a and b. The dimension of empty sets is equal to -1 or F (false). The dimension of non-empty sets is equal to the maximum number of dimensions of the intersection, specifically, 0 for points, 1 for lines, 2 for areas. Thus, the domain of the model is $\{0, 1, 2, F\}$. A simplified version of $\dim(x)$ values is obtained by mapping the values $\{0, 1, 2\}$ to T (true), so using the boolean domain $\{T, F\}$. The supported spatial relations of DE-9IM are formally described below:

Equals: Two geometries g_1 and g_2 are *equal* if the two geometries are *topologically equal*, that is if their interiors intersect and no part of the interior or boundary of one geometry intersects the exterior of the other. 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. 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 is the following: g_1 contains g_2 if no points of g_2 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. It is the inverse of *Within*. Formally:

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

Within: A geometry g_1 is within (inside) geometry g_2 if g_1 lies in the interior of g_2. *Within* is the inverse of *Contains*.

Intersects: A geometry g_1 intersects geometry g_2 if they have at least one point in common.

Covers: A geometry g_1 **covers** 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". It is the inverse of *CoveredBy*. Formally:

$$\begin{array}{c}((I(g_1)I(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2)))\vee \\ ((I(g_1)B(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2)))\vee \\ ((B(g_1)I(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2)))\vee \\ ((B(g_1)B(g_2)) \wedge \neg(E(g_1)I(g_2)) \wedge \neg(E(g_1)B(g_2)))\end{array}$$

Covered By: A geometry g_1 is **covered by** geometry 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 the inverse of *Covers*.

Crosses: A geometry g_1 *crosses* geometry g_2 if they share some but not all interior points, and the dimension of the intersection of the two geometries is less than that of at least one of the geometries.

Overlaps: A geometry g_1 *overlaps* geometry g_2 if the geometries share some, but not all points in common, and the intersection has the same dimension as the geometries themselves.

Examples of the DE-9IM relations for *LineStrings* and *Polygons* geometries are shown in Figures 1 and 2. Figure 1 presents the DE-9IM relations between LineStrings and Figure 2 demonstrates the DE-9IM relations between LineStrings and Polygons.

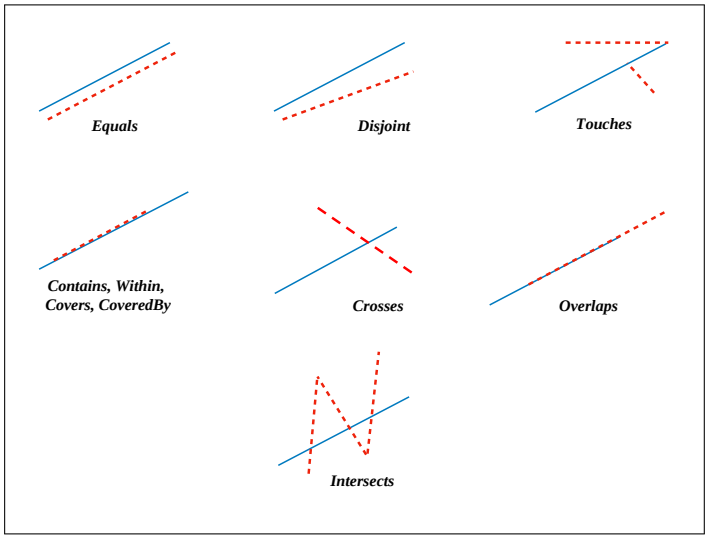

Fig. 1: Examples of DE-9IM topological relations for *LineStrings*

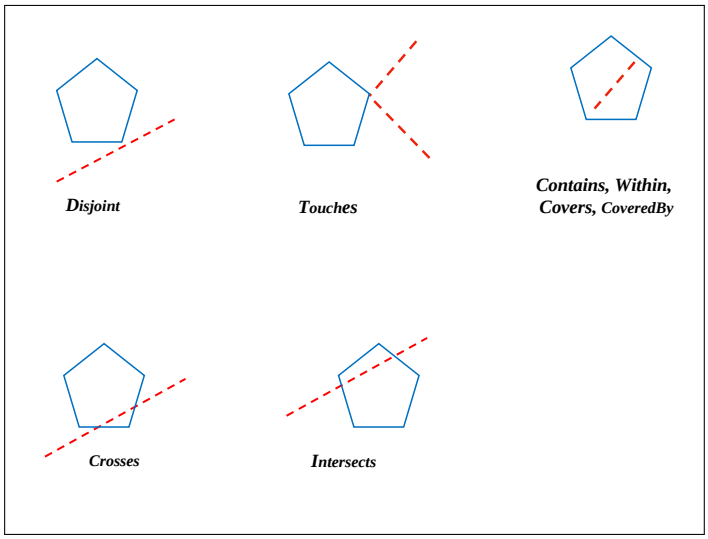

Fig. 2: Examples of DE-9IM topological relations for *LineStrings* and *Polygons*

4 Datasets

In this Section, we present the datasets we experimented with *SPgen*. Recall that the generator is *schema agnostic* and can work, in general, with trajectories, i.e., sequences of *longitude, latitude pairs*. We used two datasets generated from TomTom[5] and Spaten [18].

TomTom Data Generator*:* TomTom provides a Synthetic Trace Generator[6] developed in the context of the H2020 HOBBIT Project, which facilitates the creation of an arbitrary volume of data from statistical descriptions of vehicle traffic. More specifically, it generates *traces*, with a trace being a list of (longitude, latitude) pairs recorded by one device (phone, car, etc.) throughout one day. The generator uses probability distributions for variables like start and end locations of trips, their starting time or what is the device's update frequency. Using parameters sampled from such distributions, a map is then used to find an appropriate route for the trip and successive points are generated at a regular time interval with typical speeds for each road. TomTom's ontology is shown in Figure 3. The main class is class `Trace` that contains one or more points (class `Point`) which represents in its turn latitude, longitude pairs. Each point is associated with a *velocity* (class `Velocity`), instances of which have properties `velocityMetric` and `velocityValue`. A point also has attributes `hasTimeStamp` that takes its values in class `xsd:TimeStamp` which designates the time an object

[5] `https://www.tomtom.com/en_gr/`

[6] `https://git.project-hobbit.eu/filipe.teixeira/synthetic-trace-generator/container_registry`

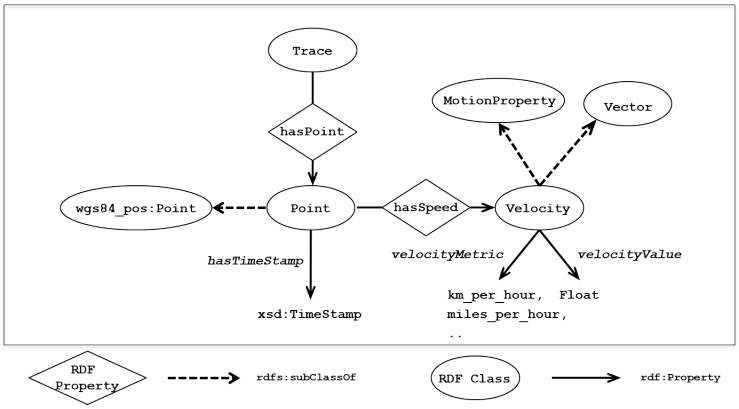

Fig. 3: TomTom Schema

was at this specific point. For our benchmark we are only interested in the points of a trace.

Spaten: Spatio-temporal and Textual Big Data Generator*:* Spaten [18] is an open-source configurable spatio-temporal and textual dataset generator, that can produce large volumes of data based on realistic user behavior. Spaten extracts GPS traces from realistic routes utilizing Google Maps API, and combines them with real POIs and relevant user comments crawled from TripAdvisor. The injection of social properties extracted by existing Twitter graphs to the generated data, along with further parameterization, leads to realistic Geo-Social Network (GeoSN) datasets. Spaten publicly offered GB-size datasets with millions of check-ins and GPS traces.[7] We used the provided trajectories of each user as our second dataset as it takes extremely long time and very powerful computing infrastructure to generate such data - Spaten developers produced those datasets in the period of 2 months. These trajectories consist of timestamps and longitude, latitude pairs (i.e., points) represented in CSV format (Listing 1.1 shows an example). We transformed the given dataset into Turtle format using the TomTom ontology before using it as input dataset in *SPgen*.

```
1 id_1,   timestamp_1,   Point_1
2 id_1,   timestamp_2,   Point_2
3       . . .
4 id_1,   timestamp_n,   Point_n
```

Listing 1.1: Spaten Example Data

[7] https://github.com/Thaleia-DimitraDoudali/Spaten

5 *SPgen*: A Link Discovery Benchmark Generator for Spatial Data

5.1 Overview

In *SPgen*,[8] we focus on relations that follow the DE-9IM (Dimensionally Extended nine-Intersection Model) and determine whether the systems are able to identify those relations between different instances. Each instance is either a *LineString* or a *Polygon*. *SPgen* gets as input traces represented as *LineStrings* and produces a *source* and a *target* dataset. The source dataset is identical to the input traces but is expressed in the Well Known Text format (WKT),[9] whereas the target dataset consists of *LineStrings* or *Polygons* that are generated from the source dataset in such a way that traces in the target dataset have a specific topological DE-9IM relation with the traces of the source dataset.

In *SPgen* we propose a set of test cases whose objective is to test whether link discovery systems for spatial data can identify whether a DE-9IM relation holds between different geometries. *SPgen* implements all topological relations of DE-9IM between LineStrings and Polygons in the two-dimensional space. The *gold standard* is produced after the generation of the source and target using RADON [8]. We discuss in Subsection 5.4 why we opted for this solution. In the next subsections we will describe *SPgen* in more detail.

5.2 *SPgen* Architecture

The architecture of *SPgen* is shown in Figure 4. *SPgen* takes a sequence of *traces* as input and a set of *user-defined parameters* such as the (a) number of instances to retrieve from the input dataset, (b) percentage of points to keep for each input trace,[10] (c) geometry of the *target* dataset (note that the target dataset can be either a *LineString* or a *Polygon*) and (d) the DE-9IM topological relation of interest.

The input dataset is processed by the Initialization Module that reads the user-defined parameters and retrieves the input traces by means of SPARQL queries. The retrieved traces are passed to the Resource Generation Module to generate the *source dataset* that transforms each retrieved trace to a *LineString* represented in WKT format. This module interacts with the Resource Transformation Module that generates the target instances represented again in WKT; the module implements the *DE-9IM topological relations* discussed in Section 5.3. The relations are implemented as an *extension*[11] of the JTS Topology Suite,[12]

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

[9] 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.

[10] A trace with a possibly huge number of points cannot be processed by systems, hence we would like to give the ability to developers to restrict the trace size.

[11] `https://github.com/jsaveta/jtsExtension`

[12] `http://svn.code.sf.net/p/jts-topo-suite/code/tags/Version_1.14/`

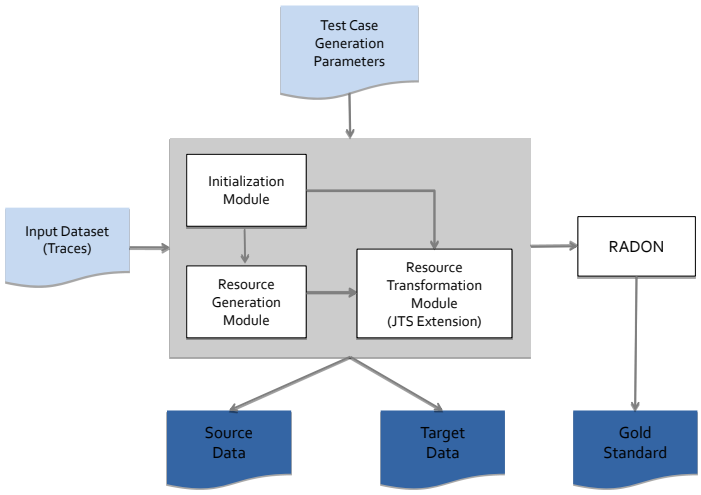

Fig. 4: *SPgen* Architecture

a JAVA API that provides a core set of spatial data operations using an explicit precision model and robust geometric algorithms.

The target dataset obtained from the Resource Transformation Module along with the source dataset, is passed as input to RADON.

5.3 Test Cases

SPgen implements all topological relations of DE-9IM between LineStrings and Polygons in the two-dimensional space. Due to space limitations, we only discuss the DE-9IM relation *Disjoint* for LineStrings, and the relation *Within* for LineStrings and Polygons. Other relations are handled in a similar fashion. Our algorithms are based on the idea of the *minimum bounding box* (*bbox*) which is an area defined by two longitudes (in the range -180 ... 180) and two latitudes (in the range -90 ... 90), such that the resulting bounding box (included within these coordinates) contains the geometry under study.

Disjoint (LineString/LineString): Given a LineString s, and the DE-9IM *Disjoint* relation r, we produce a LineString t disjoint with s, as follows: first, we compute the bounding box $b(s)$ of s, and randomly define longitude, latitude coordinates for a bounding box $b(t)$ that does not intersect with $b(s)$. In order to find $b(t)$, we find sufficiently large (or sufficiently small) coordinates for the minimum (maximum) longitude or latitude coordinates. Second, we generate a random LineString t with the same number of points as s that entirely falls inside $b(t)$, thereby guaranteeing disjointness between s and t.

In the case in which $b(s)$ covers the entire plane (i.e., its longitude, latitude coordinates have the maximum/minimum values), no $b(t)$ can be defined. In

these cases, we break s into several smaller LineStrings, say s_1; ... ; s_k, and compute their corresponding bounding boxes $b(s_1)$; ... ; $b(s_k)$. Then, we use the above process to identify a bounding box $b(t)$ that does not intersect with any of them and create a random target LineString as discussed earlier.

If, despite the partitioning of the bounding box $b(s)$ of LineString s, no appropriate t can be found, then we define a more fine-grained partition and repeat the process which ends when an appropriate disjoint LineString t can be found, or when each pair of consecutive points of s is a partition; if even this fine-

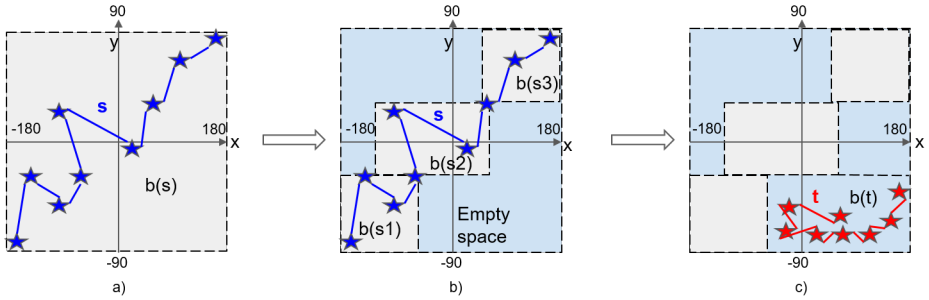

Fig. 5: Example for Disjoint (LineString/LineString)

grained partition does not allow the definition of an appropriate bounding box, then the original LineString covers the entire plane and no disjoint LineString can be created.

Figure 5 provides an example of the aforementioned process. In subfigure (a) we can see the source LineString s and its bbox $b(s)$. We are in the case where $b(s)$ covers the entire plane, thus we break s into smaller LineStrings and compute their corresponding bounding boxes $b(s_1)$; $b(s_2)$; $b(s_3)$ (subfigure (b)). We do not need to break s more as there is already an empty space where we can generate a bbox $b(t)$ and generate a disjoint to s, target LineString t (subfigure (c)).

Within (LineString/Polygon): Given a LineString s, and DE-9IM *Within* relation r, we produce a Polygon t in which s is *within*, as follows: First, using the JTS API we find the minimum-area convex polygon that contains LineString s. Then, we slightly expand the returned Polygon in order not to cross LineString s and thus we create target Polygon t. In the rare case in which s has one or more points whose longitudes are equal to -180 or 180 or one or more points whose longitudes are equal to -90 or 90, no Polygon that contains s exists. Figure 6 provides an example of the aforementioned process. In subfigure (a) we can see the source LineString s and its bbox $b(s)$ that does not cover the entire plane. Thus, we are able to define a Polygon that contains s (subfigure (b)) and then slightly expand it in order to create a Polygon t that in combination with s follows the definition of DE-9IM *Within* relation (subfigure (c)).

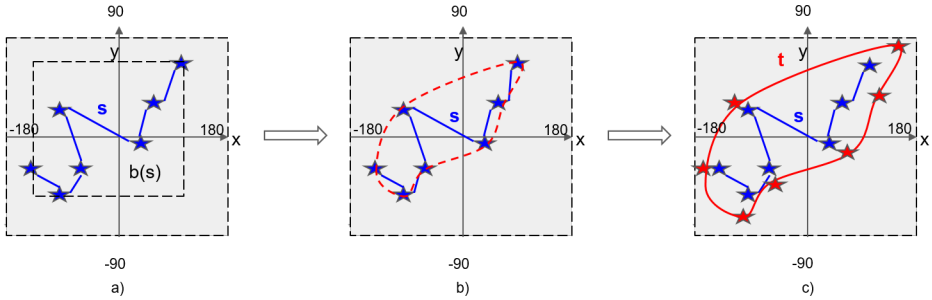

Fig. 6: Example for Within (LineString/Polygon)

5.4 Gold Standard

The *gold standard* produced by *SPgen* is not created during the generation of the target dataset, since it would not be complete: in order to compute the gold standard, we would have to check each generated target LineString or Polygon against all source LineStrings, a process that essentially amounts to implementing a system for the computation of the topological relations. Thus, to compute the gold standard, we resorted to an appropriate implemented system, namely RADON [8].

RADON was selected because it is a novel approach for rapid discovery of topological relations among geo-spatial resources. It combines space tiling, minimum bounding box approximation and a sparse index to handle very large datasets. 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 spatial link discovery systems by up to three orders of magnitude. Thus, it is appropriate for our purposes.

5.5 Key Performance Indicators

The key performance indicators of a benchmark determine the effectiveness and efficiency of the systems and tools. In *SPgen* we focus on the *output quality* in terms of standard metrics such as *precision*, *recall* and *f-measure* [19]. We also aim to quantify the *time performance* of the systems measuring the *time* needed by the link discovery system to return results.

6 Experimental Results

In this section we describe the experiments we conducted in order to show how well the various spatial linking systems performed regarding output quality and time performance for datasets of various sizes and for the different DE-9IM topological relations.

Datasets & Tasks: We ran experiments for all the DE-9IM relations and for LineString/LineString and LineString/Polygon cases for both TomTom and

Spaten datasets ranging from 200 to 2K instances, not exceeding 64 KB per instance due to a limitation of SILK.[13] This is important in order to get a fair comparison for the systems under test. We report here the results for quality output and time performance for all systems.

Experimental Setup: All the experiments were executed using the HOBBIT Platform[14] where *SPgen* is integrated and the platform time limit was set to 75 minutes. Thus, we provide a comparative analysis with benchmarks produced using *SPgen* and were able to assess and identify the capabilities of four systems, namely AgreementMakerLight (AML), OntoIdea, Rapid Discovery of Topological Relations (RADON) and Silk.

Tasks: We divided the experiments into four tasks. In the first two tasks (SLL and LLL), the systems were asked to match LineStrings to LineStrings considering a given relation for 200 and 2K instances for the TomTom and Spaten datasets. In the last two second tasks (SLP, LLP), the systems were asked to match LineStrings to Polygons (or Polygons to LineStrings depending on the relation) again for both datasets. We are only presenting results regarding the time performance and not precision, recall and f-measure as all results from all systems were equal to 1.0 except for OntoIdea (mostly for the Spaten dataset) that were between 0.91 to 0.99.

Task SLL: Small (LineStrings/LineStrings) Figure 7 presents the time performance for TomTom and Spaten datasets for AML, OntoIdea, Silk and RADON systems for 200 instances. RADON has the best performance in most cases except *Touches* and *Instersects* relations, followed by AML and OntoIdea, while Silk seems to need the most time mainly for the TomTom dataset for *Touches* and *Intersects* relations and for both datasets for *Overlaps*.

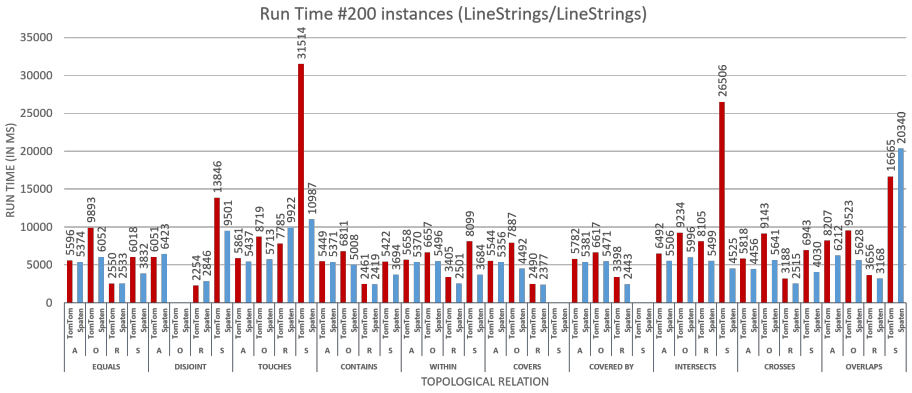

Fig. 7: Time performance for TomTom & Spaten SLL Task for AML(A), OntoIdea(O), Silk(S) and RADON(R) systems

[13] https://github.com/silk-framework/silk/issues/57

[14] http://master.project-hobbit.eu

Task LLL: Large (LineStrings/LineStrings) Figure 8 presents the time performance for TomTom and Spaten datasets for AML, OntoIdea, Silk and RADON systems for the 2K instances dataset. In contrast to Figure 7 we have a more clear view of the capabilities of the systems. In this experiment, RADON and Silk have similar behaviour as in the case of the small dataset, but this time it is more clear that the systems need much more time to match instances from the TomTom dataset. RADON has still the best performance in most cases. AML has the next best performance and is able to handle cases better than other systems (e.g. *Touches* and *Intersects*). AML also hits the platform time limit in the case of *Disjoint*. While the time performance of OntoIdea was close to RADON and AML in the smaller dataset, AML is not able to handle the larger dataset.

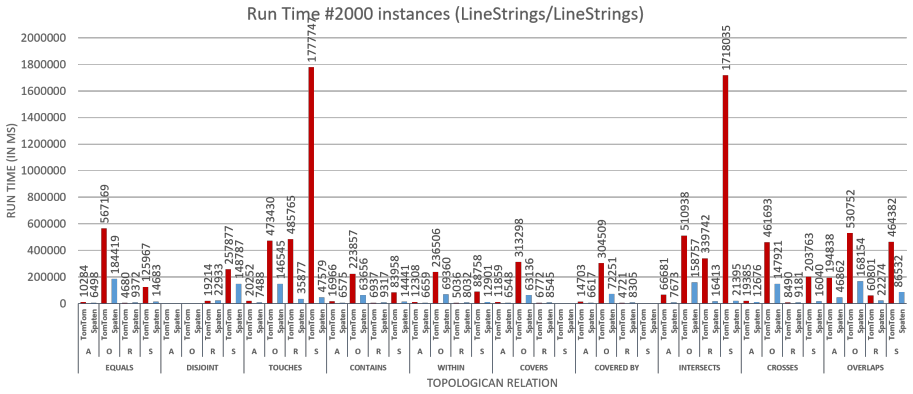

Fig. 8: Time performance for TomTom & Spaten LLL Task and for AML(A), OntoIdea(O), Silk(S) and RADON(R) systems

Task SLP: Small (LineStrings/Polygons) Figure 9 presents the time performance for TomTom and Spaten datasets for AML, Silk and RADON for 200 instances (LineStrings/Polygons or Polygons/LineStrings depending on the relation). In contrast to the two first tasks, RADON has the best performance for all relations. AML and Silk have minor time differences and, depending on the case, one is slightly better than the other. All the systems need more time for the TomTom dataset but due to the small size of the instances the time difference is minor.

Task LLP: Large (LineStrings/Polygons) Figure 10 presents the time performance for TomTom and Spaten datasets for AML, Silk and RADON for the 2K instance dataset (LineStrings/Polygons or Polygons/LineStrings depending on the relation). RADON again has the best performance in all cases. AML hits the platform time limit in *Disjoint* relations on both datasets and is better than Silk in most cases except *Contains* and *Within* on the TomTom dataset where it needs an excessive amount of time.

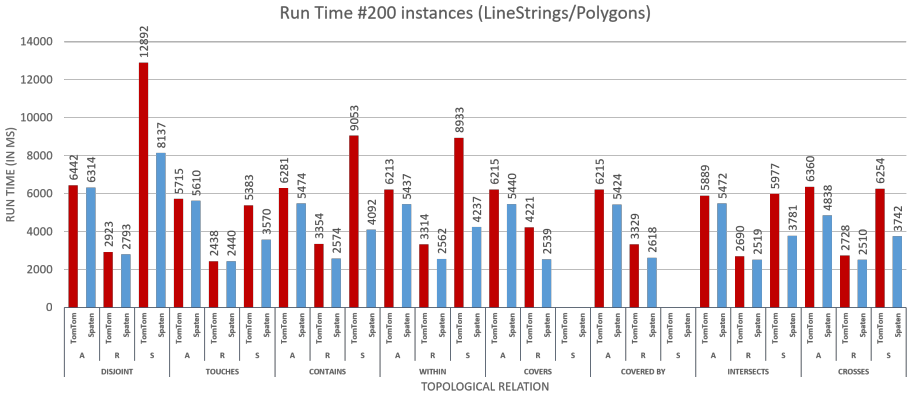

Fig. 9: Time performance for TomTom & Spaten SLP Task and for AML(A), Silk(S) and RADON(R) systems

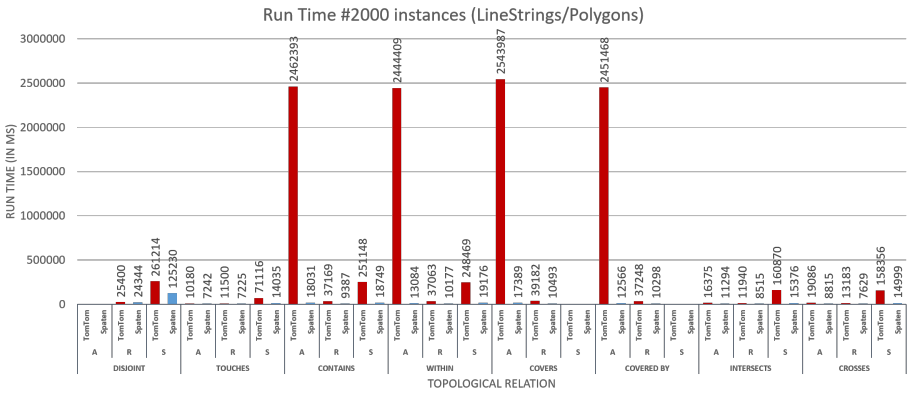

Fig. 10: Time performance for TomTom & Spaten LLP Task and for AML(A), Silk(S) and RADON(R) systems

Discussion

Taking into account the executed experiments we can identify the capabilities of the tested systems as well as suggest some improvements. All the systems participated in most of the test cases except OntoIdea that did not participate in Tasks SLP and LLP and in experiments for the *Disjoint* relation. Also Silk did not participate in *Covers* and *Covered By* experiments.

RADON is the only system that addressed all the tasks, while it can be improved for the *Touches* and *Intersects* relations for the Tasks SLL and LLL and it also has the best performance for the SLP and LLP tasks. AML performs extremely well in most cases. It can be improved in the cases of *Covers/Covered By* and *Contains/Within* when it comes to LineStrings/Polygons Tasks and also in *Disjoint* relations where it hits the platform time limit. Silk can be improved for the *Touches*, *Intersects* and *Overlaps* relations and for the SLL and LLL tasks and for the *Disjoint* relation in SLP and LLP Tasks. OntoIdea can handle small datasets efficiently, but its performance deteriorates when it comes to larger datasets.

In general, all systems needed more time to match the TomTom dataset than the Spaten one, due to the smaller number of points per instance in the latter. Comparing the LineString/LineString to the LineString/Polygon Tasks we can say that all the systems needed less time for the first in *Contains*, *Within*, *Covers* and *Covered by* relations, more time for the *Touches*, *Instersects* and *Crosses* relations, and approximately the same time for the *Disjoint* relation. Thus, depending on the test case we can choose the appropriate system.

7 Conclusions and Future Work

In this paper we presented *SPgen*, a *Spatial Benchmark Generator* that checks whether spatial link discovery systems can identify DE-9IM (Dimensionally Extended nine-Intersection Model) topological relations between LineStrings and Polygons. To the best of our knowledge, such benchmarks do not exist while the number of spatial link discovery systems that identify links for spatial datasets are limited. We evaluated four systems (AML, OntoIdea, RADON and Silk) using *SPgen* to assess and identify their capabilities. In future work, we aim to implement DE-9IM relations for all possible combinations of different geometries (Polygons/Polygons, combination with Points, LineStrings and Polygons, etc.). In addition, we plan to add more data generators in order to test *SPgen* for different use cases.

Acknowledgments

The work presented in this paper was funded by the H2020 project HOBBIT (#688227).

References

1. A.-C. Ngonga Ngomo. On link discovery using a hybrid approach. *Journal on Data Semantics*, 1(4):203–217, 2012.
2. 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.
3. C. Strobl. *Encyclopedia of GIS*, chapter Dimensionally Extended Nine-Intersection Model (DE-9IM), pages 240–245. Springer, 2008.
4. P. Boncz, T. Neumann, and O. Erling. TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark. In *TPC-TC*, pages 61–76. Springer, 2013.
5. I. F. Cruz, F. P. Antonelli, and C. Stroe. AgreementMaker: efficient matching for large real-world schemas and ontologies. *VLDB Endownment*, 2(2):1586–1589, 2009.
6. I. F. Cruz, C. Stroe, F. Caimi, A. Fabiani, C. Pesquita, F. M. Couto, and M. Palmonari. *Using agreementmaker to align ontologies for OAEI2011*, volume 814, pages 114–121. 2011.
7. A. Khiat and M. Mackeprang. I-Match and OntoIdea results for OAEI 2017. In *OM*, page 135, 2017.
8. M.-A. Sherif, K. Dreßler, P. Smeros, and A.-C. Ngonga Ngomo. RADON - Rapid Discovery of Topological Relations. In *AAAI*, 2017.
9. P. Smeros and M. Koubarakis. Discovering Spatial and Temporal Links among RDF Data. In *LDOW*, 2016.
10. G. Garbis, K. Kyzirakos, and M. Koubarakis. Geographica: A benchmark for geospatial rdf stores (long version). In *ISWC*, pages 343–359. Springer, 2013.
11. S. Ray, B. Simion, and A. D. Brown. Jackpine: A benchmark to evaluate spatial database performance. In *ICDE*, pages 1139–1150. IEEE, 2011.
12. D. Kolas. A benchmark for spatial semantic web systems. In *SSWS*, 2008.
13. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, 2005.
14. N. W. Paton, M. H. Williams, K. Dietrich, O. Liew, A. Dinn, and A. Patrick. VESPA: A benchmark for vector spatial databases. In *BNCD*, pages 81–101. Springer, 2000.
15. O. Gunther, V. Oria, P. Picouet, J. M. Saglio, and M. Scholl. Benchmarking spatial joins a la carte. In *SSDM*, pages 32–41. IEEE, 1998.
16. M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The Sequoia 2000 storage benchmark. In *ACM SIGMOD Record*, volume 22, pages 2–11. ACM, 1993.
17. J. Patel, J. Yu, N. Kabra, K. Tufte, B. Nag, J. Burger, J. Kupsch, et al. Building a scaleable geo-spatial dbms: technology, implementation, and evaluation. In *ACM SIGMOD Record*, volume 26, pages 336–347. ACM, 1997.
18. T. D. Doudali, I. Konstantinou, and N. Koziris. Spaten: a Spatio-Temporal and Textual Big Data Generator. In *IEEE Big Data*, pages 3416–3421, 2017.
19. C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *ECIR*, pages 345–359. Springer, 2005.