NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE STUDIES
“DATA SCIENCE AND INFORMATION TECHNOLOGIES”

MASTER THESIS

Transforming into RDF and Interlinking Big Geospatial Data

George E. Mandilaras

Supervisor: **Manolis Koubarakis,** Professor

ATHENS

February 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
“ΕΠΙΣΤΗΜΗ ΤΩΝ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΕΣ ΠΛΗΡΟΦΟΡΙΑΣ”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μετατροπή σε RDF και Διασύνδεση Μεγάλων Γεωχωρικών Δεδομένων

Γεώργιος Ε. Μανδηλαράς

Επιβλέπων: **Μανόλης Κουμπαράκης,** Καθηγητής

ΑΘΗΝΑ

Φεβρουάριος 2021

MASTER THESIS

Transforming into RDF and Interlinking Big Geospatial Data

George E. Mandilaras
R.N.: DS1190012

SUPERVISOR: **Manolis Koubarakis,** Professor

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μετατροπή σε RDF και Διασύνδεση Μεγάλων Γεωχωρικών Δεδομένων

Γεώργιος Ε. Μανδηλαράς
Α.Μ.: DS1190012

ΕΠΙΒΛΕΠΩΝ: Μανόλης Κουμπαράκης, Καθηγητής

ABSTRACT

In the era of big data, a vast amount of geospatial data has become available from government agencies, businesses and research projects. In most cases, this data does not follow the linked data paradigm and the conventional methods for transforming it into linked data has been proved ineffective due to its large volume. For this purpose, we extended GeoTriples, an open source tool developed by our group, to be able to massively transform big geospatial data into RDF graphs, using Apache Spark. Furthermore, by transforming it into RDF, we can interlink it with other linked data and further populated the Linked Open Data Cloud. In this work, we also present novel algorithms for batch and progressive Geospatial Interlinking, as well as how we have parallelized them in the system DS-JedAI, that runs on top of Apache Spark. In the end, we perform detailed evaluation of both systems and we show that they can operate on big geospatial data effectively.

SUBJECT AREA: Semantic Web

KEYWORDS: big data, geospatial data, Spark, RDF graphs, geospatial interlinking

ΠΕΡΙΛΗΨΗ

Στην εποχή των μεγάλων δεδομένων, μια μεγάλη ποσότητα γεωχωρικών δεδομένων είναι διαθέσιμη στο διαδίκτυο, προερχόμενη από κρατικές υπηρεσίες, εταιρίες και ερευνητικά έργα. Στις περισσότερες περιπτώσεις, αυτά τα δεδομένα δεν ακολουθούν το πρωτόκολλο των διασυνδεδεμένων δεδομένων και οι συνηθισμένοι μέθοδοι μετατροπείς τους έχουν αποδειχθεί ανεπαρκής, εξαιτίας του μεγάλου τους όγκου. Για αυτό τον λόγο, επεκτείνουμε το εργαλείο GeoTriples ώστε να μπορεί να μετατρέψει μεγάλα γεωχωρικά δεδομένα σε RDF γράφους, χρησιμοποιώντας το Apache Spark. Επιπλέον, μετατρέποντας τα δεδομένα σαν RDF τριπλέτες, μπορούμε να τα διασυνδέσουμε με άλλα υπάρχοντα συνδεδεμένα δεδομένα και να εμπλουτίσουμε περαιτέρω το σύννεφο των Ανοικτών Διασυνδεδεμένων Δεδομένων (Linked Open Data cloud). Οπότε, σε αυτήν την εργασία παρουσιάζουμε επίσης κάποιους καινοτόμους αλγορίθμους για συνολική ή βαθμιαία διασύνδεση γεωχωρικών δεδομένων, αλλά και πως τους έχουμε παραλληλοποιήσει στο σύστημα DS-JedAI, το οποίο δουλεύει πάνω στο Apache Spark. Στο τέλος, εκτελούμε αναλυτική αξιολόγηση των συστημάτων και αποδεικνύουμε ότι μπορούν να διαχειριστούν μεγάλα γεωχωρικά δεδομένα αποτελεσματικά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σημασιολογικός Ιστός

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: μεγάλα δεδομένα, γεωχωρικά δεδομένα, Spark, RDF γράφοι, γεωχωρική διασύνδεση

To my family

ACKNOWLEDGEMENTS

Firstly, I would like to thank professor Manolis Koubarakis for his guidance and for giving me the opportunity to be part of the AI research group and work on this project. I would also like to thank George Papadakis for constantly advising me during the whole procedure. The present work was co-funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 825258 (ExtremeEarth).

CONTENTS

1 INTRODUCTION **13**

2 PRELIMINARIES **16**

2.1 Semantic Web . . . **16**

2.2 Geospatial data . . . **17**

2.3 Systems for Big Data management . . . **20**

2.4 Summary . . . **22**

3 RELATED WORK **23**

3.1 Transformation into RDF graph . . . **23**

3.2 Geospatial Interlinking . . . **24**

3.3 Summary . . . **25**

4 Transforming Big Geospatial Data into Linked Data **26**

4.1 GeoTriples . . . **26**

4.2 Transformation of big geospatial data . . . **27**

4.3 Evaluation . . . **29**

4.4 Summary . . . **32**

5 Geospatial Interlinking[1] **33**

5.1 Geospatial Interlinking At large (GIA.nt) . . . **33**

5.2 Progressive Geospatial Interlinking . . . **36**

5.2.1 Progressive GIA.nt . . . 37

5.2.2 Geometry Top-k . . . 39

5.2.3 Reciprocal Geometry Top-k . . . 40

5.3 Massive Parallelization . . . **43**

5.4 Evaluation . . . **45**

5.4.1 Evaluation of GIA.nt . . . 45

5.5 Evaluation of progressive algorithms . . . **46**

5.6 Summary . . . **50**

6 POLAR USE-CASE[2] **51**

6.1 Ice Monitoring 51
6.2 Approach 52

7 CONCLUSION 54

ACRONYMS 55

REFERENCES 55

LIST OF FIGURES

Figure 1 An example of an RDF graph 17

Figure 2 The Linked Open Data Cloud 18

Figure 3 Geospatial Interlinking 19

Figure 4 System architecture of GeoTriples 27

Figure 5 The GeoTriples-Spark architecture 28

Figure 6 CSV experiments 30

Figure 7 ESRI shapefiles experiments: Transformation of big shapefiles . . . 31

Figure 8 Space tiling 34

Figure 9 System architecture of DS-JedAI 44

Figure 10 Comparison between GIA.nt and GeoSpark using only the `intersects` relation 46

Figure 11 Evaluation of all progressive methods 47

Figure 12 The results as they are presented to the user. 52

LIST OF TABLES

Table 1 ESRI Shapefile experiments: Transformation of multiple shapefiles of varying sizes 31

Table 2 Large scale experiments with CSV documents 32

Table 3 Large scale experiments with ESRI shapefiles 32

Table 4 Technical characteristics of the real datasets for Geospatial Interlinking. 45

Table 5 Evaluation of progressive methods using all weighting schemes and 5M budget 49

Table 6 Evaluation of all progressive methods using all weighting schemes and 10M budget 49

1. INTRODUCTION

In the recent years, a vast amount of geospatial data has become available on the Web, originating from numerous sources. From crowed-sourced projects such as OpenStreetMap[1], geospatial search engines like Google Maps, data hubs like the ESRI Open Data Hub[2] and earth observation imagery projects like the Copernicus[3] and the US Landsat program[4].

Researchers and practitioners have started publishing geospatial data as linked data, interlinking them and further populating the Linked Open Data (LOD) cloud. For example, the project LinkedGeoData[5] [4] was the first project to add a spatial dimension to the Semantic Web by collecting information from OpenStreetMap (OSM) and converting it into linked data, gathering more than 3 billion geographic entities and 20 billion RDF triples. Furthermore, projects such as TELEIOS[6], LEO[7], MELODIES[8] and Copernicus App Lab[9] have published a number of geospatial datasets that are Earth Observation (EO) products such as the CORINE Land Cover[10] and Urban Atlas[11].

This has led to the development of geospatial Knowledge Graphs such as YAGO2geo[12] [25]. YAGO2geo is a recently developed Knowledge Graph that extends the YAGO2 [17, 18] Knowledge Base with precise geospatial information originated from various official government sources, such as the Ordnance Survey[13] of the United Kingdom and the National Boundary Dataset[14] (NBD) of USA, but also from volunteered open data of OSM. Another big collection of geospatial data represented as linked data is Geographica 2.0 [20] which consists of almost half a billion RDF triples. Geographica 2.0 has gathered information from various open datasets (e.g., Greek Administrative Geography Dataset[15], Geonames[16], DBpedia[17], etc.) and it is used as benchmark to examine the performance of triples-stores.

Publishing geospatial data as linked data, enables users to leverage the advantages of using ontologies and to utilize Semantic Web technologies. Furthermore, enables users to take advantage of other linked open data by interlinking them. In order to interlink geospatial data, we need to discover the topological relations that may exists among their geometries, a process known as *Geospatial Interlinking* or *Link Discovery*. However, Geospatial Interlinking is consider to be a complex process, as a naive approach would require to examine all the entities with each other, leading to a quadratic complexity ($O(n^2)$). Such complexity is prohibitive especially when we are dealing with big data.

[1] https://www.openstreetmap.org/
[2] https://hub.arcgis.com/search
[3] https://www.copernicus.eu/
[4] https://www.usgs.gov/core-science-systems/nli/landsat
[5] http://linkedgeodata.org/About
[6] http://www.earthobservatory.eu/
[7] http://www.linkedeodata.eu/
[8] https://www.melodiesproject.eu/
[9] https://www.app-lab.eu/
[10] https://land.copernicus.eu/pan-european
[11] http://kr.di.uoa.gr/#datasets
[12] http://yago2geo.di.uoa.gr
[13] https://www.ordnancesurvey.co.uk/
[14] https://www.sciencebase.gov/catalog/item/4f70b219e4b058caae3f8e19
[15] http://linkedopendata.gr/dataset/greek-administrative-geography
[16] http://www.geonames.org/
[17] https://wiki.dbpedia.org/online-access/DBpediaLive

In many cases, geospatial data is considered as *big data* as it satisfies the *5Vs* principle[18]. The 5Vs principle is termed as the characteristics of big data and defines the followings properties:

1. *Volume* refers to the size of data. If the volume of data is very large, then it is actually considered as a big data. This means whether particular data can actually be considered as big data or not, is dependent upon the volume of data.
2. *Velocity* refers to the high speed of accumulation of data. This determines the potential of data that how fast the data is generated and processed to meet the demands.
3. *Variety* refers to the nature of data that may be of diverse types and collected from different sources. Generally, big data is classified as structured, semi-structured and unstructured data.
4. *Veracity* or also commonly known as *Validity*, refers the assurance of quality or credibility of the collected data. Decision-making that depends on data requires from it to be credible and to originate from reliable sources.
5. *Value*, which refers to how worthy is the data and to its impact on the business model and decision-making.

Geospatial data satisfies the 5Vs principle. For instance, the Copernicus project has already produced a large amount of EO products (i.e. high Volume) and keeps generating new products on a regular basis (i.e. high Velocity). In more details, the Open Access Hub [19] has released almost 13M of EO products in 2019 and generates tens of thousands new EO products on a daily basis. Regarding Variety, geospatial data may be stored either in raster or vector formats and may originate from a plethora of different sources. The Validity of the geospatial data strictly depends on its source. For instance there are sources that may contain noisy data (e.g., OpenStreetMap), but on the other hand there are sources that produce high quality geospatial data, like the EO products produced by the Copernicus project. Last but not least, geospatial data is of significant importance as it is used in a variety of applications, especially when it is interlinked with other data (e.g., satellite data with in-situ observations).

Consequently, dealing with geospatial data requires adequate resources and specially designed tools able to handle big geospatial datasets consisting of complex geometries. In this work, we focus on two problems that combine big geospatial data and linked data. The first one concerns the transformation of big geospatial data into linked data, and the second one focuses on developing algorithms able to perform geospatial interlinking in a large scale. In more details, the contributions of this work are the following:

- We design and implement the system *GeoTriples-Spark*[20], which is an extension of GeoTriples that runs on top of Apache Spark and enables the transformation of big geospatial data into linked data. GeoTriples-Spark is an open source project, licensed under the Apache license version 2.0.

[18] https://www.bbva.com/en/five-vs-big-data/
[19] https://scihub.copernicus.eu/
[20] https://github.com/LinkedEOData/GeoTriples

- We evaluate our system using datasets of varying input sizes, in different scenarios, and compare its performance with its main competitors: GeoTriples-Hadoop [27] and TripleGeo-Spark [34]. We also show that GeoTriples-Spark is able to transform terabytes of data in a reasonable amount of time, when no other system has been proven to be able to do so.
- We introduce *GIA.nt*, an algorithm for Holistic Geospatial Interlinking and also progressive algorithms that prioritize certain geometry pairs that are more likely to relate.
- We present the system *DS-JedAI* which implements the parallelized versions of these algorithms based on Apache Spark, and we evaluate all approaches through a thorough experimental analysis that involves six real, large-scale datasets.

The structure of the document is the following: in Chapter 2 we present preliminaries, basic concepts necessary for better understanding of this work. Then, in Chapter 3 we present related work regarding the transformation of geospatial data into RDF, and the Geospatial Interlinking. In Chapter 4 we present and evaluate GeoTriples-Spark and we show that it is capable of transforming up to *terabytes* of input data. In Chapter 5 we introduce novel batch and progressive algorithms for Holistic Geospatial Interlinking, and how we have parallelize them in the system *DS-JedAI* using Apache Spark. Furthermore, we present detailed evaluation of these algorithms. In Chapter 6 we present a use-case in which we combine these two systems in order to improve ice monitoring in the Arctic. Finally, in Chapter 7, we summarise the whole work and we present our future plans.

2. PRELIMINARIES

In this chapter we introduce some basic concepts, like the Semantic Web and its related terms, the geospatial data and the topological relations, as well as systems and frameworks necessary for big data management.

2.1 Semantic Web

The *Semantic Web* is a vision about an extension of the existing World Wide Web, which provides software programs with machine-interpretable metadata of the published information and data. In other words, the existing content and data on the Web will be further extended with data descriptors. As a result, computers will able to make meaningful interpretations similar to the way humans process information to achieve their goals

The ultimate ambition of the Semantic Web, as its founder Tim Berners-Lee [1] sees it, is to enable computers to better manipulate information on our behalf. He further explains that, in the context of the Semantic Web, the word “semantic” indicates machine-processable or what a machine is able to do with the data. Whereas “Web” conveys the idea of a navigable space of interconnected objects with mappings from URIs (Uniform Resource Identifiers) to resources.

Fundamental for the adoption of the Semantic Web vision was the development of a set of standards established by the international standards body – the World Wide Web Consortium (W3C):

- *Resource Description Framework(RDF)* is a simple language for describing objects and their relations in a graph.
- *SPARQL Protocol and RDF Query Language (SPARQL)* is the main protocol and query language for RDF data.
- *Uniform Resource Identifier(URI)* is a string of characters designed for unambiguous identification of resources and extensibility via the URI scheme.

In more details, RDF is a family of W3C specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modelling of information that is implemented in Web resources, using a variety of syntax notations and data serialization formats, such as N-Triples, Turtle, N3, JSON-LD, etc.

The RDF data model is similar to classical conceptual modelling approaches (such as entity–relationship or class diagrams). It is based on the idea of making statements about resources (in particular Web resources) in expressions of the form subject–predicate–object, known as RDF triples. The subject denotes a resource or an entity, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. Similarly, objects can also denote other entities or literal values like a name or a date. Both subject, predicates and objects (in case they describe an entity) are defined using HTTP URIs denoting entities within the Web. Using HTTP URIs in RDF statements makes information more structured and more

[1] `https://www.w3.org/People/Berners-Lee/`

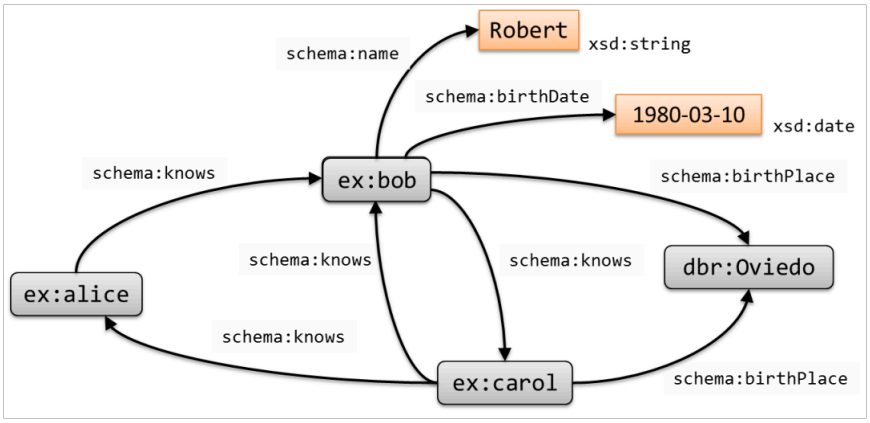


Figure 1: An example of an RDF graph

meaningful to software programs, allowing them to interact with the Web the same way as people do.

A collection of RDF triples forms an RDF graph, like the one displayed in Figure 1. If the entities of an RDF graph represent real-word objects, events, or abstract concepts then the graph is considered to be a *Knowledge Graph (KG)* . In a KG, the relations between the entities have formal semantics that allow both people and computers to process them in an efficient and unambiguous manner. Furthermore, the information can be explored via structured queries (e.g. using SPARQL). Probably, the most famous KG is *DBpedia* which contains facts extracted from the structured information of Wikipedia pages like info-boxes and articles metadata.

Moreover, many KGs contain information from different sources and from other KGs, interlinking their entities and creating a network of accumulated knowledge. For instance, DBpedia is connected with multiple other KGs like YAGO, Wikidata and Geonames. As a result, a vast network of interconnected KGs has been created, which is known as *Linked Open Data (LOD)* Cloud[2] (Figure 2) and it is one of the principals and integral parts of the Semantic Web. Additionally, it is interesting to mention that a big portion of LOD concerns on geographical entities like administrative units.

2.2 Geospatial data

Here we provide some information about geospatial data, topological relations and we define the problem of Geospatial Interlinking.

Geospatial data can exist in raster or vector forms and is usually accompanied by their metadata. Raster data is made up of pixels (also referred as grid cells), it is usually regularly-spaced and pixels are associated with a value (continuous) or class (discrete). Some well-known formats for storing raster data are GeoTIFF, an industry standard file for images from GIS and satellite remote sensing applications, and ESRI Grid files, a proprietary format developed by ESRI. Vector data are made up of vertices and edges and are composed by three basic geometry types: points, lines and polygons. They are commonly available in formats such as ESRI shapefile, GeoJSON, KML and GML documents and in spatially-enabled RDBMS like PostGIS. The ESRI shapefile format is a group of files consisting of three mandatory components, a main file that contains the

[2] https://lod-cloud.net/

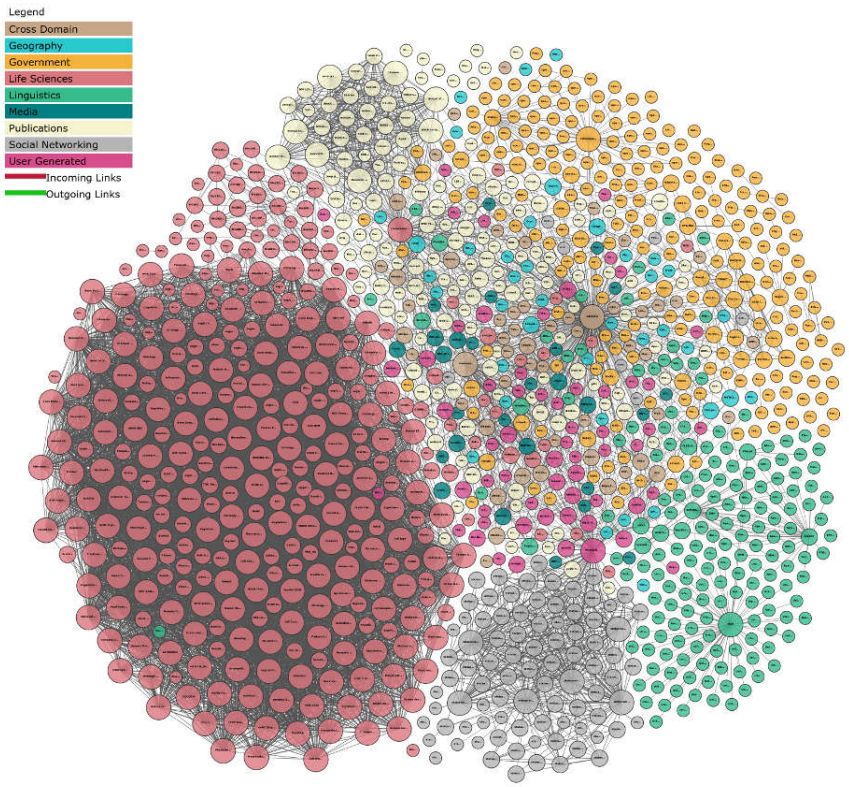


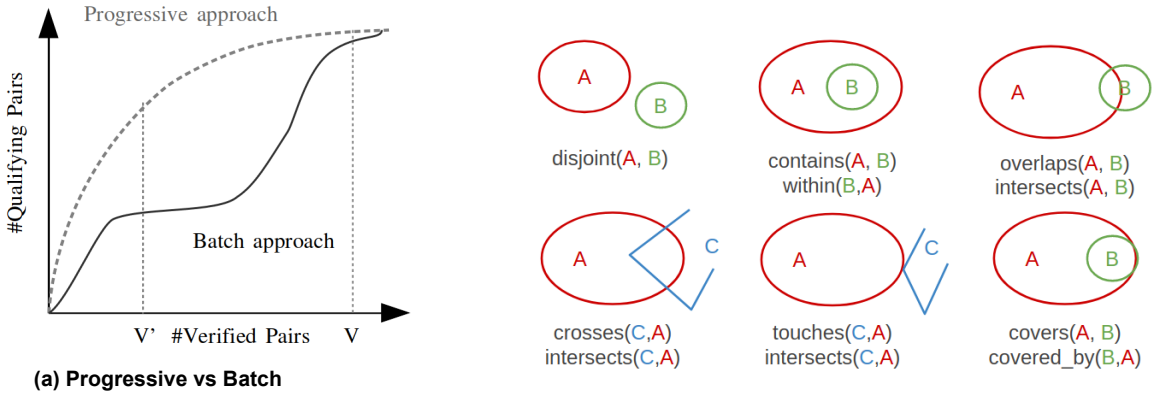
Figure 2: The Linked Open Data Cloud

geometries as lists of vertices, an index file and a dBASE file that contains the thematic features of each record. The shapefile format is widely preferred because it requires less storage space and it is easier to read and write, because it is compatible with many libraries (e.g., JTS Topology Suite[3]) and Geographic Information Systems (GIS). GeoJSON is an open standard format designed for representing simple geographical features, along with their non-spatial attributes. There are several formats of GeoJSON, but the standard specification is based on RFC 7946[4]. Moreover, CSV files can also store geospatial data as vectors, by containing complex geometric types expressed as Well Known Text[5] (WKT), a text markup language for representing vector geometry.

Geospatial data can also be represented as RDF, based on the OGC[6] GeoSPARQL[7] standard. GeoSPARQL defines a vocabulary for representing geospatial data in RDF, and it defines an extension to the SPARQL query language for processing geospatial data. The geometries are represented as literals using either WKT or KML.

Topological relations describe qualitative properties that characterize the relative position of spatial objects. The most common topological model is the *Dimensionally Extended 9-Intersection Model (DE-9IM)* [14, 9, 8] which is used to describe the spatial relations of two regions. DE-9IM defines an $3x3$ Intersection Matrix (IM) based on the interior, exterior and the boundaries of geometries, in order to express all the possible topological relations. The Intersection Matrix for the geometries s and t is defined as:

[3] https://github.com/locationtech/jts
[4] https://tools.ietf.org/html/rfc7946
[5] https://www.ogc.org/standards/wkt-crs
[6] https://www.ogc.org/standards/sfs
[7] https://www.ogc.org/standards/geosparql



(a) Progressive vs Batch approach

(b) Examples of spatial relations

Figure 3: Geospatial Interlinking

$$IM(s,t) = \begin{bmatrix} dim(I(s) \cap I(t)) & dim(I(s) \cap B(t)) & dim(I(s) \cap E(t)) \\ dim(B(s) \cap I(t)) & dim(B(s) \cap B(t)) & dim(B(s) \cap E(t)) \\ dim(E(s) \cap I(t)) & dim(E(s) \cap B(t)) & dim(E(s) \cap E(t)) \end{bmatrix}$$

,

where dim is the dimension of the intersection ($\cap$) of the interior (I), boundary (B), and exterior (E) of geometries s and t. The dimension of non-empty sets ($\neg\emptyset$) are denoted with the maximum number of dimensions of the intersection, specifically 0 for points, 1 for lines and 2 for areas. Empty sets are expressed using F. Hence, the domain of the model is $\{0, 1, 2, F\}$.

This way we can express all the possible topological relations between two geometries, but most applications focus on just ten relations, which can be expressed by the English language. Those are the followings:

1. $\texttt{Intersects}(s,t)$ suggests that s and t share at least one point in their interior or boundary.
2. $\texttt{Contains}(s,t)$ means that s lies inside t such that only their interiors intersect.
3. $\texttt{Within}(s,t)$ means that t contains s.
4. $\texttt{Covers}(s,t)$ indicates that s lies inside t such that their interiors or their boundaries intersect.
5. $\texttt{Covered\_by}(s,t)$ means that t Covers s.
6. $\texttt{Equals}(s,t)$ means that the interiors of s and t intersect, but no point of s intersects the exterior of t and vice versa.
7. $\texttt{Touches}(s,t)$ indicates that the two geometries share at least one point, but their interiors do not intersect.
8. $\texttt{Crosses}(s,t)$ indicates that the two geometries share some but not all interior points and that the dimension of their intersection is smaller than that of at least one of them.
9. $\texttt{Overlap}(s,t)$ differs from $\texttt{Crosses}(s,t)$ in that the two geometries have the same dimension, and so does their intersection.

10. `Disjoint`(s,t) designates that s and t share no interior or boundary point.

Figure 3b displays some examples of topological relations between two geometries.

Geospatial Interlinking is the procedure of discovering the topological relations between the geometries of two datasets (source and target). For instance, to look for geometry pairs that intersect or pairs of nearby points. Given that, each topological relation r is a predicate evaluating to true or false. We can define Geospatial Interlinking as:

Definition 1 (Geospatial Interlinking) *Given a source dataset S, a target dataset T and a topological relation r, discover the set of links $L_r = \{(s,r,t)|s \in S \wedge t \in T \wedge r(s,t)\}$.*

However, in linked data we want to discover all the topological relations between the geometries. So, instead of examining individual topological relations, we use the DE-9IM topological model in order to concurrently find all the topological relations between the two datasets, except the `Disjoint`. `Disjoint` is hard to compute because the vast majority of geometry pairs typically pertains to unrelated geometries. Furthermore, we can imply `Disjoint` from the absence of the other relations. We call the process of discovering all topological relations as *Holistic Geospatial Interlinking* and we define it as:

Definition 2 (Holistic Geospatial Interlinking) *Given a source dataset S, a target one T, and the set of non-trivial topological relations R, derive the set of links $L_R = \{(s,r,t)|s \in S \wedge t \in T \wedge r \in R \wedge r(s,t)\}$ from the Intersection Matrix of geometry pairs.*

In this work, we also examine methods that solve the Geospatial Interlinking task in a *progressive*, i.e., pay-as-you-go, manner, when we have limited time or computational resources. We assume that the available resources for progressive Geospatial Interlinking are defined in terms of the number of geometry pairs that are actually verified. The goal of a progressive algorithm is to prioritize geometry pairs that are more likely to relate. As displayed in Figure 3a, a progressive approach should discover more qualifying pairs in the first V' verifications, than a batch approach. However, after examining all geometry pairs, both approach will conclude to the same qualifying pairs.

2.3 Systems for Big Data management

The management of big data is a difficult procedure which cannot be applied using conventional systems that process data sequentially. Such systems would face storage or memory issues, while would also require an extreme amount of time for processing. Consequently, for processing big data, we use distributed systems able to operate in clusters of computers. Regularly, such systems are based on the *MapReduce* [11] framework.

MapReduce is a processing technique and a programming model for distributed computing based on Java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always

performed after the map job. The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes.

MapReduce is one of the three main components of *Apache Hadoop*. Hadoop is a framework that allows distributed processing of large datasets across clusters of computers, providing distributed storage and computation. Hadoop consists of three main components: the MapReduce, the *(*HDFS), and *YARN*. The *Hadoop Distributed File System (HDFS)* [38] is the file-system used by Hadoop and allows to store big datasets distributedly, by splitting the datasets into chunks and storing them into different machines. *YARN*(which stands for Yet Another Resource Negotiator) is the resource manager and job scheduler of Hadoop. Hadoop is widely used for processing big data, but recently the MapReduce has been replaced with other more powerful distributed processing engines, like *Apache Spark* and *Apache Flink*, which are based on MapReduce.

An extension of Hadoop is *Hops* which runs in the data platform *Hopsworks*[8] [21], developed by the Swedish startup LogicalClocks[9]. Hopsworks has its own resource manager *HopsYarn* but its key component is the file-system HopsFS [29]. HopsFS is a new distribution of HDFS, that replaces the single node in-memory metadata service of HDFS, with a no-shared state distributed system built on a NewSQL database. Hence, it allows to create larger clusters while maintaining client latencies low. As a result, HopsFS is capable of surpassing 1 million file system operations per second, which is at least 16 times higher throughput than HDFS [22].

Probably, the most popular distributed processing engine is *Apache Spark*[10] [46]. Spark is an open-source, distributed, general-purpose, cluster-computing framework that uses a *master/worker* architecture. There is a *Driver* program that talks to a single coordinator called master which manages the worker nodes in which *Executors* run. Driver is responsible to split the job into tasks, to schedule them to run on executors and to coordinate the overall execution. Executors are distributed agents that execute tasks in parallel (or sequentially). At the core of Apache Spark is the notion of data abstraction as a distributed collection of objects, known as *Resilient Distributed Datasets* (RDD) [45]. RDDs allow the user to apply a series of transformations (i.e. map, filters, etc.), creating a lineage graph which will not be executed before calling an action (i.e. count, write to file, etc.). All transformations and actions are performed in parallel, as each Executor is assigned with a portion of the overall data known as partition and the execution of the transformation linkage graph runs inside a task. The number of concurrent tasks is configurable and it is defined by the user and the available resources. Both systems discussed in this work, are developed on top of Apache Spark.

Another very popular distributed processing engine is *Apache Flink*[11] [5]. Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Unbounded streams have a start but no defined end and provide data as it is generated, while bounded streams is finite streams with a defined start and end. Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale.

[8] https://github.com/logicalclocks/hopsworks
[9] https://www.logicalclocks.com/
[10] https://spark.apache.org/
[11] https://flink.apache.org/

2.4 Summary

In this chapter, we introduce the concept of Semantic Web, as well as we provide information about geospatial data and the topological relations between geometries. Furthermore, we introduce the problem of Geospatial Interlinking and also the Holistic Geospatial Interlinking where we try to discover all the topological relations between two datasets. Moreover, since both of the systems discussed in Chapters 4 and 5 are designed to process big data and to run on distributed clusters of computers, we also present some state-of-the-art systems for big data management.

3. RELATED WORK

In this chapter, we present related work regarding the transformation of structured and semi-structured data into RDF graphs and Geospatial Interlinking.

3.1 Transformation into RDF graph

Regarding the transformation of relational and non-relational data as RDF graphs, there are two main approaches: direct mapping and using mapping rules. Direct mapping[1] is a straightforward approach where the tables of the relational database become classes, the attributes are mapped into RDF properties that represent the relation between subject and object and subjects are defined by the identifiers. Even though this is a very simple method, the generated triples are strictly defined by the schema of the relational data. Alternatively, using mapping rules, we define a set of rules which will be applied during the transformation, and define how the input data will be mapped into triples. Two well-known mapping languages are the *R2RML*[2] and *RML*[3] [12] the first of which is a W3C recommendation. R2RML is a language for expressing customized mappings from relational databases into RDF graphs while RML is a more generic mapping language that can express rules that map data with heterogeneous structures (like XML, JSON) into RDF graphs.

Regarding the transformation of geospatial data into RDF, there is not a lot of work available. Geometry2RDF [10] was one of the first tools that enabled users to transform spatially-enabled RDB systems into RDF graphs. Even though this project is no longer maintained, its code-base worked as the basis for the development of other tools that serve this purpose. A different approach appears in [7] which shows how R2RML can be combined with a spatially-enabled relational database in order to transform geospatial data into RDF. However, the transformation of other geospatial data sources like vector forms e.g., shapefiles is not discussed.

TripleGeo[4] [34, 35] is a tool for transforming geospatial features from various sources into RDF graphs, developed in the project GeoKnow[5]. TripleGeo supports the transformation of structured data (e.g., ESRI shapefiles, CSV, GeoJSON, GPX) or semi-structured data (in XML, GML, or KML), as well as from spatially-enabled DBMSs and of less standard formats such as OpenStreetMap data and certain INSPIRE data and metadata. Furthermore, recently in the project SLIPO[6], TripleGeo was further extended with several novel features and specific functionalities to efficiently support the transformation of large point of interests (POIs). This was achieved by extending TripleGeo to run on top of Apache Spark, and therefore currently TripleGeo and GeoTriples-Spark, the system presented in this paper, are the only tools available that support the transformation of big geospatial data into RDF graphs (the author of this work implemented the Spark extension of TripleGeo in collaboration with Spiros Athanasiou and Kostas Patroumpas of the TripleGeo team).

[1] https://www.w3.org/TR/rdb-direct-mapping/
[2] https://www.w3.org/TR/r2rml/
[3] https://rml.io/
[4] https://github.com/SLIPO-EU/TripleGeo
[5] http://geoknow.eu/Welcome.html
[6] http://slipo.eu/

3.2 Geospatial Interlinking

Most of the existing related work focuses on spatial join which examines a specific relation (i.e., `intersects`, `contains`, etc.) at a time. Instead, in Holistic Geospatial Interlinking the target is to discover all the possible topological relations.

Silk-spatial [40][7] is a well-known tool for spatial link discovery, implemented as an extension of *Silk* [42] [8] which is an open source framework for integrating heterogeneous data sources. Silk-spatial employs a static space tiling approach that defines a fixed EquiGrid on Earth's surface, and compares the geometries that coexist in the same tiles. However, since the tiling technique is independently of the input data, it leads to tiles that are usually coarse-grained, which means that the number of geometry pairs that are verified is too large. This is partially ameliorated through massive parallelization on top of Apache Hadoop.

RADON [37] is an algorithm implemented in LIMES[9] which is a link discovery framework for the Web of Data. RADON improves on Silk-spatial by building dynamic, fine-grained tiles: in each dimension, the extent of the tiles has a length equal to the average extent of the geometry Minimum Bounding Rectangles (MBRs) in that dimension. Given that every geometry is assigned to all tiles intersecting its MBR, and similarly to Silk-spatial, all geometries that coexist in the same tiles will be examined. However, since the MBRs can intersect with multiple tiles, it leads to redundant geometry pairs that co-occur in multiple tiles. In order to avoid duplicate verifications, RADON maintains a hash-table in memory with all the verified geometries. Even though this is a simple solution, the size of the table can grow significantly when the number of geometric pairs is very big. Thorough experimental evaluation in [36] demonstrates that RADON is significantly faster than Silk-spatial, constituting the state-of-the-art approach. Additionally, RADON in [3] was enriched with the DE-9IM topological model in order to be able to compute all spatial relations at once.

GeoSpark[10] [44] (currently known as Apache Sedona) is an in-memory cluster computing framework developed by the Data Systems Lab[11], in order to support spatial data types, indexes, and processing of large-scale spatial data. GeoSpark provides a geometrical operations library that accesses Spatial RDDs to perform basic geometrical operations like spatial joins based on topological relations (e.g., `overlap`, `intersect`). System users can leverage the newly defined SRDDs to effectively develop spatial data processing programs in Spark. To better process geometries in parallel, GeoSpark enables users to spatial partition their data so all the potentially spatial related geometries to co-occur in the same partition. To spatial partition, GeoSpark collects and builds a spatial index (e.g., `Quad-Trees`, `KDB-Trees`, `R-Trees`, etc.) based on a sample of data, which is then used in order to assign each geometry to the appropriate partition. This way, in spatial joins, GeoSpark examines only the geometry pairs that co-exist in the same partitions.

More importantly, in this work we focus on pay-as-you-algorithms that try to optimize the processing order of geometry pairs within a limited budget (in terms of computational resources). Previous work on progressive computation of spatial joins focuses on stream processing and is not relevant to our problem. A problem related to this work is *Progressive*

[7] `http://silk.di.uoa.gr/`
[8] `http://silkframework.org/`
[9] `https://aksw.org/Projects/LIMES.html`
[10] `http://geospark.datasyslab.org/`
[11] `https://www.datasyslab.net/`

Entity Resolution [33, 39, 43], where the goal is to detect matches, i.e., entity profiles describing the same real-world object, in a pay-as-you-go manner.

3.3 Summary

In this chapter we discuss related work about the two subjects of this work, transformation of geospatial data and Geospatial Interlinking. In both cases, we see that there are extensions of existing tools developed either with Apache Spark or Apache Hadoop, in order to be able to process big data. However, regarding Geospatial Interlinking, only RADON can use the DE-9IM model to discover all topological relations, but it is not designed for processing big data. Furthermore, to the best of our knowledge, there is no existing framework for progressive Geospatial Interlinking.

4. TRANSFORMING BIG GEOSPATIAL DATA INTO LINKED DATA

In this chapter we present *GeoTriples-Spark*[1], an open source system for the massive conversion of big geospatial data into RDF graphs. GeoTriples-Spark is an extension of GeoTriples [27] that runs on top of Apache Spark, and it is able to transform up to terabytes of data into RDF graphs. In this chapter we introduce GeoTriples and how we have extended it with Apache Spark. Moreover, we evaluate our system using datasets of varying input sizes, in different scenarios, and compare its performance with its main competitors: GeoTriples-Hadoop [27] and TripleGeo-Spark. We show that, in most cases, GeoTriples-Spark decreases the transformation time by approximately 40%.

4.1 GeoTriples

In this section we give a short description of the main components of the tool GeoTriples and how it operates.

GeoTriples is an open source tool developed by our team[2] in the National and Kapodistrian University of Athens for the transformation of geospatial data, from various data sources, into linked data [27]. GeoTriples, currently supports the transformation of spatially-enabled databases (PostGIS and MonetDB), ESRI shapefiles, XML documents (hence GML documents), KML, GeoJSON and CSV documents. It consists of three components: a mapping generator that given an input file it generates a mapping file containing the mapping rules, a mapping processor that applies the mapping rules in order to map each instance of the input data into the appropriate RDF triples, and an stSPARQL/GeoSPARQL evaluator that can performs stSPARQL or GeoSPARQL queries in a spatial enabled database given a mapping file. In addition, the first element of GeoTriples is a connector which is responsible for providing an abstraction layer that allows the other components to transparently access the input data regardless of the format of the source. The execution of GeoTriples comprises three steps. In the first step we use the mapping generator which creates a mapping file containing the mapping rules. Then as a second optional step the user may edit the mapping file, so the produced triples will adopt any vocabulary or ontology the user wants. Finally, in the last step follows the transformation of the input file, which applies the mapping rules and maps the input data into RDF triples. The produced graph by default, will be compliant with the GeoSPARQL vocabulary and can be manifested in any of the popular RDF syntaxes such as Turtle, RDF/XML, Notation3 or N-Triples. Figure 4 presents the architecture of GeoTriples.

The mappings produced by GeoTriples consists of two triples maps: one for handling non-geometric (thematic) data, and one related to the geospatial data. The triples map that handles thematic information defines a logical table that contains the attributes of the input data source and a unique identifier for the generated instances. Combined with a URI template, the unique identifier is used to produce the URIs that will be the subjects of the produced triples. For each column of the input data source, GeoTriples generates an RDF predicate according to the name of the column and a predicate-object map. This map generates predicate-object pairs consisting of the generated predicate and the column values. The triples map that handles geospatial information defines a logical table with a

[1] `https://github.com/LinkedEOData/GeoTriples`
[2] `http://ai.di.uoa.gr/`

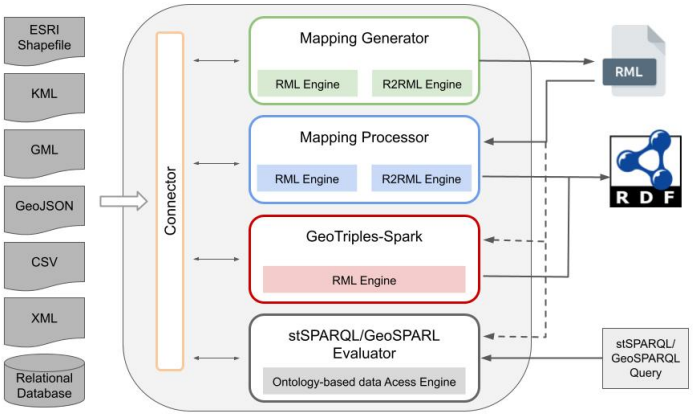


Figure 4: System architecture of GeoTriples

unique identifier similar to the thematic one. The logical table contains a serialization of the geometric information according to the WKT format, and all attributes of the geometry that are required for producing a GeoSPARQL compliant RDF graph. Hence, if the input is an ESRI shapefile, GeoTriples constructs RML mappings with transformations that invoke GeoSPARQL/stSPARQL extension functions. If the input is a relational database, GeoTriples constructs SQL queries that utilize the appropriate spatial functions of the Open Geospatial Consortium[3] [16] standard that generate the information required.

In the beginning of the execution, GeoTriples parses the input mappings and extracts the content of the logical table using the appropriate way (e.g. a SELECT query). If the subject map is a template-valued term map or a column-valued term map, the related columns are extracted and stored in memory. Then, the processor iterates over all predicate-object maps, and for each one it extracts all template- and column-valued term maps. These term maps are cached in memory along with the position that they appear on. Afterwards, the processor extracts all the features that are referenced by the term maps that appear in the subject, predicate and object positions for the current predicate map and iterates over the results. For each predicate and object value in the result row, a new RDF triple is constructed.

4.2 Transformation of big geospatial data

In this section we present *GeoTriples-Spark*. GeoTriples-Spark is developed to run on top of Apache Spark and it enables the massive conversion of big geospatial data into RDF graphs. The input big geospatial data can be provided as multiple separate files which will be transformed concurrently or as a big single file. However, in case of multiple files, all of them must comply to the same schema and mapping rules. Currently, GeoTriples-Spark supports the transformation of CSV, GeoJSON and ESRI shapefiles. Figure 5 displays its architecture.

The first component of GeoTriples-Spark is the Reader, which employs the appropriate libraries regarding the format of the source to load the input data into a Spark Dataset, which is a distributed immutable collection of data organized into named columns. When a dataset is stored in a distributed filesystem (like HDFS), it is split into multiple chunks of constant size. When GeoTriples-Spark starts, the Reader loads these data chunks

[3] `https://www.ogc.org/standards/sfs`

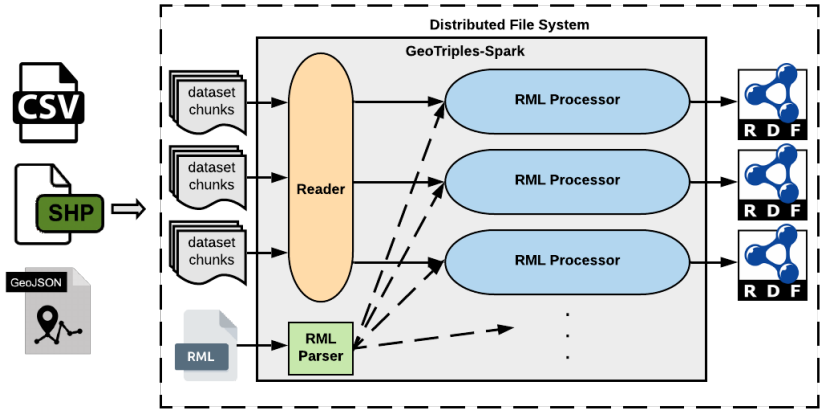


Figure 5: The GeoTriples-Spark architecture

into partitions, which will be transformed as individual units in parallel. Users can change the default number of partitions in order to increase parallelization, but this may invoke data shuffling. In order to load ESRI shapefiles and GeoJSON, we use the library *GeoSpark*. After initializing the Dataset, a new column is inserted containing a monotonically increasing unique index which will be combined with a URI template to form the subjects of produced triples. This is the default way of constructing subjects, but the user can overwrite it by editing the mapping file. Moreover, this generated index is not consecutive as this would require to have counted all the instances of the input data, which would add an overhead in the execution. Last, before the transformation starts, the Reader loads and extracts the mapping rules from the mapping file, and broadcasts them so they will be available in all the nodes.

The transformation starts by a map stage where, for each partition, an RML processor is initialized, which is in charge of transforming all the instances of the input partitions into RDF triples. The RML processor iterates over the records of its input partition and applies the mapping rules loaded by the mapping file, generating the corresponding triples. Note that the produced triples significantly over-exceed the size of the initial dataset and hence we avoid to reduce the partitions in order to construct a single partition containing all the produced triples, as this would either require an excessive amount of available memory or it would lead to memory errors. Additionally, collecting all the data into one node is not considered to be a good tactic. Consequently, after the transformation, the triples of each partition are stored in individual files, which can easily be concatenated as the produced triples follow the N-Triples RDF syntax. Hence, the whole procedure is a highly parallelized one as each running process needs to load its corresponding data, transform it based on the mapping rules and store it.

Spark does not load the input data as a whole, but as multiple partitions distributed across the cluster. For each partition, Spark spawns a task which its target is to transform the partition into RDF triples and to store them in a file. Except of the broadcast of the mapping rules, there is no need for further data shuffling during the whole procedure as each partition already contains all the necessary information for performing the transformation. Furthermore, the whole procedure is memory friendly, as it does not need to maintain neither the initial dataset nor the produced triples in memory. When the produced triples of a record is generated, they are directly written in the target file. Therefore, GeoTriples-Spark is capable of transforming large amount of data with minimum memory requirements.

The parallelization of the whole procedure is based on the number of partitions and on available resources (physical cores/threads). More partitions means the more parallelized the procedure can be (according to the number of concurrent executed threads), but also it means that it will have to transform a smaller chunk of data, as the initial dataset will have been divided into more partitions of fewer records. The number of partitions is determined by the file format of the source, its size and the configuration settings of the filesystem, but it can also be configured by the user. However, increasing the initial number of partitions may invoke data shuffling, and hence, in a distributed cluster, it would probably invoke network and disk I/O, which can significantly impact the performance of the procedure.

The transformation of CSV and GeoJSON documents is quite similar as these filetypes are considered text files and therefore they are loaded as multiple partitions by Spark because they are distributedly stored across the cluster. The geometry feature in CSV files is expected to be in WKT and hence it does not require any further processing. Regarding GeoJSON, the application loads them as simple JSON files that follow the GeoJSON specification of RFC 7946. The FeatureCollection of the GeoJSON file is loaded into a Spark Dataset which is then extended with a new column containing the geometries as WKT.

The case of ESRI shapefiles is a little more complicated. As mentioned, we load them using GeoSpark, whose shapefile reader always loads the input shapefile into a single partition[4]. Since shapefiles are composed of multiple files, in order to load them, we first need to merge all the related component files into one. This is happening because all the related component files must be located in the same datanode to utilize the shapefile index and the related attributes. Loading shapefiles from a distributed filesystem is a well-known problem and it has been studied extensively in [2]. Hence, if users want to parallelize the transformation of a single shapefile, it requires to re-partition in order to re-distribute the input data as multiple partitions in the cluster. Consequently, this will probably have a negative effect in the performance, as re-partitioning will invoke data shuffling. However, shapefiles are considered to be small files. Actually there is a 2GB size limit for any of its component files[5] and it is common to store data as multiple shapefiles. Thus, we have enabled GeoTriples-Spark to be able to transform multiple shapefiles concurrently loaded as multiple partitions.

4.3 Evaluation

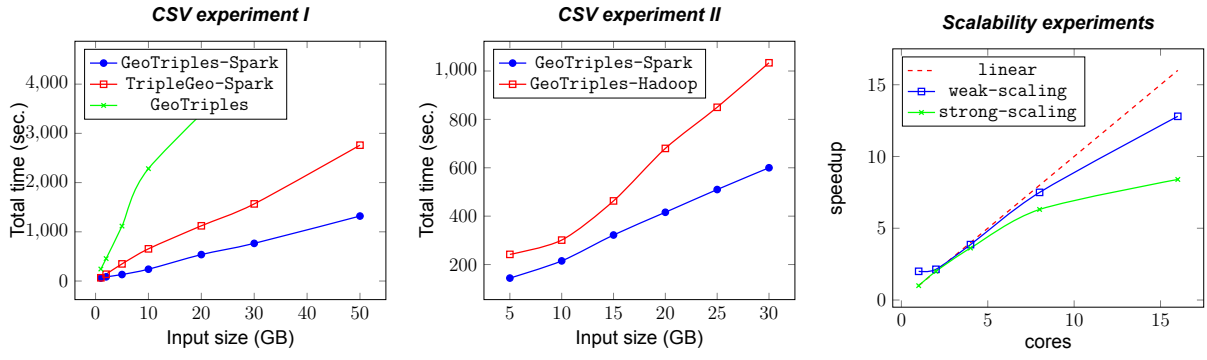
For the evaluation of GeoTriples-Spark, we compare it with the centralized version of GeoTriples, with the Hadoop-based implementation of GeoTriples [6] and with the Spark-based implementation of TripleGeo[7]. The following experiments concern the performance of the systems against varying input sizes, the scalability of the system and also the performance of the system in transforming big geospatial data into RDF. Regarding the comparison with GeoTriples-Hadoop using shapefiles, we reproduce the same experiments presented in [27], while for the comparison using CSV files we perform larger scale experiments with bigger files.

[4] https://github.com/DataSystemsLab/GeoSpark/issues/356

[5] https://desktop.arcgis.com/en/arcmap/latest/manage-data/shapefiles/geoprocessing-considerations-for-shapefile-output.htm

[6] https://github.com/dimitrianos/GeoTriples-Hadoop

[7] https://github.com/SLIPO-EU/TripleGeo/tree/master/src/eu/slipo/athenarc/triplegeo/partitioning



(a) Performed in standalone machine

(b) Performed in Hadoop cluster

(c) Performed in standalone machine

Figure 6: CSV experiments

We performed the experiments in three different environment, a Hadoop cluster, a standalone machine that runs Apache Spark, and a large scale cluster that runs the Hospworks data platform. Ideally we would like to avoid the standalone machine, but we were not able to execute TripleGeo on HDFS, as it was not able to read the configuration file and to store the output triples. Hence we compare the Spark- and Hadoop-based implementations of GeoTriples in the Hadoop cluster, and we compare the Spark-based implementations of GeoTriples and TripleGeo in the standalone machine. The Hadoop cluster consists of four nodes with 8 cores each of Intel(R) Xeon(R) CPU E5-2650 v3 at 2.30GHz and 8GB of memory. The standalone machine contains 32 virtual cores[8] at 2.20GHz and 128GB of memory. The large scale cluster is a very powerful cluster provided by Logical Clocks, containing approximately 1000 CPU cores at 2.40GHz, 12TB of RAM and 1PB of storage. The data used for the experiments are extracts of the OpenStreetMap project that are publicly available from GEOFABRIK[9], which we further edit and replicate in order to increase the input size.

Figures 6a and 6b show the performance of GeoTriples-Spark for varying input CSV file sizes against the Hadoop-based implementation of GeoTriples and the Spark-based implementation of TripleGeo. In the experiment of Figure 6a, both GeoTriples-Spark and TripleGeo-Spark, load the input data as 32 partitions which are transformed concurrently by 32 tasks. In the experiment of Figure 6b, we did not change the initial number of loaded partitions of the datasets, as it would invoke network I/O which we wanted to avoid. In both experiments of Figure 6, GeoTriples-Spark outperforms its competitors and we can also observe that as the size of input data increases, the effectiveness of GeoTriples-Spark becomes even clearer, particularly for the last datasets where the execution time decreases up to 47% compared to TripleGeo-Spark and 42% compared to GeoTriples-Hadoop. The results are similar when using GeoJSON documents as input.

Figure 6c depicts the scalability experiments with regards to *strong* and *weak* scaling[10]. In *strong scaling* we examine how the overall computational time of the job scales as we increase the number of available processing cores. In *weak scaling* we examine the speedup while increasing both the job size and the number of processing elements. In strong scaling experiment the size of the job is 15GB. In weak scaling the input size is equivalent to the number of active cores (i.e., 2 core $\rightarrow$ 2GB, 4 cores $\rightarrow$ 4GB). In weak scaling we can observe that the execution is almost linear but we can notice that there

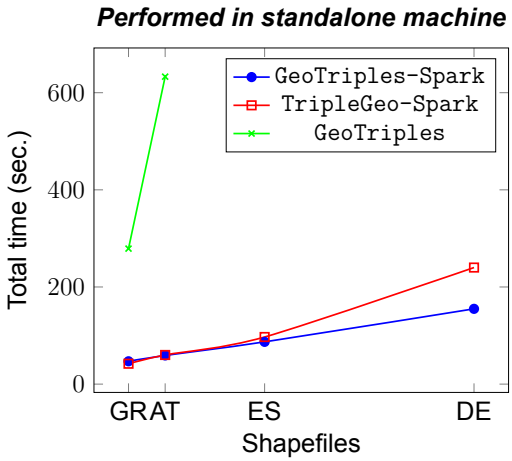
[8]The system uses hyperthreading hence it has 16 physical cores

[9]`http://download.geofabrik.de/`

[10]`https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/`

Dataset	*Size*
GR	440MB
AT	764MB
ES	1.7GB
DE	3.7GB

(a) Shapefile datasets



(b) SHP Experiment

Figure 7: ESRI shapefiles experiments: Transformation of big shapefiles

Table 1: ESRI Shapefile experiments: Transformation of multiple shapefiles of varying sizes

Dataset	Size (MB)	Times loaded	GeoTriples-Spark (sec.)	GeoTriples-Hadoop (sec.)
Andorra	888	15	345	370
Australia	247	60	382	499
Ukraine	2	1000	428	1002

is a small deceleration as the number of cores increases. Similar deceleration we also observe in strong scaling experiment. The main reason for this is because the Executors read and write in the same disk, hence more active cores lead to bigger latencies in disk I/O.

Regarding the experiments with ESRI shapefiles, we evaluate the performance of GeoTriples- Spark in two kinds of experiments.

In the first experiment we compare the performance of GeoTriples-Spark and TripleGeo-Spark in the transformation of big ESRI shapefiles. The shapefiles are displayed in Table 7a and contain data of the road system of the corresponding countries (i.e. Greece, Austria, Spain and Germany) originated from OSM. As mentioned, most shapefiles are relatively small files, hence in order to create bigger ones, we merged multiple shapefiles into one. The largest shapefile we use (*DE*) contains the whole road-system of Germany and it was created by merging the shapefiles of the road-system of the states of Germany. In these experiments, both tools re-partition the input data into 32 partitions which are all transformed in parallel. The results are presented in Figure 7b. Both systems perform well and quite similarly, but in the last and largest dataset, clearly GeoTriples-Spark outperforms TripleGeo-Spark, as it requires 62.5% of the time TripleGeo-Spark needs to transform it.

In the second type of experiments, we examine and compare the performance of Spark- and Hadoop-based implementations of GeoTriples regarding the transformation of multiple shapefiles concurrently. Similarly to GeoTriples-Spark, GeoTriples- Hadoop loads the data of a shapefile into a single mapper, but in contrast with the Spark implementation, GeoTriples-Hadoop cannot re-distribute the load to other mappers, as mentioned in [27]. Therefore, GeoTriples- Hadoop is good for transforming multiple shapefiles where each

Table 2: Large scale experiments with CSV documents

Dataset	Times loaded	Input Size	#Executors	Output Size	Total time (in minutes)
100GB.csv	1	100GB	41	840.1GB	3.3
250GB.csv	1	250GB	60	2.1TB	6.6
250GB.csv	2	500GB	65	4.1TB	13
250GB.csv	4	**1TB**	70	8.3TB	26
250GB.csv	8	**2TB**	80	16.6 TB	50

Table 3: Large scale experiments with ESRI shapefiles

Dataset	Times loaded	Input Size	#Executors	Output Size	Total time (in minutes)
AT	153	100 GB	20	427.7 GB	4.3
AT	381	250 GB	30	1068.6 GB	9.9
DE	136	500 GB	15	2.5 TB	17
DE	258	**1TB**	27	5.1 TB	34

one is assigned to a different mapper, but it is incapable of transforming shapefiles where their size exceeds the available memory of mappers. In this experiment we load three different shapefiles of varying sizes multiple times, in order to evaluate how the tools perform when the goal is to transform multiple small, medium and large files. The results are displayed in Table 1 and we can see that both tools perform similarly regarding the big and the medium shapefiles, with GeoTriples-Spark performing slightly better. However, we observe significant difference in the last dataset where GeoTriples-Spark advances over GeoTriples-Hadoop, as it requires less than 50% of the time it needs.

Last, there is the large scale experiments, presented in Tables 2 and 3. For the experiments with CSV documents, we constructed datasets up to 250GB which we replicate and load multiple times. Likewise, for the experiments of ESRI shapefiles we replicate and load the AT and DE shapefiles. The memory requirements of each Executor are the minimum, as neither the input data nor the generated triples are cached in memory. Furthermore, there is no need for large Spark execution memory[11] since there is little to none data shuffling. So, in these experiments we equipped each Executor with 2GB of memory. In the end, we managed to transform 2TB of CSV input in less than an hour and 1TB of shapefiles input in less than half an hour.

4.4 Summary

In this chapter, we describe how we can transform big geospatial data into RDF using GeoTriples-Spark. In more details, we introduce GeoTriples and GeoTriples-Spark and we perform detailed evaluation between GeoTriples-Spark and its competitors. In the end, we show that GeoTriples-Spark not only outperforms its competitors, but we also show that it is capable of transforming up to terabytes of input data, in reasonable amount of time. GeoTriples-Spark is used in the project ExtremeEarth[12] in order to transform data extracted form satellite images, into linked data.

[11] `https://spark.apache.org/docs/latest/tuning.html`
[12] `http://earthanalytics.eu/`

5. GEOSPATIAL INTERLINKING[1]

In this chapter, we discuss the novel batch algorithm *Geospatial Interlinking At large (GIA.nt)*, and we introduce some progressive geospatial interlinking algorithm that prioritize geometry pairs based on certain weighting schemes. All these algorithms are implemented in a parallelized version in the open-source system *DS-JedAI*[2] (Distributed Spatial JedAI), that runs on top of Apache Spark. In the end, we compare GIA.nt with GeoSpark, and we evaluate the progressive algorithms using the appropriate metrics.

5.1 Geospatial Interlinking At large (GIA.nt)

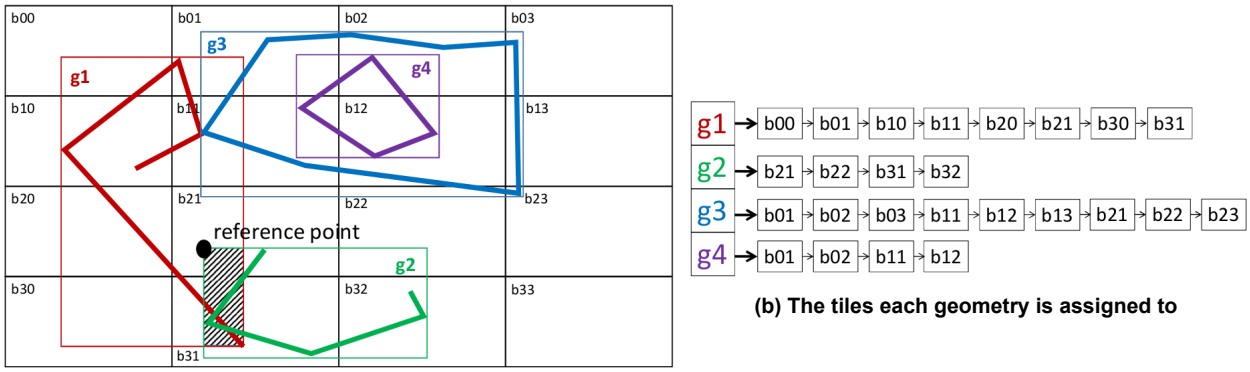
In certain parts, GIA.nt is quite similar to RADON. Given two geometry sets S (called source) and T (called target), the goal of GIA.nt is to discover all topological relations among the geometries of these two geometry sets. As we have already mention, verifying all geometry pairs is a highly expensive procedure that leads to quadratic complexity ($O(n^2)$). Additionally, verifying geometry pairs that are not topologically close, is a waste of time since those pairs will not relate. So, in order to avoid redundant verifications, we use a dynamic space tiling, like the one used in RADON. The granularity of space tiling is defined by the average extend of the MBRs of the geometries, in both dimensions. For instance, in Figure 8a, each geometry is assigned to the tiles its MBR overlap, and GIA.nt will verify all the geometry pairs that coexist in the same tiles. We call this process as the *Filtering Step*.

From Figure 8, we can see that the geometries g_1 and g_2 share multiple tiles, hence, by default, this verification would be performed multiple times. To avoid duplicate verifications, RADON maintains the executed verifications in a hash-table, and executes new ones only if they do not exist in this hash-table. The goal with GIA.nt is to be executed in a distributed environment, so, maintaining such hash-table would require to keep it updated in all parallel processes. Such approach would be expensive and would drastically worsen the performance. So, instead, we use the *Reference Point (RF) technique* introduced in [13], which states that for each pair of candidates, the verification is carried out only in the tile that contains the top-left corner of the intersection of their MBRs. For example, in Figure 8, the verification of g_1 and g_2, will be executed once, only in the tile b_{21} which contains the reference point.

To further filter redundant verifications, GIA.nt implements another filtering technique which we call *IntersectingMBR*. In this filtering technique we examine if the MBRs of the candidate geometries intersect, and if they do not, then we omit the verification. This derives from the fact that if the MBRs of the geometry pairs do not intersect, then the actual geometries cannot possible intersect (i.e. they will be disjoint). Performing verifications with MBRs is significantly easier and faster than with actual geometries, as it reduces the problem to just simple comparisons between a few points. IntersectingMBR of the geometries s and t is performed by checking the following condition:

[1] Parts of this chapter appear in the paper *"Progressive, Holistic Geospatial Interlinking"*, in proceedings of the WebConference 2021 (WWW '21)

[2] `https://github.com/GiorgosMandi/DS-JedAI`



(a) Space tiling and filtering techniques

(b) The tiles each geometry is assigned to

Figure 8: Space tiling

$$IntersectingMBR = \neg (min(MBR(s).width) > max(MBR(t).width) \vee max(MBR(s).width) < min(MBR(t).width) \vee min(MBR(s).length) > max(MBR(t).length) \vee max(MBR(s).length) < min(MBR(t).length))$$

Another key difference between GIA.nt and RADON, is that in GIA.nt we maintain only the one of the two datasets in the memory (i.e.the source), while the other dataset is loaded iteratively. Consequently, tiling granularity and the Filtering step are based only on the S dataset.

GIA.nt's functionality appears in Algorithm 1. Lines 1-12 apply $Filtering$ to index the source dataset, which is set as the smallest one so as to minimize the memory footprint. In Line 2, the longitude and latitude granularity of tiles are defined as $\Delta_x = mean_{s \in S} MBR(s).width$ and $\Delta_y = mean_{s \in S} MBR(s).length$, respectively,by adapting RADON's approach so that it considers only the S dataset. For each geometry $s \in S$ (Line 3), GIA.nt estimates the diagonal corners of its MBR (Line 4) - the lower left point $(x_1(s), y_1(s))$ and the upper right point $(x_2(s), y_2(s))$. Using them along with Δ_x and Δ_y, it determines the tiles that intersect $MBR(s)$ and should contain s (Lines 5-11).

GIA.nt's verification is applied in Lines 13-29. The next target geometry $t \in T$ is read from the disk (Lines 14-15) and the tiles that contain it are estimated, based on its MBR (Lines 17-28). For each tile, GIA.nt retrieves the source geometries it contains and places them in the local set of candidates T_S (Line 20). As a result, every geometry s that is likely to be related to t appears in T_S. Next, GIA.nt iterates over the geometries of T_S (Line 21) and filters out the redundant verifications using the $IntersectingMBR$ and $ReferencePoint$ techniques (Line 22). In Line 27 computes the corresponding intersection matrix IM. The topological relations that are extracted from IM are added to the list of detected links L (Line 24), which is returned as output (Line 30).

Algorithm 1: GIA.nt

input : the source dataset S, a reader for the target one $rd(T)$ & the set of non-trivial topological relations R

output : the links $L = \{(s, r, t) | s \in S \wedge t \in T \wedge r \in R \wedge r(s, t)\}$

```
/* Filtering step                                                    */
I ← {} ;                                          // Equigrid index structure
(Δx, Δy) ← (mean_{s∈S} MBR(s).width, mean_{s∈S} MBR(s).length);
foreach geometry s ∈ S do
    (x1(s), y1(s), x2(s), y2(s)) ← getDiagCorners(s);
    for i ← ⌊x1(s) · Δx⌋ to ⌈x2(s) · Δx⌉ do
        for j ← ⌊y1(s) · Δy⌋ to ⌈y2(s) · Δy⌉ do
            I.addToIndex(i, j, s);
            j ← j + 1;
        end
        i ← i + 1;
    end
end
/* Verification step                                                 */
L ← {} ;                                          // The set of detected links
while rd(T).hasNext() do
    t ← rd(T).next() ;                            // The current target geometry
    (x1(t), y1(t), x2(t), y2(t)) ← getDiagCorners(t);
    for i ← ⌊x1(t) · Δx⌋ to ⌈x2(t) · Δx⌉ do
        for j ← ⌊y1(t) · Δy⌋ to ⌈y2(t) · Δy⌉ do
            b ← (i, j);
            TS ← I.getTileContents(b);
            foreach geometry s ∈ TS do
                if intersectingMBRs(s, t) & ReferencePoint(b , s, t) then
                    IM ← verify(s, t);
                    L ← L ∪ IM.getRelations();
                end
            end
        end
    end
end
return L;
```

5.2 Progressive Geospatial Interlinking

Although, GIA.nt uses novel techniques that significantly improves RADON in terms of time and space requirements, still the total number of verifications is very big, especially when the input datasets (source and target) are big. Furthermore, examining the relations between a pair of geometries is a very expensive procedure, especially when the geometries consist of numerous points and edges [6]. Therefore, in order to avoid verifications of unrelated geometry pairs, we use progressive algorithms that prioritize the verification of pairs, that are more likely to relate. In this section, we present three progressive algorithms: *Progressive GIA.nt*, *Geometry Top-*k and *Reciprocal Geometry Top-*k. These algorithms take as input a budget BU that indicates the total number of verifications that will be performed, and a weighting scheme W.

In order to quantify the probability of a pair of geometries to relate, these algorithms uses certain weighting schemes that assign a weight to every candidate pair. These schemes produce a numerical estimate of how likely two geometries s and t are to satisfy a non-trivial topological relation, judging exclusively from the tiles that contain them. The more tiles they share, the higher is the weight that is assigned to them and the more likely they are to be topologically related.

The following schemes are defined:

- $Co-occurrence\ Frequency$ (CF): indicates the number of common tiles that are shared by s and t. $CF(s,t) = |B_s \cap B_t|$, where B_k stands for the set of tiles/blocks containing geometry k.
- $Jaccard\ Similarity$ (JS): normalizes the overlap similarity defined by CF, i.e. $JS(s,t) = \frac{|B_s \cap B_t|}{|B_s|+|B_t|-|B_s \cap B_t|}$.
- $Pearson's\ \chi^2\ test$ (χ^2): extends CF by assessing whether two geometries s and t appear independently in the set of tiles. To infer their dependency, it estimates whether the distribution of tiles containing s in B is the same as the distribution if we exclude the tiles that contain t.

Budget BU indicates the total number of verifications that will be performed. All the algorithms use a min-max priority queue of max-size BU, where they store the geometry pairs that have collected the biggest weights. This priority queue is referred as PQ.

5.2.1 Progressive GIA.nt

Progressive GIA.nt turns GIA.nt into a progressive algorithm, in which we maintain a min-max priority queue (PQ) with the BU most promising geometry pairs from the entire input datasets. The functionality of Progressive GIA.nt is exactly the same as with GIA.nt, but instead of verifying geometry pairs after *Filtering*, it weights them based on the input weighting scheme, and inserts them in PQ. Whenever the size of PQ exceeds BU, the pair with the lowest weight is evicted. In the end, the PQ contains the BU top weighted pairs of the entire input datasets.

Algorithm 2 describes the functionality of Progressive GIA.nt. Progressive GIA.nt takes as input the same arguments as GIA.nt, but additionally it requires a budget BU and a weighting scheme W, which can be one of the schemes we mentioned earlier (i.e. CF, JS, $Pearson's\ \chi^2\ test$). The algorithm starts by using the dynamic tiling technique to index the source dataset, just like in GIA.nt (Lines 1-12). Afterwards, does not follow the Verification step like in GIA.nt, but the *Scheduling step*, in which it weights and inserts the candidate pairs, that were not filtered out by the $IntersectingMBR$ and $ReferencePoint$ techniques, into the min-max priority queue PQ of fixed size (Lines 13-35).

To insert a new pair in PQ, its weight must exceed a minimum weight (denoted by $minWeight$), which is initialized as 0.0. As long as PQ is not full, we do not update $minWeight$ and we insert all the new pairs inside PQ. When PQ exceeds its maximum size, we remove the pair with the minimum weight (Line 28), and we update $minWeight$ with the minimum weight that currently there is in PQ (Line 29). So, when PQ is full, a new pair is inserted only if its weight exceeds $minWeight$. In case it does, the new pair is inserted and the pair with the minimum weight is evicted. Finally, $minWeight$ is updated with the minimum weight of PQ (Lines 25-31).

In the end of the Scheduling step, we have populated PQ with the BU candidate pairs that are associated the biggest weights. Afterwards, follows the Verification step, which pops the geometry pairs from PQ in a decreasing order, and computes the IM. The relations extracted from IM is added in L (Lines 36-41).

We choose to use a min-max priority queue instead of a regular priority queue because we want fast access to both smallest and biggest item of queue. When we execute the verifications, we pop the pair with the biggest weight, while when we populate the priority queue we always remove the pair with the smallest weight.

Algorithm 2: Progressive GIA.nt

input : the source dataset S, a reader for the target one $rd(T)$, the set of non-trivial topological relations R, the budget BU and the pair weighting scheme W
output : the links $L = \{(s, r, t) | s \in S \wedge t \in T \wedge r \in R \wedge r(s, t)\}$

```
/* Filtering step                                                    */
I ← {} ;                                          // Equigrid index structure
(Δx, Δy) ← (mean_{s∈S} MBR(s).width, mean_{s∈S} MBR(s).length);
foreach geometry s ∈ S do
    (x1(s), y1(s), x2(s), y2(s)) ← getDiagCorners(s);
    for i ← ⌊x1(s) · Δx⌋ to ⌈x2(s) · Δx⌉ do
        for j ← ⌊y1(s) · Δy⌋ to ⌈y2(s) · Δy⌉ do
            I.addToIndex(i, j, s);
            j ← j + 1;
        end
        i ← i + 1;
    end
end
/* Scheduling step                                                   */
PQ ← MinMaxPriorityQueue(maxsize=BU);
minWeight = 0.0;
while rd(T).hasNext() do
    t ← rd(T).next() ;                          // The current target geometry
    (x1(t), y1(t), x2(t), y2(t)) ← getDiagCorners(t);
    for i ← ⌊x1(t) · Δx⌋ to ⌈x2(t) · Δx⌉ do
        for j ← ⌊y1(t) · Δy⌋ to ⌈y2(t) · Δy⌉ do
            b ← (i, j);
            T_S ← I.getTileContents(b);
            foreach geometry s ∈ T_S do
                if intersectingMBRs(s, t) & ReferencePoint(b , s, t) then
                    w_{s,t} ← W.getWeight(s, t) ;
                    if minWeight < w_{s,t} then
                        PQ ← PQ ∪ {{s, t}, w_{s,t}} ;
                        if BU < PQ.size() then
                            PQ.pop() ;          // Remove pair with minimum weight
                            minWeight = PQ.peekFirst().getWeight() ;   // Update
                        end
                    end
                end
            end
        end
    end
    /* Verification step                                             */
    L ← {};
    while PQ ≠ {} do
        tail ← PQ.popLast();
        IM ← verify(tail.s, tail.t);
        L ← L ∪ IM.getRelations();
    end
end
return L;
```

5.2.2 Geometry Top-k

In *Geometry Top-k* we find the top-k candidate geometries of each geometry, for every $s \in S$ and $t \in T$. For each geometry, we store its top-k candidates in a min-max priority queue and in the end we accumulate them in the global min-max priority queue PQ of max-size BU. Finally, we implement the same procedure as in Progressive GIA.nt, in which we pop the geometry pairs from PQ in a decreasing order, compute the IM and extract the requested relations. The k is calculated by dividing the initial BU by the sum of the sizes of S and T (i.e. $k = \lceil \frac{BU}{|S|+|T|} \rceil$). This require to have pre-computed the size of the target dataset.

To collect the top-k candidate geometries of T, we initialize a min-max priority queue (denoted by PQ_T) of size k. For each geometry $t_m \in T$, we find its candidate geometries and populate PQ_T. After examining all candidates of t_m, we pop all the geometries of PQ_T and insert them in PQ. So, after emptying the PQ_T, we continue to the next geometry t_{m+1} where we repeat the same procedure. This procedure occurs in Lines 39-48 of Algorithm 3.

The discovery of the top-k candidates of each geometry $s \in S$ is a little bit more complicated. For this, we use an array that contains min-max priority queues. In this array, the priority queue in position i will contain the top-k geometries of geometry s_i, which is located in position i of array S. After adding the geometry s_i in the top-k of a geometry t, follows the procedure of adding t in the top-k of s_i. In order to access the top-k of s_i, we use the index i which denotes its position in array S. Furthermore, before inserting, we check if the priority queue of s_i is defined, and in case it is not, we initialize it. So, while populating the top-k of target geometries, we also populate the top-k of source geometries concurrently. This procedure takes place in Lines 49-52.

After examining all the geometries of target, the top-k of source geometries have not yet been inserted in PQ, which currently (i.e., Line 64) contains only the top-k of target geometries. After computing all the top-k of both source and target geometries, we iteratively insert the top-k of source in PQ. However, since PQ already contains the top-k geometries of target, there might be candidate pairs that already exist in PQ. In this case, we do not add the duplicate pairs, but we update their weights, if the existing weights in PQ are smaller than the new ones. The whole procedure takes place in Lines 64-72. Afterwards, follows the verification and the relation extraction.

The functionality of Geometry Top-k is presented in Algorithm 3. In the first 33 Lines, we apply the Filtering step by indexing the source dataset. Then we initialize the min-max priority queues, and their auxiliary variables that store the minimum weight they contain (Lines 34-38). The $ArrPQ_S$ is the array that will contain the priority queues, and their minimum weights will be stored in the array $minWeights_{ArrPQ_S}$, where the value in position i is the minimum weight of the priority queue, located in position i in $ArrPQ_S$. These minimum weights are initialized after initializing their corresponding priority queue (i.e Line 51). The $insert$ function, which is called multiple times, does not simply insert a weighted pair in the specified priority queue, but only if its weight is greater than the minimum weight of the priority queue. Additionally, it maintains the size of the priority queue fixed by removing the pair with the minimum weight, whenever the size exceeds its maximum size. In more details, the $insert$ function does exactly the same as the Lines 25-31 of Algorithm 2.

Regarding the $update$ function used in Line 68, it finds the pair inside the specified priority

queue and updates its weight, in case it exists with a smaller one. However, look up is an expensive action for a min-max priority queue. Therefore, in the actual implementation, this is implemented using an auxiliary hash-set that contains all the top-k pairs of all geometries. Then, the final PQ is constructed using this hash-set. For the economy of space and simplicity, we omit it in the Algorithm.

5.2.3 Reciprocal Geometry Top-k

The last progressive algorithm we present is *Reciprocal Geometry Top-k*, which is very similar to Geometry Top-k. Their main difference is that Reciprocal Geometry Top-k prioritizes only the pairs (s', t') that s' belongs to the top-k of t' and t' belongs to the top-k of s'. To achieve this, we maintain the top-k pairs of each geometry $t \in T$ in a hash-set. Hence, to insert the pair (s_n, t_m) in the PQ, t_m must be one of the top-k of s_n, and s_n must also exist in the hash-set that contains the top-k of t_m.

The Algorithm 4 implements Reciprocal Geometry Top-k. A key element of the algorithm is the array H_T which will contain the hash-sets we mentioned earlier. In more details, the location m of the hash-set will contain the top-k candidates of the geometry t_m, which is located in position m in array T. This will be initialized after populating PQ_T with the top-k pairs of geometry t_m (Lines 41-63). In Lines 61-63, we extract the pairs from PQ_T and put them in the hash-set $H_T[m]$, we also empty PQ_T and set its minimum weight to 0.0. Meanwhile, we also gather the top-k pairs of source geometries using the same procedure as in Geometry Top-k (Lines 51-56). Then, we populate PQ by inserting the pair (s_n, t_m), extracted from priority queue $ArrPQ_S[n]$, only if s_n exists in the hash-set $H_T[m]$, thus if s_n belong to the top-k pairs of t_m (Lines 64-71). Finally, for each pair in PQ, we compute the IM, extract the requested relations and return them in L.

Furthermore, another key difference with top-k is that we do not put the top-k geometries of target in PQ, but we just use them to filter pairs from the top-k geometries of source. Consequently, k is calculated by dividing BU with the number of source geometries. Moreover, since we also find the top-k of the target geometries, we compute a different k (denoted by k_T) by dividing BU with number of target geometries. However, if k was computed as $\lceil \frac{BU}{|S|} \rceil$, the maximum pairs we could get would be BU, but since we filter based on target geometries' top-k, most of the times we would get fewer pairs. Therefore, we calculate k as $\lceil \frac{2*BU}{|S|} \rceil$ and we ensure that we will get BU pairs by defining PQ max-size as BU.

Algorithm 3: Geometry Top-k

input : the source dataset S, a reader for the target one $rd(T)$, the set of non-trivial topological relations R, the budget BU & the weighting scheme W
output : the links $L = \{(s, r, t) | s \in S \wedge t \in T \wedge r \in R \wedge r(s, t)\}$

/* As in Alg. 2, Lines 1-32 */
... k = $\lceil BU/(|T| + |S|) \rceil$;
$PQ \leftarrow$ MinMaxPriorityQueue(maxsize=BU);
$PQ_T \leftarrow$ MinMaxPriorityQueue(maxsize=k);
$ArrPQ_S[] \leftarrow \{\}$; // Array of priority queues
$minWeights_{ArrPQ_S}[] \leftarrow \{\}$; // Array of min weights
$minWeight_{PQ} = 0.0$; $minWeight_{PQ_T} = 0.0$;
foreach *geometry* $t \in T$ **do**
  $(x_1(t), y_1(t), x_2(t), y_2(t)) \leftarrow$ getDiagCorners(t);
  for $i \leftarrow \lfloor x_1(t) \cdot \Delta_x \rfloor$ **to** $\lceil x_2(t) \cdot \Delta_x \rceil$ **do**
    for $j \leftarrow \lfloor y_1(t) \cdot \Delta_y \rfloor$ **to** $\lceil y_2(t) \cdot \Delta_y \rceil$ **do**
      $b \leftarrow$ (i, j);
      $T_S \leftarrow I$.getTileContents(b);
      foreach *geometry* $s_n \in T_S$ **do**
        if *intersectingMBRs*(s_n, t) & *ReferencePoint*(b , s_n, t) **then**
          $w_{s_n,t} \leftarrow W$.getWeight(s_n, t) ;
          insert(PQ_T, $minWeight_{PQ_T}$, $\{\{s_n, t\}, w_{s_n,t}\}$) ;
          /* initialize and insert s_n's top-k */
          if $\neg ArrPQ_S[n]$.*isDefined* **then**
            $ArrPQ_S[n] \leftarrow$ MinMaxPriorityQueue(maxsize=k);
            $minWeights_{ArrPQ_S}[n] = 0.0$;
          end
          insert($ArrPQ_S[n]$, $minWeights_{ArrPQ_S}[n]$, $\{\{s_n, t\}, w_{s_n,t}\}$);
        end
      end
    end
  end
  /* add t top-k in PQ */
  while $PQ_T \neq \{\}$ **do**
    $tail \leftarrow PQ_T$.popLast();
    insert(PQ, $minWeight_{PQ}$, $tail$) ;
  end
  $minWeight_{PQ_T} = 0.0$;
end
/* add top-k of every $s \in S$ in PQ */
foreach $pq_s \in ArrPQ_S$ **do**
  while $pq_s \neq \{\}$ **do**
    $tail \leftarrow pq_s$.popLast();
    if $tail \in PQ$ **then**
      update(PQ, $tail$)
    else
      insert(PQ, $minWeight_{PQ}$, $tail$)
    end
  end
end
... **return** L;

Algorithm 4: Reciprocal Geometry Top-k

input : the source dataset S, a reader for the target one $rd(T)$, the set of non-trivial topological relations R, the budget BU & the weighting scheme W

output : the links $L = \{(s, r, t) | s \in S \wedge t \in T \wedge r \in R \wedge r(s, t)\}$

`/* As in Alg. 2, Lines 1-32 */`

... k = $\lceil (2 * BU)/|S| \rceil$;
k_T = $\lceil (2 * BU)/|T| \rceil$;
$PQ \leftarrow$ MinMaxPriorityQueue(maxsize=BU);
$PQ_T \leftarrow$ MinMaxPriorityQueue(maxsize=k_T);
$H_T[] \leftarrow \{\}$;
$ArrPQ_S \leftarrow \{\}$;
$minWeights_{ArrPQ_S} \leftarrow \{\}$;
$minWeight_{PQ} = 0.0$;
foreach *geometry* $t_m \in T$ **do**
  $(x_1(t), y_1(t), x_2(t), y_2(t)) \leftarrow$ getDiagCorners(t);
  for $i \leftarrow \lfloor x_1(t) \cdot \Delta_x \rfloor$ **to** $\lceil x_2(t) \cdot \Delta_x \rceil$ **do**
    for $j \leftarrow \lfloor y_1(t) \cdot \Delta_y \rfloor$ **to** $\lceil y_2(t) \cdot \Delta_y \rceil$ **do**
      $b \leftarrow (i, j)$;
      $T_S \leftarrow I$.getTileContents(b);
      foreach *geometry* $s_n \in T_S$ **do**
        if *intersectingMBRs(*s_n, t_m*) & ReferencePoint(*b , s_n, t_m*)* **then**
          $w_{s_n,t_m} \leftarrow W$.getWeight(s_n, t_m) ;
          $insert(PQ_T, minWeight_{PQ_T}, \{\{s_n, t_m\}, w_{s_n,t_m}\})$;
          if $\neg\ ArrPQ_S[n]$.*isDefined* **then**
            $ArrPQ_S[n] \leftarrow$ MinMaxPriorityQueue(maxsize=k);
            $minWeights_{ArrPQ_S}[n] = 0.0$;
          end
          insert($ArrPQ_S[n]$, $minWeights_{ArrPQ_S}[n]$, $\{\{s_n, t_m\}, w_{s_n,t_m}\}$) ;
        end
      end
    end
  end
  $H_T[m] \leftarrow$ {PQ_T.items};
  PQ_T.clear() ;
  $minWeight_{PQ_T} = 0.0$;
end
foreach $pq_s \in ArrPQ_S$ **do**
  while $pq_s \neq \{\}$ **do**
    $\{\{s_n, t_m\}, w_{s_n,t_m}\} \leftarrow pq_s$.popLast();
    if $H_T[m]$.*contains(*s_n*)* **then**
      insert(PQ, $minWeight_{PQ}$, $\{\{s_n, t_m\}, w_{s_n,t_m}\}$) ;
    end
  end
end
... **return** L;

5.3 Massive Parallelization

We have implemented these algorithms in a parallelized version according to the MapReduce framework, in the system *DS-JedAI*[3] (Distributed - Spatial JedAI). This system is implemented on top of Apache Spark and can run in any distributed or standalone environment that supports the execution of Apache Spark jobs. Its name originates from the system JedAI[4] (Java gEneric DAta Integration) [32, 24, 31], which is a tool for Entity Resolution, i.e., tries to identify entities that refer to the same real world entity.

DS-JedAI loads both datasets as RDDs that are spatially partitioned based on GeoSpark's spatial partitioning techniques. We can spatial partition the datasets either by using a KDB-Tree or a Quad-Tree spatial index, which is build based on a sample of the source dataset. Both RDDs are partitioned using the same partitioner and thus, the topologically close geometries belong to partitions with the same partition id. The RDDs with the same partition id are then merged such that each partition contains all geometries from both datasets that lie within its area. This way, we ensure that all geometries that are likely to satisfy a topological relation coexist in the same partitions.

For GIA.nt, each Executor receives one of these partitions as input, during the Map phase. It indexes the source geometries and for each target geometry t, it estimates the tiles that intersect its MBR. Using the index, it retrieves the distinct source geometries in these tiles and verifies their topological relations with t. All qualifying pairs are aggregated during Reduce phase.

The granularity of space tiling (i.e., Δ_x, Δ_y) is computed by the Driver, which broadcasts it to the Executors. To compute it, requires information from all geometries of source, and hence this triggers the execution plan of source RDD. By caching it after loading it, we avoid the re-execution of the execution plan so that there is no impact on the performance of the algorithm.

Regarding the progressive algorithms, every Executor receives as input a partition of both input datasets, during the Map phase, and applies Filtering to index the source geometries. Then, it processes the target geometries one by one, estimating their weights with the intersecting source geometries. Then follows the Scheduling step, where each algorithm finds the top BU pairs and verifies them. The qualifying pairs of each Executor are aggregated by the Reduce phase. Initially, each partition computed just a portion of the BU verifications, which we used to call local budget, and it was estimated by dividing the global budget BU among the data partitions in proportion to the source geometries they contain. However, this resulted to fewer verifications as it was estimated by the number of source geometries and not by the total verifications within partitions. Hence, in the current implementation, each partition computes a fixed number of verifications.

Note that no data shuffling is required during Scheduling for the weight estimations, since all necessary information is locally available: every Executor estimates all tiles that should contain every geometry. Thus, each Executor operates independently of the others, promoting concurrency and making the most of massive parallelization.

We should also stress that all algorithms employ the reference point technique to avoid redundant pairs. This is because every geometry that crosses the borders between two

[3] https://github.com/GiorgosMandi/DS-JedAI

[4] https://github.com/scify/JedAIToolkit

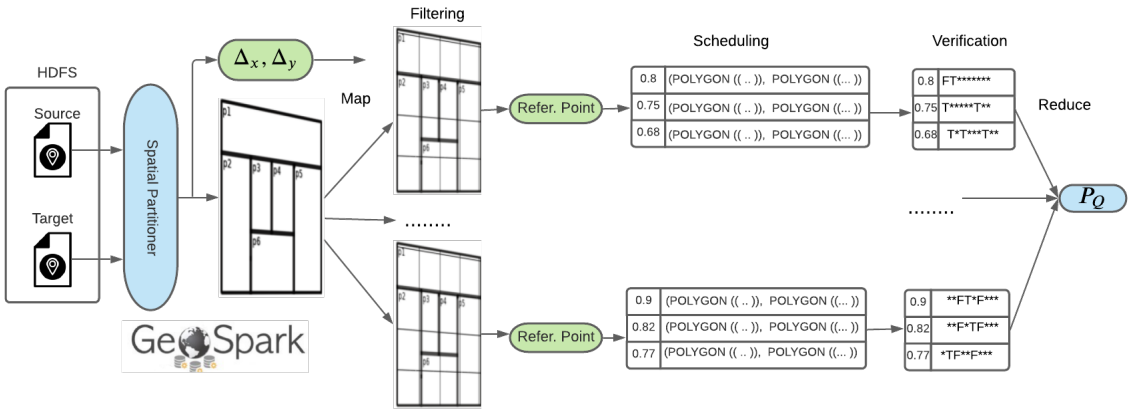


Figure 9: System architecture of DS-JedAI

partitions is added to both of them during spatial partitioning. To avoid the resulting redundancy, both algorithms ensure that every pair is verified only in the partition that contains the top left corner of their intersection.

Finally, it is worth noting that spatial partitioning yields uneven partitions, which are skewed with respect to the volume of data and the corresponding computational cost. That is, some partitions are overloaded and require significant time, while others complete their jobs instantaneously, leaving the corresponding nodes idle. To tackle this issue, both algorithms take special care of the overloaded partitions, whose size exceeds significantly the average size of all partitions. After completing the processing of the well-balanced partitions, the entities of the overloaded partitions are indexed and re-partitioned using a HashPartitioner that is based on tiles id. In this way, geometries indexed in the same tiles will be placed in same partitions, thus missing no candidate pairs. Redundant pairs are again discarded with the reference point technique. This is an effective and efficient strategy as long as it applies to a small portion of the input data, because it requires the duplication of each entity as many times as the numbers its tiles.

Additionally, we have enriched our system with the capability to perform temporal filtering. This way, users can specify the temporal fields of the input datasets and the resulted interlinked pairs will coincide temporally. This means that the difference between the timestamps of geometry pairs will not exceed a pre-defined time margin. By default, this time margin is set to one day, thus the timestamps of the interlinked pairs will be at most 24 hours apart. This time margin is configurable by the users. Such functionality is very important, especially when we process EO products due to the changes in Earth's surface as the time passes by.

Figure 9 outlines the architecture of DS-JedAI. Currently, it supports most of the RDF formats (i.e., N-Triples, Turtle, RDF/JSON and RDF/XML), as well as CSV, TSV, GeoJSON and ESRI shapefiles. To load distributed RDF datasets into RDDs, we use the open-source library SANSA-Stack[5] [28], which is a big data engine for scalable processing of large-scale RDF data. Moreover, the input triples must follow the GeoSPARQL vocabulary. CSV and TSV are loaded using Spark's API and GeoJSON and ESRI shapefiles are loaded using GeoSpark. All RDDs, regardless source, are loaded into RDDs of *SpatialEntities*, which are objects that maintain only the geometries and their

[5] https://github.com/SANSA-Stack/SANSA-Stack

Table 4: Technical characteristics of the real datasets for Geospatial Interlinking.

	D_1	D_2	D_3	D_4	D_5	D_6
Source Dataset	AREAWATER	AREAWATER	Lakes	Parks	ROADS	Roads
Target Dataset	LINEARWATER	ROADS	Parks	Roads	EDGES	Buildings
#Source Geometries	2,292,766	2,292,766	8,419,320	9,961,891	19,592,688	72,339,926
#Target Geometries	5,838,339	19,592,688	9,961,891	72,339,926	70,380,191	114,796,567
Cartesian Product	$1.34 \cdot 10^{13}$	$4.49 \cdot 10^{13}$	$8.39 \cdot 10^{13}$	$7.21 \cdot 10^{14}$	$1.38 \cdot 10^{15}$	$8.30 \cdot 10^{15}$
#Qualifying Pairs	2,401,396	199,122	5,551,014	14,163,325	163,982,135	1,041,562
`#Contains`	806,158	3,792	947,788	6,323,433	12,218,867	276,010
`#CoveredBy`	0	0	3,031,403	48,922	53,758,452	83,936
`#Covers`	832,843	4,692	948,086	6,470,655	12,218,867	276,023
`#Crosses`	40,489	106,823	270,248	6,490,937	6,769	314,708
`#Equals`	0	0	557,465	3,147	12,218,867	18,972
`#Intersects`	2,401,396	199,122	5,551,014	14,163,325	163,982,135	1,041,562
`#Overlaps`	0	0	822,241	45,116	73	54,899
`#Touches`	1,554,749	88,507	1,037,412	1,258,163	110,216,841	332,249
`#Within`	0	0	3,030,790	48,823	53,758,452	82,668
Total Topological Relations	5,635,635	402,936	16,196,447	34,852,521	418,379,323	2,481,027

identifiers. The geometries are represented using the JTS geometry class [6], and the algorithm that computes the DE-9IM is the function $relate$[7].

5.4 Evaluation

For the evaluation of the algorithms, and the overall DS-JedAI system, we compare GIA.nt with GeoSpark and we examine the performance of the progressive algorithms in terms of progressive Recall (PGR), Recall and Precision. Our experiments are performed using real world datasets, widely used in the related literature [15, 41] and publicly available [8]. The technical characteristics of the datasets we use are reported in Table 4.

The datasets contain public data about area hydrography (AREAWATER), linear hydrography (LINEARWATER), roads (ROADS) and all edges (EDGES) in the USA. They also contain the boundaries of all lakes (Lakes), parks or green areas (Parks), roads and streets (Roads) as well as of all buildings (Buildings) around the world. Each column of Table 4 shows statistics for a pair (D_1–D_6) of interlinked datasets.

All experiments were performed in a standalone machine that contains 32 virtual cores[9] at 2.20GHz and 128GB of memory. In all experiments we used 16 Executors with 2 cores each and 7GB of memory. The implementation of DS-JedAI is in Scala 2.12 using Spark 2.4.

5.4.1 Evaluation of GIA.nt

To evaluate the effectiveness of our parallelized implementation of GIA.nt we compare it with GeoSpark. Since GeoSpark does not compute the DE-9IM model, we examine only the `intersects` relation between the geometries. The experiments using GeoSpark was based on the examples provided in its repository[10].

[6] `https://locationtech.github.io/jts/javadoc/org/locationtech/jts/geom/Geometry.html`

[7] `https://locationtech.github.io/jts/javadoc/org/locationtech/jts/geom/Geometry.html#relate-org.locationtech.jts.geom.Geometry-`

[8] `http://spatialhadoop.cs.umn.edu/datasets.html`

[9] The system uses hyperthreading hence it has 16 physical cores

[10] `https://github.com/apache/incubator-sedona/blob/master/examples/sql/src/main/scala/ScalaExample.scala`

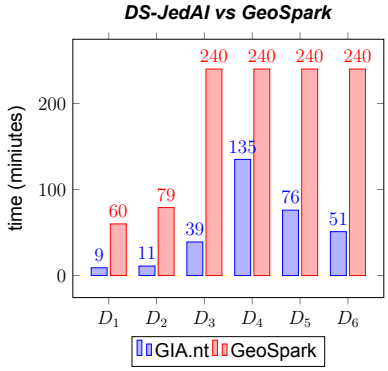


Figure 10: Comparison between GIA.nt and GeoSpark using only the `intersects` relation

Figure 10 shows the performance of both systems for the six dataset pairs. We timed out the execution of GeoSpark after 4 hours of execution. In all cases, GIA.nt significantly outperforms GeoSpark, as even in the case with the most complicated dataset (i.e., D_4), GIA.nt was able to complete the calculation in approximately 2 hours, while GeoSpark was not able to complete it even after 4 hours. Regarding D_1 and D_2, which GeoSpark managed to complete, GIA.nt requires approximately 15% of the time needed by GeoSpark. Despite the fact that GeoSpark uses spatial partitioning, it performs all the verifications within the partitions and does not use any tiling-based filtering technique.

The performance of D_4 is an outlier and requires double time than expected. This is probably because it contains complex geometry collections instead of individual geometries, thus requiring a time-consuming verification.

5.5 Evaluation of progressive algorithms

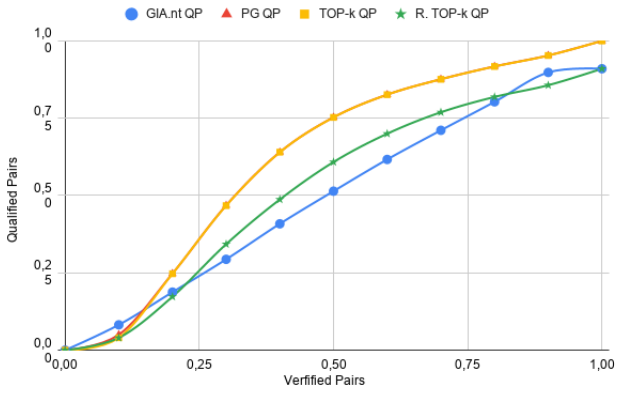
To assess the relative performance of progressive methods by the rate of producing results as more pairs are verified. We actually define *Progressive Geometry Recall (PGR)* as the rate of detecting qualifying geometry pairs and quantify it by the area under the curve that is formed by the corresponding lines in Figure 3a. The larger this area is, the earlier the interlinked pairs are detected or more relations are computed, and the more effective is the progressive method. We formalize this measure as

$$PGR = \sum_{|P|}^{i=1} \frac{P_Q^i}{P_Q^{BU}}$$

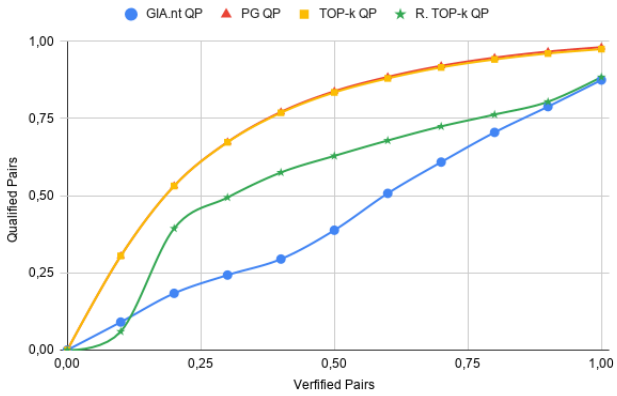
where $P \subseteq S \times T$ is the set of distinct geometry pairs that pass the Filtering step, $|P|$ is its size (i.e., the total number of candidate pairs), $P_Q^{BU} \subseteq P$ is the set of qualifying geometry pairs within the given budget BU, and P_Q^i is the total number of qualifying geometry pairs that have already been detected when processing the i^{th} candidate pair. PGR takes values in [0, 1], with higher values indicating higher effectiveness.

To assess the effectiveness, we also use *Recall* and *Precision*. To better evaluate the progressive algorithms, we have adjusted these metrics in terms of the given budget (BU) and the number of qualifying pairs within the budget (P_Q). Therefore, we calculate them as:

$$Recall = \frac{P_Q^D}{P_Q^{BU}} \qquad Precision = \frac{P_Q^D}{BU}$$



(a) PGR of all approaches over D_1 with $BU = 10M$



(b) PGR of all approaches over D_2 with $BU = 10M$

Figure 11: Evaluation of all progressive methods

where P_Q^D stands for the number of detected qualifying pairs within the budget. For all measures, higher values indicate higher effectiveness. However, the values of Precision and PGR are defined by the size of the given budget and the number of qualifying pairs within the budget, hence the optimal values are not always 1 but defined by the characteristics of the problem. Consequently, for each case, we calculate and show the optimal values.

As baseline methods we consider the *Optimal* approach, which verifies all qualifying pairs before the non-qualifying candidate ones, and the batch algorithm GIA.nt which specifies a deterministic processing order for the input data. In GIA.nt the sequence of the verifications is random, depending on the order of the geometries inside the datasets.

Since DS-JedAI works in parallel, each partition forms its own priority-queue consisting of its top geometry pairs. Normally, the partitions perform the verifications in parallel and discover the relations, however, this way we cannot compute the PGR as the verifications are not performed sequentially. So, in order to compute these metrics, we collect the top geometry pairs of each partition and we order them based on their weights. Then we perform the verifications sequentially in descending order and compute PGR as well as the other metrics. This procedure is executed only when we want to get the evaluation metrics.

Tables 5 and 6 reports the performance of all progressive metrics, for all three weighting schemes and for all the datasets, using two input budgets: 5 and 10 million verifications respectively. In almost all cases, the progressive methods perform better than the random prioritization of GIA.nt. Interestingly, there are certain cases, like in D_5, where the progressive methods perform twice as good as GIA.nt, discovering most of the qualifying pairs. Notable is the performance of Reciprocal Geometry Top-k with $Pearson's\ \chi^2$ weighting scheme, where using BU=10M manages to achieve Precision up to **0.935** in the dataset D_5, with almost all the verification performed, to be qualifying pairs.

From the experiments, we can observe that the JS and the χ^2 weighting schemes perform better than CF, except for the case of $D2$. Furthermore, Progressive GIA.nt and Geometries Top-k, tends perform similarly and producing the same results. This is also visible in Figure 11, as their curves overlap. Reciprocal Geometries Top-k performs similarly with the other methods, but with some fluctuations. However, in Reciprocal Geometries Top-k fewer geometry pairs are being verified as there are cases where there are just a few common pairs between the top-k of source and target. In order to overcome

this issue, we used bigger budgets for each partition, but in cases like the D_2 of Table 6 did not work. In the end, we conclude that there is not a single methods that stands out from the rest, and the best choice depends on the geometries and their characteristics.

Regarding the execution time, we see that the overall time is dictated by the verification of the pairs, and the Filtering and Scheduling steps have little impact to it. This is notable by the fact that the overall time of GIA.nt, which does not apply any prioritization, is similar with the overall time of the progressive methods.

Figure 11 shows the curves of all progressive methods with regards to the discovered qualifying pairs per verification for D_1 and D_2 with BU=10M. We can observe that the progressive methods have faster acceleration than GIA.nt and only Progressive GIA.nt and Geometries Top-k manages to find all the qualifying pairs within the budget.

However, we notice that the progressive methods are not so effective with the datasets D_3, D_4 and D_6 . In our analysis, we observed that most of the pairs are assigned with the same weights and hence the prioritization is useless. This is because most of the geometries of those datasets are very small, while also contain a few very big geometries. Therefore, most of the geometries are assigned to just a few tiles and hence the number of the shared overlapping tiles are the same, for most of the pairs. In order to address such issues, we are working on weighting schemes that considers other properties and not just the number of common tiles, like the area of the intersection of the MBRs.

Table 5: Evaluation of progressive methods using all weighting schemes and 5M budget

		Optimal	GIA.nt	Progressive GIA.nt			TOP-K			RECIPROCAL TOP-K		
				CF	JS	χ^2	CF	JS	χ^2	CF	JS	χ^2
D_1	Verifications	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	4,400,792	4,401,290	3,210,536
	P_Q^D	2,401,396	1,948,479	1,865,138	2,281,947	2,278,590	1,871,386	2,281,349	2,277,729	2,187,065	2,187,085	2,186,925
	Recall	1	0.810	0.776	0.950	0.968	0.779	0.950	0.948	0.910	0.910	0.910
	Precision	0.48	0.380	0.373	0.456	0.455	0.374	0.456	0.456	0.496	0.496	0.496
	PGR	0.76	0.407	0.339	**0.650**	**0.647**	0.340	**0.650**	**0.647**	0.404	**0.611**	**0.609**
	time (s)	-	507	498	498	498	372	372	372	314	314	314
D_2	Verifications	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000
	P_Q^D	154,386	154,386	118,046	129,423	124,781	117,829	128,832	123,856	127,620	132,091	128,335
	Recall	1	0.421	0.764	0.838	0.808	0.763	0.834	0.802	0.826	0.855	0.831
	Precision	0.04	0.013	0.023	0.025	0.024	0.023	0.025	0.024	0.025	0.026	0.025
	PGR	0.98	0.212	**0.578**	**0.545**	0.505	**0.578**	**0.543**	0.503	**0.606**	**0.555**	0.518
	time (s)	-	479	408	408	408	503	503	503	347	347	347
D_3	Verifications	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000
	P_Q^D	3,805,385	2,108,430	927,778	1,742,838	1,750,085	924,371	1,742,585	1,749,670	1,321,132	1,749,064	1,754,066
	Recall	1	0.554	0.244	0.458	0.460	0.243	0.458	0.460	0.347	0.460	0.461
	Precision	0.76	0.422	0.186	0.349	0.350	0.185	0.349	0.350	0.264	0.350	0.351
	PGR	0.619	0.309	0.122	0.267	0.268	0.122	0.267	0.268	0.164	0.268	0.269
	time (s)	-	987	693	693	693	672	672	672	578	578	578
D_4	Verifications	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000
	P_Q^D	5,000,000	1,021,233	1,808,267	794,069	772,515	1,807,726	807,358	771,324	1,826,482	797,737	775,992
	Recall	1	0.204	0.362	0.159	0.155	0.362	0.161	0.154	0.365	0.160	0.155
	Precision	1	0.204	0.362	0.159	0.155	0.362	0.161	0.154	0.365	0.160	0.155
	PGR	0.5	0.112	0.192	0.083	0.082	0.192	0.083	0.081	0.195	0.083	0.082
	time (s)	-	1,667	1,517	1,517	1,517	1,409	1,409	1,409	1,692	1,692	1,692
D_5	Verifications	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000
	P_Q^D	5,000,000	2,247,847	976,937	4,519,008	4,591,464	1,302,743	4,549,633	4,621,306	3,316,463	4,606,489	4,720,124
	Recall	1	0.450	0.195	0.904	0.918	0.261	0.910	0.924	0.663	0.921	0.944
	Precision	1	0.450	0.195	0.904	0.918	0.261	0.910	0.924	0.663	0.921	0.944
	PGR	0.5	0.219	0.094	**0.452**	**0.459**	0.117	**0.455**	**0.462**	0.289	**0.461**	**0.472**
	time (s)	-	894	881	881	881	883	883	883	881	881	881
D_6	Verifications	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000	5,000,000
	P_Q^D	1,037,153	74,220	50,716	149,541	146,045	49,670	139,934	139,955	56,049	171,076	172,937
	Recall	1	0.072	0.049	0.144	0.141	0.048	0.135	0.135	0.054	0.165	0.167
	Precision	0.207	0.015	0.010	0.030	0.029	0.010	0.028	0.028	0.011	0.034	0.035
	PGR	0.89	0.039	0.025	0.065	0.064	0.025	0.058	0.060	0.027	0.080	0.081
	time (s)	-	2,802	2,714	2,714	2,714	2,507	2,507	2,507	2,495	2,495	2,495

Table 6: Evaluation of all progressive methods using all weighting schemes and 10M budget

		Optimal	GIA.nt	Progressive GIA.nt			TOP-K			RECIPROCAL TOP-K		
				CF	JS	χ^2	CF	JS	χ^2	CF	JS	χ^2
D_1	Verifications	6,309,676	6,309,676	6,309,676	6,309,676	6,309,676	6,309,362	6,306,739	6,304,539	4,400,792	4,401,290	4,402,420
	P_Q^D	2,401,396	2,401,396	2,401,396	2,401,396	2,401,396	2,401,396	2,401,396	2,401,396	2,187,065	2,187,085	2,186,925
	Recall	1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.910	0.910	0.910
	Precision	0.38	0.380	0.380	0.380	0.38	0.38	0.38	0.380	0.496	0.496	0.496
	PGR	0.808	0.500	0.450	**0.718**	**0.716**	0.450	**0.718**	**0.715**	0.401	**0.611**	**0.609**
	time (s)	-	533	542	542	542	519	519	519	314	314	314
D_2	Verifications	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	5,830,517	5,833,331	5,839,031
	P_Q^D	154,386	135,048	144,972	154,386	150,546	146,285	150,489	148,408	136,501	135,216	132,143
	Recall	1	0.874	0.939	0.981	0.975	0.947	0.974	0.961	0.884	0.876	0.856
	Precision	0.015	0.013	0.014	0.015	0.015	0.014	0.015	0.014	0.023	0.023	0.023
	PGR	0.992	0.425	**0.714**	**0.735**	**0.708**	**0.718**	**0.732**	**0.702**	**0.643**	**0.600**	0.565
	time (s)	-	629	601	601	601	611	611	611	416	416	416
D_3	Verifications	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000
	P_Q^D	3,805,385	3,066,368	1,943,127	3,185,780	3,180,786	1,926,925	3,183,831	3,178,875	2,736,876	2,996,605	2,999,240
	Recall	1	0.806	0.511	0.837	0.836	0.506	0.837	0.835	0.719	0.787	0.788
	Precision	0.308	0.307	0.194	0.319	0.318	0.193	0.318	0.318	0.274	0.300	0.300
	PGR	0.801	0.514	0.248	**0.465**	**0.466**	0.246	**0.465**	**0.466**	0.354	**0.460**	**0.462**
	time (s)	-	1,448	1,044	1,044	1,044	1,013	1,013	1,013	746	746	746
D_4	Verifications	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000
	P_Q^D	10,000,000	2,015,836	3,234,364	1,610,101	1,632,532	3,236,344	1,610,379	1,634,468	3,168,507	1,626,189	1,645,141
	Recall	1	0.202	0.323	0.161	0.163	0.324	0.161	0.163	0.317	0.163	0.165
	Precision	1	0.202	0.323	0.161	0.163	0.324	0.161	0.163	0.317	0.163	0.165
	PGR	0.5	0.107	0.175	0.081	0.080	0.175	0.081	0.080	0.176	0.081	0.081
	time (s)	-	1,912	1,561	1,561	1,561	1,566	1,566	1,566	1,398	1,398	1,398
D_5	Verifications	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000
	P_Q^D	10,000,000	4,190,668	2,182,571	8,971,887	9,101,636	2,702,275	9,028,098	9,167,172	7,496,133	9,143,726	9,351,004
	Recall	1	0.419	0.218	0.897	0.910	0.270	0.903	0.917	0.750	0.914	0.935
	Precision	1	0.419	0.218	0.897	0.910	0.270	0.903	0.917	0.750	0.914	0.935
	PGR	0.5	0.216	0.105	**0.448**	**0.455**	0.129	**0.451**	**0.458**	**0.338**	**0.457**	**0.466**
	time (s)	-	956	891	891	891	901	901	901	905	905	905
D_6	Verifications	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000
	P_Q^D	1,037,153	119,082	79,133	235,026	236,252	78,383	221,462	221,705	158,241	296,186	295,982
	Recall	1	0.115	0.076	0.227	0.228	0.076	0.214	0.214	0.153	0.286	0.285
	Precision	0.103	0.012	0.008	0.024	0.024	0.008	0.022	0.022	0.016	0.030	0.030
	PGR	0.948	0.068	0.040	0.110	0.112	0.039	0.105	0.107	0.066	0.153	0.154
	time (s)	-	2,821	2,465	2,465	2,465	2,524	2,524	2,524	2,515	2,515	2,515

5.6 Summary

In this chapter we introduced the algorithms GIA.nt, Progressive GIA.nt, Geometry Top-k and Reciprocal Geometry Top-k for batch and progressive Holistic Geospatial Interlinking. Furthermore we show how we have implemented and parallelized them in the system DS-JedAI that works on top of Apache Spark. In the end we present detailed evaluation of the algorithms and we show that progressive algorithms are able to discover most of the relations by performing fewer verifications.

6. POLAR USE-CASE[1]

In this chapter we present the polar use case, in which we use both tools in order to interlink satellite images with in-situ ice observations, with the view to provide better ice monitoring capabilities. The end result is useful for building training sets with satellite images associated with high quality ground observations and essential for the construction of a robust automatically derived mapping product that could be updated frequently. Most importantly, by expressing the end result as RDF statements, we can leverage a wealth of Semantic Web tools for performing analytics, reasoning as well as visualization. A crucial aspect in this process is the time efficiency, as the large volume of data calls for scalable techniques.

6.1 Ice Monitoring

The safety of ship navigation in the Arctic relies on the continuous task of monitoring sea ice and icebergs. This task has recently become more critical, due to changes in ice conditions in the Arctic driven by climate change that have made the ice more mobile and driven an increase in ship traffic. To address it, automatic techniques are now used to combine in-situ observational data with satellite images. Such in-situ data can be used to validate and improve the interpretation of satellite images, and subsequently to improve routine ice chart products and assist in building training sets for the future development of machine learning algorithms.

In more detail, the Ice Watch project[2] of the Norwegian Meteorological Institute collects data from ships performing visual sea ice observations while navigating the Arctic. These in-situ observational data record the time, point locations, and other important properties of sea ice.

In this work, we work with a set of satellite images provided by the EU Copernicus Programme, that cover the Arctic and coincide spatially and temporally with the in-situ observations. Our goal is to interlink these two data sources, so as to identify in-situ observations that match closely in time and space to satellite images.

This process takes as input three datasets:

- a set containing observation points
- a set containing ice observation data
- a set containing information about the satellite images

The observation point set contains the time and the location of each observation. The ice observation set contains detailed information about the ice (such as ice thickness, snow thickness, floe size, etc), which were recorded by the observer, along with an observation id that links back to the observation. The ice observation set contains more than 20,000 observations, however, since it is only updated when a cruise has been out in the Arctic

[1] The results of this chapter appear in the paper *"Ice Monitoring With ExtremeEarth"*, in proceedings of the Large Scale RDF Analytics (LASCAR) workshop of ESWC 2020

[2] `https://icewatch.met.no/`

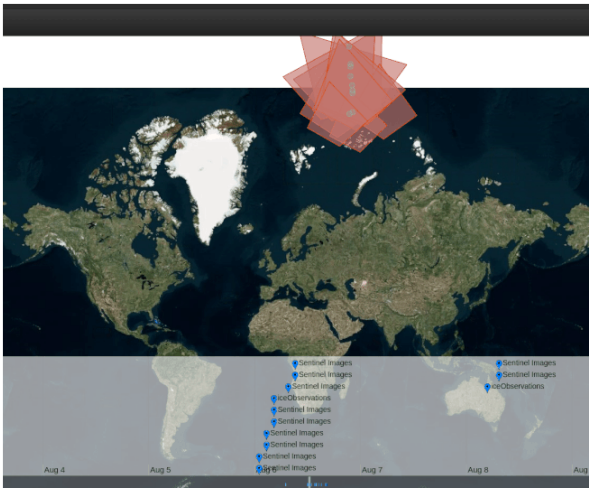


Figure 12: The results as they are presented to the user.

and uploaded its data, there are large gaps with no observations. These sets are provided by the Ice Watch project.

The set of satellite images contains information about images captured by Sentinel-1[3] and is provided by the Copernicus project. This information includes the location of the image, the satellite acquisition date and time, the coverage (geographic extents of the images), as well as some other useful properties. Note that the data source of satellite images is updated on a daily basis and contains more than 200K images.

6.2 Approach

Our approach starts by transforming the datasets into RDF graph using GeoTriples-Spark. In the mapping file of GeoTriples-Spark, we define a specialized ontlogy [4] we have developed based on the Sea Ice GeoReferenced Information and Data - SIGRID-3 document [23], which describes a set of standards to code, exchange, and archive digital ice charts. The generated triples are stored in the N-Triples format.

Then we use DS-JedAI to discover the relations between the in-situ observations and satellite images. In more details we wanted to find which satellite images captured the ice observation points at the same day that the observations occurred. Consequently, the temporal dimension has a deterministic role as we are examining ice thickness which can alter from days to days. Therefore, we focus on the `contains` relation and we also use temporal filtering to ensure that the interlinked geometries are not over 24 hours apart. This way, we link the images to the observation points that are spatially contained in the coverage of the image and also their timestamp is within a range of a day from the time that the image was shot.

We store the produced RDF graphs and the discovered relations in the spatiotemporal RDF store *Strabon* [26] a state-of-the-art open-source spatiotemporal triplestore that efficiently executes GeoSPARQL and stSPARQL queries. Finally we visualize our results via *Sextant* [30], which constitutes a web-based application for exploring, interacting, and visualizing time-evolving linked geospatial data. The results are shown in Figure 12 and

[3] `https://www.esa.int/Applications/Observing_the_Earth/Copernicus/Sentinel-1`
[4] `http://pyravlos-vm5.di.uoa.gr/polarUC.svg`

you can see the whole visualization as a GIF in [1], which exactly how it was presented to the user.

7. CONCLUSION

In this work we focus on the transformation of big geospatial data into RDF graphs, by presenting GeoTriples-Spark, a new version of GeoTriples able to perform transformation on scale. Furthermore, we also focus on the field of Geospatial Interlinking by presenting novel algorithms for batch and progressive Holistic Geospatial Interlinking. All these algorithms are implemented in the system DS-JedAI that runs on top of Apache Spark and it is able to perform Geospatial Interlinking at scale. We also present detailed evaluation of both systems and algorithms and we show that they can operate on big geospatial data.

As for future work for GeoTriples-Spark, we plan to extend it in order to be able to transform data from other geospatial sources like big KML and GML documents, and from systems that are build on top of Hadoop, like Apache Hive[1] and Apache Accumulo[2]. Moreover, we plan to extend both GeoTriples and GeoTriples-Spark to support the *GeoSPARQL+* [19] vocabulary, for handling raster formats of geospatial data.

Regarding Geospatial Interlinking, we examine new weighting schemes and new scheduling approaches based on the frequencies of the pairs and the area of their MBRs' intersection. Additionally, we try to reduce the cost of verification by using different libraries like the *s2geometry*[3], or by developing our own algorithm able to determine the relations between a pair of geometries by considering the least possible points. Last but not least, we work on the development of meta-progressive algorithms which will improve scheduling by producing a better order of geometry pairs, i.e., fewer false positives. We believe we can achieve this by combining the ranking lists of N weighting schemes into a single one.

Both systems (i.e. GeoTriples-Spark, DS-JedAI) were developed as part of the ExtremeEarth project[4] , which focuses on Artificial Intelligence and Big Data technologies that scale to the petabytes of big Copernicus data. ExtremeEarth applies these technologies in two of the thematic exploitation platforms of the European Space Agency: one dedicated to Food Security and one dedicated to the Polar regions. Its goal is to develop techniques and software that will enable the extraction of information and knowledge from big Copernicus data using deep learning techniques and extreme geospatial analytics, making this information and knowledge available as linked data.

[1] https://hive.apache.org/
[2] https://accumulo.apache.org/
[3] https://s2geometry.io/
[4] http://earthanalytics.eu/index.html

ACRONYMS

OSM	OpenStreetMap
EO	Earth Observation
LOD	Linked Open Data
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
KG	Knowledge Graph
WKT	Well-Known Text
OGC	Open Geospatial Consortium
DE-9IM	Dimensionally Extended 9-Intersection Model
RDD	Resilient Distributed Dataset
IM	Intersection Matrix
RML	RDF Mapping Language
R2RML	RDB to RDF Mapping Language
MBR	Minimum Bounding Rectangle
HDFS	Hadoop Distributed File System
GIA.nt	Geospatial Interlinking At large
RF	Reference Point
PQ	Priority Queue
CF	Co-occurrence Frequency
JS	Jaccard Similarity
DS-JedAI	Distributed - Spatial Java gEneric DAta Integration

REFERENCES

[1] Extremeearth polar use-casein-situ ice observations interlinked with satellite images, 2020.

[2] J. Abdul, M. Alkathiri, and M. B. Potdar. Geospatial hadoop (gs-hadoop) an efficient mapreduce based engine for distributed processing of shapefiles. In *2nd International Conference on Advances in Computing, Communication, Automation (ICACCA)*, 2016.

[3] Abdullah Fathi Ahmed, Mohamed Ahmed Sherif, and Axel-Cyrille Ngonga Ngomo. RADON2 - a buffered-intersection matrix computing approach to accelerate link discovery over geo-spatial RDF knowledge bases: OAEI2018 results. In Pavel Shvaiko, Jérôme Euzenat, Ernesto Jiménez-Ruiz, Michelle Cheatham, and Oktie Hassanzadeh, editors, *Proceedings of the 13th International Workshop on Ontology Matching co-located with the 17th International Semantic Web Conference, OM@ISWC 2018, Monterey, CA, USA, October 8, 2018*, volume 2288 of *CEUR Workshop Proceedings*, pages 197–204. CEUR-WS.org, 2018.

[4] Sören Auer, Jens Lehmann, and Sebastian Hellmann. Linkedgeodata: Adding a spatial dimension to the web of data. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, volume 5823 of *Lecture Notes in Computer Science*, pages 731–746. Springer, 2009.

[5] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[6] Edward P. F. Chan and Jimmy N. H. Ng. A general and efficient implementation of geometric operators and predicates. In Michel Scholl and Agnès Voisard, editors, *Advances in Spatial Databases, 5th International Symposium, SSD'97, Berlin, Germany, July 15-18, 1997, Proceedings*, volume 1262 of *Lecture Notes in Computer Science*, pages 69–93. Springer, 1997.

[7] Kevin Chentout and Alejandro A. Vaisman. Adding spatial support to R2RML mappings. In *On the Move to Meaningful Internet Systems: OTM 2013 Workshops - 2013*, volume 8186, pages 398–407.

[8] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. A small set of formal topological relationships suitable for end-user interaction. In David J. Abel and Beng Chin Ooi, editors, *Advances in Spatial Databases, Third International Symposium, SSD'93, Singapore, June 23-25, 1993, Proceedings*, volume 692 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 1993.

[9] Eliseo Clementini, Jayant Sharma, and Max J. Egenhofer. Modelling topological spatial relations: Strategies for query processing. *Comput. Graph.*, 18(6):815–822, 1994.

[10] Alexander de León, Victor Saquicela, Luis M. Vilches, Boris Villazón-Terrazas, Freddy Priyatna, and Oscar Corcho. Geographical linked data: A Spanish use case. In *Proceedings of the 6th International Conference on Semantic Systems*. ACM, 2010.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[12] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A generic language for integrated RDF mappings of heterogeneous data. In *Proceedings of the Workshop on Linked Data on the Web, International World Wide Web Conference (WWW 2014)*, volume 1184, 2014.

[13] Jens-Peter Dittrich and Bernhard Seeger. Data redundancy and duplicate detection in spatial join processing. In David B. Lomet and Gerhard Weikum, editors, *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, pages 535–546. IEEE Computer Society, 2000.

[14] Max J. Egenhofer and Robert D. Franzosa. Point set topological relations. *Int. J. Geogr. Inf. Sci.*, 5(2):161–174, 1991.

[15] Ahmed Eldawy and Mohamed F. Mokbel. SpatialHadoop: A MapReduce Framework for Spatial Data. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1352–1363, 2015.

[16] John R. Herring. OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option. Open Geospatial Consortium standard, 2010. `http://portal.opengeospatial.org/files/?artifact_id=25354`.

[17] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. YAGO2: exploring and querying world knowledge in time, space, context, and many languages. In *WWW*, 2011.

[18] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Intell.*, 194, 2013.

[19] Timo Homburg, Steffen Staab, and Daniel Janke. Geosparql+: Syntax, semantics and system for integrated querying of graph, raster and vector data. In Jeff Z. Pan, Valentina A. M. Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part I*, volume 12506 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2020.

[20] Theofilos Ioannidis, George Garbis, Kostis Kyzirakos, Konstantina Bereta, and Manolis Koubarakis. Evaluating geospatial RDF stores using the benchmark geographica 2. *CoRR*, abs/1906.01933, 2019.

[21] Mahmoud Ismail, Ermias Gebremeskel, Theofilos Kakantousis, Gautier Berthou, and Jim Dowling. Hopsworks: Improving user experience and development on hadoop with scalable, strongly consistent metadata. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017*, pages 2525–2528.

[22] Mahmoud Ismail, Salman Niazi, Mikael Ronström, Seif Haridi, and Jim Dowling. Scaling HDFS to more than 1 million operations per second with hopsfs. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, pages 683–688, 2017.

[23] JCOMM Expert Team on Sea Ice. Sigrid-3: A vector archive format for sea ice georeferenced information and data. 2014.

[24] Wajdi Al Jedaibi and Sufian Khamis. Towards measuring the project management process during large scale software system implementation phase. *ISC Int. J. Inf. Secur.*, 11(3):161–172, 2019.

[25] Nikolaos Karalis, Georgios M. Mandilaras, and Manolis Koubarakis. Extending the YAGO2 knowledge graph with precise geospatial knowledge. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, volume 11779 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2019.

[26] Kostis Kyzirakos, Manos Karpathiotakis, Konstantina Bereta, George Garbis, Charalampos Nikolaou, Panayiotis Smeros, Stella Giannakopoulou, Kallirroi Dogani, and Manolis Koubarakis. The spatiotemporal RDF store strabon. In *Advances in Spatial and Temporal Databases - 13th International Symposium (SSTD)*, volume 8098, pages 496–500, 2013.

[27] Kostis Kyzirakos, Dimitrianos Savva, Ioannis Vlachopoulos, Alexandros Vasileiou, Nikolaos Karalis, Manolis Koubarakis, and Stefan Manegold. Geotriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings. *Journal of Web Semantics*, 52-53, 2018.

[28] Jens Lehmann, Gezim Sejdiu, Lorenz Bühmann, Patrick Westphal, Claus Stadler, Ivan Ermilov, Simon Bin, Nilesh Chakraborty, Muhammad Saleem, Axel-Cyrille Ngomo Ngonga, and Hajira Jabeen. Distributed semantic analytics using the sansa stack. In *Proceedings of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*, pages 147–155. Springer, 2017.

[29] Salman Niazi, Mahmoud Ismail, Seif Haridi, Jim Dowling, Steffen Grohsschmiedt, and Mikael Ronström. Hopsfs: Scaling hierarchical file system metadata using newsql databases. In *15th USENIX Conference on File and Storage Technologies, FAST 2017*, pages 89–104.

[30] Charalampos Nikolaou, Kallirroi Dogani, Konstantina Bereta, George Garbis, Manos Karpathiotakis, Kostis Kyzirakos, and Manolis Koubarakis. Sextant: Visualizing time-evolving linked geospatial data. *J. Web Semant.*, 35:35–52, 2015.

[31] George Papadakis, Georgios M. Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. Three-dimensional entity resolution with jedai. *Inf. Syst.*, 93:101565, 2020.

[32] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. Jedai: The force behind entity resolution. In Eva Blomqvist, Katja Hose, Heiko Paulheim, Agnieszka Lawrynowicz, Fabio Ciravegna, and Olaf Hartig, editors, *The Semantic Web: ESWC 2017 Satellite Events - ESWC 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers*, volume 10577 of *Lecture Notes in Computer Science*, pages 161–166. Springer, 2017.

[33] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. Progressive duplicate detection. *IEEE Trans. Knowl. Data Eng.*, 27(5):1316–1329, 2015.

[34] Kostas Patroumpas, Michalis Alexakis, Giorgos Giannopoulos, and Spiros Athanasiou. Triplegeo: an ETL tool for transforming geospatial data into RDF triples. In *Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference (EDBT/ICDT 2014)*, volume 1133, pages 275–278, 2014.

[35] Kostas Patroumpas, Dimitrios Skoutas, Georgios M. Mandilaras, Giorgos Giannopoulos, and Spiros Athanasiou. Exposing points of interest as linked geospatial data. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases, SSTD 2019*, pages 21–30.

[36] Tzanina Saveta, Irini Fundulaki, Giorgos Flouris, and Axel-Cyrille Ngonga Ngomo. Spgen : A benchmark generator for spatial link discovery tools. In Denny Vrandecic, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, volume 11136 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 2018.

[37] Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. Radon - rapid discovery of topological relations. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 175–181. AAAI Press, 2017.

[38] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In Mohammed G. Khatib, Xubin He, and Michael Factor, editors, *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010*, pages 1–10. IEEE Computer Society, 2010.

[39] Giovanni Simonini, Sonia Bergamaschi, and H. V. Jagadish. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *Proc. VLDB Endow.*, 9(12):1173–1184, 2016.

[40] Panayiotis Smeros and Manolis Koubarakis. Discovering spatial and temporal links among RDF data. In Sören Auer, Tim Berners-Lee, Christian Bizer, and Tom Heath, editors, *Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, co-located with 25th International World Wide Web Conference (WWW 2016)*, volume 1593 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[41] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. Parallel in-memory evaluation of spatial joins. In Farnoush Banaei Kashani, Goce Trajcevski, Ralf Hartmut Güting, Lars Kulik, and Shawn D. Newsam, editors, *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2019, Chicago, IL, USA, November 5-8, 2019*, pages 516–519. ACM, 2019.

[42] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk - A link discovery framework for the web of data. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Kingsley Idehen, editors, *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009*, volume 538 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[43] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. Pay-as-you-go entity resolution. *IEEE Trans. Knowl. Data Eng.*, 25(5):1111–1124, 2013.

[44] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: a cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information System*, pages 70:1–70:4, 2015.

[45] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, pages 15–28, 2012.

[46] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.