**NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS**

**SCHOOL OF SCIENCE**
**DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION**

**BSc THESIS**

# Spatial and Temporal information in the Semantic Web

**George E. Mandilaras**

**Supervisor:** **Koubarakis Manolis,** Professor

**ATHENS**

**MARCH 2019**

**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Χωρική και Χρονική πληροφορία στον Σημασιολογικό Ιστό

**Γεώργιος Ε. Μανδηλαράς**

**Επιβλέπων:** **Κουμπαράκης Μανόλης,** Καθηγητής

**ΑΘΗΝΑ**

**ΜΑΡΤΙΟΣ 2019**

**BSc THESIS**

# Spatial and Temporal information in the Semantic Web

**George E. Mandilaras**
**S.N.:** 1115201200097

**SUPERVISOR:** **Manolis Koubarakis,** Professor

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Χωρική και Χρονική πληροφορία στον Σημασιολογικό Ιστό

**Γεώργιος Ε. Μανδηλαράς**
**Α.Μ.:** 1115201200097

**ΕΠΙΒΛΕΠΩΝ:** **Κουμπαράκης Μανόλης,** Καθηγητής

# ABSTRACT

The main purpose of this thesis is the enhancement of the Semantic Web with geospatial and temporal information by extending the YAGO knowledge graph with such information. It is composed of three parts. The first part refers to the conversion of OpenStreetMap data into RDF triples. OpenStreetMap is a collaborative project of a free editable map of the whole world. It contains a lot of useful information which is prerequisite for several applications and therefore its transformation into RDF triples is of significant importance. The second part concerns the conversion of big geospatial data in RDF triples. In this implementation, an ETL utility is extended to work on top of Spark which enables the parallelization of the conversion which results in the reduction of the execution cost. The third part is about the extension of the YAGO knowledge base with temporal and geospatial information the former administrative division of Greece.



**SUBJECT AREA**: Semantic Web

**KEYWORDS**: YAGO, Temporal Information, Geospatial information, Spark, RDF

# ΠΕΡΙΛΗΨΗ

Ο κύριος σκοπός της πτυχιακής εργασίας είναι η ενίσχυση του Σημασιολογικού Ιστού με χρονική και χωρική πληροφορία επεκτείνοντας τον γράφο γνώσης YAGO με τέτοια πληροφορία. Η εργασία αποτελείται από τρία μέρη. Το πρώτο μέρος αναφέρεται στην μετατροπή των δεδομένων του OpenStreetMap σε RDF τριπλέτες. Το OpenStreetMap είναι ένας χάρτης με ελεύθερη άδεια ο οποίος αναπτύσσεται από μια κοινότητα εθελοντών και περιέχει πληροφορίες για όλο τον κόσμο. Τα δεδομένα του είναι ιδιαίτερα χρήσιμα και απαραίτητα για πολλές εφαρμογές, και για αυτό η παροχή τους σε μορφή RDF είναι ιδιαίτερα σημαντική. Το δεύτερο σκέλος της πτυχιακής αφορά αφορά την μετατροπή μεγάλων χωρικών δεδομένων σε RDF τριπλέτες. Σε αυτήν την υλοποίηση επεκτείνουμε ένα ETL εργαλείο με την τεχνολογία Spark η οποία μας επιτρέπει να παραλλιλοποιήσουμε την μετατροπή των δεδομένων σε RDF με αποτέλεσμα να μειωθεί σημαντικά ο χρόνος εκτέλεσης. Το τρίτο κομμάτι έχει να κάνει με την επέκταση της γνωσιακής βάσης YAGO με χρονική και χωρική πληροφορία σχετικά με την πρώην διοικητική διαίρεση της Ελλάδας.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ**: Σημασιολογικός Ιστός

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ**: YAGO, Χρονική Πληροφορία, Χωρική Πληροφορία, Spark, RDF

# ACKNOWLEDGEMENTS

Firstly I would like to thank Prof. Manolis Koubarakis for giving me the opportunity to work on this subject and for his guidance. Furthermore, I would like to thank Prof. Kostas Patroumpas and Nikos Karalis, for constantly advising me during the whole procedure. Last but not least, I would like to thank Spiros Athanasiou and the ATHENA research lab for providing me with the necessary resources and data, in order to make it possible.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1. INTRODUCTION

## 1.1 The Semantic Web

The Semantic Web [1] 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 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) [2]:

- Resource Description Framework(RDF) [3] – a simple language for describing objects and their relations in a graph;
- SPARQL Protocol and RDF Query Language (SPARQL) [4] – a protocol and query language for RDF data;
- Uniform Resource Identifier(URI) – a string of characters designed for unambiguous identification of resources and extensibility via the URI scheme.

## 1.2 Resource Description Framework

The Resource Description Framework (RDF) is a family of World Wide Web Consortium (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.

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 (see Figure 1). The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. In plain English, an RDF statement states facts, relationships, and data by linking resources of a different kind. With the help of an RDF statement, just about anything can be expressed by a uniform structure, consisting of three linked data pieces. For example the phrase "Paule lives in San Francisco" can be modelled as follows: Paul is the subject, lives in is the predicate and San Francisco is the object of the triple.


**Figure 1: RDF triple**

RDF is the key part of the Semantic Web since it is built around resources with URIs, which can be entities that exist within the web. Using HTTP URIs in RDF statements makes information more structured and more meaningful to software programs allowing them to interact with the web the same way as people do. Furthermore, it will facilitate the process of exchanging data across the web without the intervening of human action. This movement is also known as Linked Open Data (LOD).

RDF is a very important part of the Semantic web and its advantages are listed below.

### Data Interoperability

It is in data exchange and interoperability that RDF really shines. RDF can capture and convey the metadata or information in unstructured (text), semi-structured (HTML documents) or structured sources (standard databases). This makes RDF almost a "universal solvent" for data representation. Once in a common RDF representation, it is easy to incorporate new datasets or new attributes. It is also easy to aggregate disparate data sources as if they came from a single source. This enables meaningful composition of data from different applications regardless of format or serialization.

### Schema Unbound

A major advantage of RDF is that there are not schema boundaries. Schema rigidity or schema fragility is one of the major burdens of data integration. Once data relationships are set, they remain and they can not easily be changed in a conventional data management systems nor in the applications that use them. Even though relational database management systems(RDBMS) are tremendously useful and enable the addition of more data records, they are neither adaptive nor flexible. RDF has no such limitations. RDF is well suited and can provide a common framework to represent both data instances and the schema that describe them.

### Increment, Evolve, Extend, Adapt

The fluidity of RDF is another key strength. Since a basic RDF model can be processed even in the absence of more detailed information, input data and basic inferences can proceed early and logically as a simple fact basis. This means that the data or the schema can be extended. Partial representations can be incorporated as readily as complete ones, and the schema can extend and evolve as the new structure is discovered or encountered. RDF provides a data and schema representation framework that can evolve and adapt to what data exists and what structure is known. As new data with new attributes are

discovered, or as new relationships are found, these can be added to the existing model without any change to the prior existing schema. This very adaptability is what enables RDF to be viewed as data-driven design. By replacing the rigid relational data model with one based on RDF, we gain robustness, flexibility, universality and structural persistence over fragility.

### Data-driven Applications

Two fundamental tools are the RDF query language, SPARQL, and inferencing. SPARQL provides a generalized basis for driving reports and templated data displays, as well as standard querying. Utilizing the simple triple structure of RDF, SPARQL can also be used to query a dataset without knowing anything in advance about the data. This provides a very useful discovery mode. Simple inferencing can be applied to broaden and contextualize search, retrieval, and analysis. Inference tables can also be created in advance and layered over existing RDF datastores for automatic invoking of inferencing. More complicated inferencing means that RDF models can also perform as complete conceptual views of the world or knowledge bases.

### A Graph Representation

An RDF graph, which is a directed graph, consists of multiple RDF triples. A graph structure has many advantages. Graphs are modular and can be both readily combined and broken apart. From a computational standpoint, this can lend itself to parallelized information processing (and, therefore, scalability). With specific reference to RDF, it also means that graph extractions are themselves valid RDF models. Graph algorithms are a significant field of interest in mathematics and computer science. Graphs also have some unique aspects in search and pattern matching. Besides options like finding paths between two nodes, depth-first search, breadth-first search, or finding shortest paths, emerging graph and pattern-matching approaches may offer entirely new paradigms for search. Graphs also provide new approaches for visualization and navigation, useful for both seeing relationships and framing information from the local to global contexts. The interconnectedness of the graph allows data to be explored via contextual facets, which is revolutionizing data understanding.

### RDF as a Relational Model

Despite the differences in fragility and robustness, there are many logical affinities between the relational model and RDF. RDF can be modelled relationally as a single table with three columns corresponding to the subject–predicate–object triple. Conversely, a relational table can be modelled in RDF with the subject IRI derived from the primary key or a blank node; the predicate from the column identifier; and the object from the cell value. Because of these affinities, it is also possible to store RDF data models in existing relational databases. Moreover, these affinities also mean that RDF stored in this manner can also take advantage of the historical learnings around RDBMS and SQL query optimizations.

## 1.3 Knowledge Bases

There is a continuous effort in order to make data publicly available and this attempt has been assisted by the emerging of Knowledge Bases. A Knowledge Base (KB) is a computer-processable collection of knowledge, that contains entities which represent objects of the real world and facts about them. In most cases, their information is structured in order special designed softwares to be able to utilize it. Knowledge Bases can either contain encyclopedic content or information about a particular subject. Furthermore, most of them follow the RDF data model which allows them to be interlinked with other knowledge bases and datasets. This results in the construction of a graph that connects the entities of the KB, known as the knowledge graph. Moreover, many KBs contain information from different sources and from other KBs, interlinking their Knowledge Graphs and creating a network of accumulated knowledge. As a result, a vast network of interconnected KBs has been created, which is known as Linked Data Cloud [5] and it is one of the principals and integral parts of the Semantic Web.

Some of the most famous knowledge bases are the following:

**DBpedia** [6] is probably the most popular KB. Its content is collected by the structured information that exists in the Wikipedia pages like info-boxes and articles metadata. DBpedia is interlinked with other knowledge bases (e.g. YAGO) and it is part of the Linked Data Cloud. Its data is accessible and can be queried via its online SPARQL endpoints.

**Wikidata** [7] is a free collaboratively edited knowledge base hosted by the Wikimedia Foundation and its content represents entities of the real world like people, places and objects. It is the successor of Freebase which powered Google's Knowledge Graph. Wikidata provides a graphical user interface where users can edit its current information or add a new one. This triggered people interest and has led to significantly increase its size and to provide its information in multiple languages. Wikidata is also part of the Linked Data Cloud and its data is accessible and can be queried via its online SPARQL endpoints.

**YAGO** [8] is a knowledge base developed by the Max Planck Institute and It is one of the first knowledge bases, that was created from multiples sources. Its content contains more than 14 million entities and more than 447 million facts. Its first version was released in 2007 and it was created by extracting and combining knowledge from two different sources, WordNet and Wikipedia. In 2011, the second version of YAGO was released known as YAGO2 [9], where its knowledge graph was enriched with temporal and spatial facts. The spatial facts were extracted by Wikipedia pages and from a new source GeoNames. GeoNames is a gazetteer, whose data and accuracy have been studied extensively [10]. Temporal information was added mainly to entities that represent people, groups, artefacts or events. YAGO3 [11] is the latest version of YAGO and it was released in 2015. In this version was added information from Wikipedia pages with different languages.

## 1.4 Goal and Structure

The main goal of this thesis is to contribute to the YAGO knowledge base by extending it with temporal and geospatial information regarding the former administrative division of Greece. OpenStreetMap is a rich source of geospatial information. Another goal of this work is the conversion of OpenStreetMap data into RDF so that it can be used in to further extend not only YAGO but also the Semantic Web in general. This is accomplished by constructing a routine that automates the conversion of OpenStreetMap data into RDF using the ETL tool TripleGeo. Since the volume of OSM data is large, we also extending

TripleGeo to work on top of Apache Spark in order to make it capable of handling such data. The extension of YAGO's knowledge graph is part of a greater project conducted by Prof. Manolis Koubarakis and Nikos Karalis. In this project, YAGO's knowledge graph is extended with spatial information from multiple data sources [12].

The rest of this thesis is structured as follows. Chapter 2 discusses related work. Chapter 3 presents the conversion of OpenStreetMap data into RDF triples using the ETL utility TripleGeo. Chapter 4 analyzes the extension of TripleGeo to work on top of Spark in order to parallelise the process of conversion of big geospatial data. Chapter 5 describes the enrichment of the knowledge graph of YAGO with spatial and temporal information about the former Greek administrative division. Finally, chapter 6 is dedicated to the conclusion and future work.

# 2. RELATED WORK

## 2.1 Spatio-Temporal Information in KBs

All the entities of the real world have a direct connection with time and space. People are born and die in specific time points, structures are built and destroyed and events occur in a day or last a time span. Moreover, all physical objects have a location in space, for example, countries, cities, mountains, and rivers have a permanent physical location and shape on Earth. Therefore, it is important that KBs contain such information as it will enable us to perform queries that take into account the spatio-temporal dimension.

### 2.1.1 Spatio-Temporal Information in YAGO

Regarding the temporal information, YAGO uses the data type *yagoDate* in order to denote timepoints. This type follows the ISO 8601 [13] which is based on the Gregorian calendar (i.e. YYYY-MM-DD) and it provides the resolution to express not only days but also cruder time points like years and months. If we want to represent a year we use the wildcard # in order to omit the digits that refer to day and month, so we just write "YYYY-##-##". Dates are also declared as literals followed by the *xsd:date* which is the schema definition for dates, recommended by the World Wide Web Consortium (W3C).

Facts can only express time points, however, many entities come into existence into a certain point of time and cease to exist in another. Therefore, time spans are represented by two relations that together form a time interval, for instance, the predicates *wasBornOnDate* and *diedOnDate* are used in order to express the lifespan of a person. There are several predicates that YAGO uses in order to express the temporal information and they are used according to the type of the entity. The former predicates are used to entities that represent living beings. In case that the represented entity is still alive, it is associated only with its date of birth. The time span of artefacts such as buildings, books, music songs, or albums is expressed using the predicates *wasCreatedOnDate* and *wasDestroyedOnDate* (for entities that ceased to exist like buildings). Events that lasted for a period of time like wars or festivals are expressed using the predicates *startedOnDate* and *endedOnDate*, but in the cases where the event occurred in a single day, it is prefered the use of *happenedOnDate*. All of these relations are declared as sub-properties of the generic entity-time relations which are *startsExistingOnDate* and *endsExistingOnDate*. Figure 2 depicts some temporal facts of YAGO.

```
@base <http://yago-knowledge.org/resource/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema>.
<In_Mexico>                                  <wasCreatedOnDate>     "2010-##-##"^^xsd:date.
<Supernatural_Love>                          <wasCreatedOnDate>     "1984-10-##"^^xsd:date.
<55_Day_War>                                 <happenedOnDate>       "1993-##-##"^^xsd:date.
<Runciman_railway_station>                   <wasDestroyedOnDate>   "1918-##-##"^^xsd:date.
<I'm_Your_Man_(Leonard_Cohen_album)><wasCreatedOnDate>              "1987-11-##"^^xsd:date.
<Hallelujah_(Leonard_Cohen_song)>            <wasCreatedOnDate>     "1984-06-##"^^xsd:date.
<Leonard_Cohen>                              <wasBornOnDate>        "1934-09-21"^^xsd:date.
<Leonard_Cohen>                              <diedOnDate>           "2016-11-07"^^xsd:date.
```

**Figure 2: Examples of temporal facts in YAGO**

Furthermore, facts too can have a temporal dimension. The fact *BarackObama holdsPoliticalPosition PresidentOfTheUnitedStates* denotes an epoch from the time Obama was elected until either another president is elected or Obama resigns. Therefore this fact can be connected with the time span of this epoch. In order to implement this YAGO introduced in YAGO2 two new relations, *occursSince* and *occursUntil* and assigned all the facts with a unique ID. So if the above fact had the fact id #1, YAGO would express its time span like "*#1 occursSince 2009-01-20"*. For facts that are denoted by a single time point, the shorthand notation *occursOnDate* is used. Moreover, If the same fact occurs more than one time, then YAGO will contain it multiple times with different fact ids. For example, since Bob Dylan has won two Grammy awards, the fact *BobDylan hasWonPrize GrammyAward* is associated with two different fact ids and each id is assigned with a corresponding date using the relation *occursOnDate*.

In YAGO2, the knowledge graph of YAGO was extended with spatial information from Wikipedia and GeoNames. Its constructors introduced the new class *yagoGeoEntity*, which groups together all geo-entities. Each instance of *yagoGeoEntity* is directly connected to its geographical coordinates by the *hasGeoCoordinates* relation or to its longitude and latitude by the *hasLatitude* and *hasLongitude* predicates. The geographical coordinates are declared by the type *yagoGeoCoordinates* which is consisted of a pair of longitude and latitude. Therefore YAGO2 only knows about coordinates, not polygons, so even locations that have a physical extent are represented by a single geo-coordinate pair. Figure 3 depicts some spatial facts of YAGO.

```
@base <http://yago-knowledge.org/resource/> .
<Lyons,_New_York>            <hasLatitude>     "43.05972222222222"^^<degrees>.
<Lyons,_New_York>            <hasLongitude>    "-76.9925"^^<degrees>          .
<Omonoia,_Athens>            <hasLatitude>     "37.98388888888889"^^<degrees>.
<Omonoia,_Athens>            <hasLongitude>    "23.727777777777778"^^<degrees>.
<Rainbow_Bridge_(Tokyo)>     <hasLatitude>     "35.63638888888889"^^<degrees>.
<Rainbow_Bridge_(Tokyo)>     <hasLongitude>    "139.76361111111112"^^<degrees>.
<Finland>                    <hasLatitude>              "64.0"^^<degrees>.
<Finland>                    <hasLongitude>    "26.0"^^<degrees>.
```

**Figure 3: Examples of spatial facts in YAGO**

Similarly, with the temporal information, facts can also be associated with spatial information. For example, the fact that Leonard Cohen was born in 1934 happened in his city of birth, Montreal. YAGO connects facts with their spatial information, using the same strategy that uses for the connection of facts with their temporal information. It utilises facts ids and the relation *occursIn*, which holds between a (reified) fact and a geo-entity. Therefore, the statement *LeonardCohen wasBornOnDate 1934* is associated with a fact id which is then connected with the entity *Montreal* via the relation *occursIn*.

### 2.1.2 Spatio-Temporal Information in DBpedia and Wikidata

In contrary to YAGO, both DBpedia and Wikidata have a less strict schema for expressing the spatio-temporal information. They use more predicates to connect entities with their spatial and temporal information, from which some of them are dedicated to a specific range of entities.

Temporal and spatial information in DBpedia originates mostly from Wikipedia's infoboxes and from other sources like GeoNames. It uses a big variety of predicates for expressing the temporal information, from which the most used of them are the *deathDate* and *birthDate*. DBpedia's drawback is in the representation of dates. YAGO represents years

as incomplete dates so that there is a single unified way of expressing a date. DBpedia has different relations for complete dates and for years. This yields a number of relations that are semantic duplicates but are not synchronized with each other. Regarding DBpedia's geospatial information, the location of entities is assigned as a pair of geographic coordinates. Furthermore, even though DBpedia supports the connection of geo-entities with their geographic shape using the predicate *geo:geometry*, all the geometries that it contains are just points that represent entities' coordinates in a different way.

Wikidata is a free and open knowledge base that can be read and edited by both humans and machines. For expressing dates it uses the data type "*point in time*" which is very similar to *yagoDate*.This data type stores a date in Gregorian or Julian calendar, and can be used to express days, months and years. Since Wikidata is crowdsourced and can be edited freely, it currently contains 43 different predicates for expressing the temporal information. Most of them are not universal and are dedicated to entities of certain types. Regarding geospatial information, WikiData uses two different data types to express the geographic location of entities, the Geographic coordinates and the Geographic shapes. The Geographic coordinates are expressed as points that contain the pair of longitude and latitude of the entities. The Geographic shapes are stored as GeoJSON and can be visualized in a map.

The following table (Table 1) provides statistics about the temporal and spatial facts that are provided by each knowledge base. These stats were found by querying their KBs using the SPARQL query language. Regarding the temporal information, we chose to depict these relations (date of birth and date of death) because they are used the most. The numbers of date facts of YAGO and DBpedia are quite close because the main source of their temporal information is the same (i.e. Wikipedia). Furthermore, we can observe that YAGO contains excessively more geographic locations than the other KBs, however, it doesn't contain geographic shapes. Only Wikidata contains a small number of geographic shapes regarding the geometries of countries and big cities.

**Table 1: Spatio-Temporal Facts of KBs**

| **Description of the relation** | **YAGO's Total Facts** | **DBpedia's Total Facts** | **Wikidata's Total Facts** |
|---|---|---|---|
| Date of Birth | 1 661 704<br>*wasBornOnDate* | 1 740 612<br>*dbo:birthDate* | 3 507 905<br>*date of birth* |
| Date of death | 797 564<br>*diedOnDate* | 721 196<br>*dbo:deathDate* | 1 733 224<br>date of death |
| Longitude | 12 085 028<br>*hasLongitude* | 1 023 236<br>*geo:long* | 7 254 130<br>*coordinates location* |
| Latitude | 12 085 028<br>*hasLatitude* | 1 023 228<br>*geo:lat* | 7 254 130<br>*coordinates location* |

## 2.2 OpenStreetMap

OpenStreetMap[1](OSM) is a collaborative project to create a free editable map of the whole world. It was inspired by Wikipedia and as such it provides well-known wiki features such as an edit-tab and a full revision history of the edits. However, rather than editing articles, users edit geographic entities. The three fundamental ones are as follows:

- **Nodes** are the most primitive entities and represent geographic points with a latitude and longitude relative to the WGS84 reference system.
- **Ways** are entities that have a list of at least two node references associated with them. Depending on whether the first reference equals the last one, a way is called closed or open, respectively.
- **Relations** relate points, ways and potentially other relations to each other, thereby forming complex objects. Each entity participating in a relation plays a certain role in it. Multipolygons are modelled with relations.

Each of these entities has a numeric identifier (called OSM ID), a set of generic attributes, and most importantly is described using a set of key-value pairs, known as tags. An example of a relation is the administrative boundary of Germany having the OSM identifier 51477 It is comprised by more than 1000 ways, which represent certain segments of the German border; the German border with Luxembourg e.g. is composed of approx. 40-way segments. The relation currently has about 30 associated tag-value pairs, which, for example, contain the name of Germany in different languages. One of those tag-value pairs (boundary=administrative) indicates that this relation represents an administrative boundary. This information is used by the OSM map renderer to decide how this relation should be rendered on the map. Further tags are used for timezone, currency, and ISO country. The relation has also a few metadata entries (such as the timestamp of the last edit and the last editor) attached [14].

Rather than the map itself, the data generated by the project is considered its primary output. The creation and growth of OSM have been motivated by restrictions on use or availability of map information across much of the world, and the advent of inexpensive portable satellite navigation devices. Furthermore, the fact that OSM's crowdsourced data is available to the public under the Open Database License, it has significantly assisted the construction of various open-source applications like routing machines and distance calculator, but more importantly, it has motivated people to contribute to the enrichment of its content. However, the quantity of its data varies amongst regions. In countries like the United States and Germany where their people are more interested in contributing to the OSM project, it contains a larger quantity of information, in contrast with other countries like in Central Africa. In spite of this fact, OSM is a very rich database containing numerous useful geospatial information about the overall world.

### 2.2.1 LinkedGeoData

LinkedGeoData[2] [14] is an effort to add a spatial dimension to the Semantic Web by collecting information from OpenStreetMap and converting it into RDF. The RDF triples are stored in a large spatial KB that follows the Linked Data principles and it consists of more than 3 billion nodes and 300 million ways and the resulting RDF data comprises approximately 20 billion triples. The data is available according to the Linked Data

1 https://www.openstreetmap.org/
2 http://linkedgeodata.org/

principles and it is interlinked with DBpedia, GeoNames and therefore with the Linked Open Data Cloud. Furthermore, LinkedGeoData provides a SPARQL endpoint which allows queries in GeoSPARQL that enables us to perform complex geospatial calculations. Unfortunately, LinkedGeoData is no longer maintained, hence its data is not up-to-date.

### 2.2.2 GeoQA

GeoQA [15] is an attempt to offer a question answering service on top of linked geospatial data sources. This system has been implemented as re-usable components of the Qanary [16] question answering architecture. It provides a natural language interface for common users to express their information needs. Users commonly pose questions or information requests with a geospatial dimension to search engines, e.g., “Christmas market in Germany”, “Schools in London”, “Which countries border Greece?”, which will be transformed into queries in GeoSPARQ or in stSPARQL [17]. Those queries are asked over a large dataset which has been constructed by linking various geospatial data sources such as GADM, OSM and DBpedia. The OSM data was collected by converting the non-commercial shapefiles provided by the company GEOFABRIK into RDF triples using GeoTriples [18]. Then they interlinked the OSM entities with their equivalent entities of DBpedia by comparing them both nominally and geographically. However, the non-commercial shapefiles provided by GEOFABRIK contain deficient information in comparison with other OSM data sources. This motivated us to study alternative data sources (e.g., binary OSM files).

## 2.3 Transforming spatial data of various sources into RDF

### 2.3.1 TripleGeo

TripleGeo[3] [19] is an open-source ETL(Extract Transform Load) utility that can extract geospatial features from various sources and transform them into triples for subsequent loading into RDF stores. TripleGeo can directly access both geometric representations and thematic attributes either from standard geographic formats like OSM PBF, ESRI shapefiles, CSV and GeoJSON or widely used DBMSs. It can also reproject input geometries on-the-fly into a different Coordinate Reference System, before exporting the resulting triples into a variety of notations. Most importantly, TripleGeo supports the recent GeoSPARQL [20] standard endorsed by the Open Geospatial Consortium, although it can extract geometries into other vocabularies as well. This tool has been validated against OpenStreetMap layers with millions of geometries, opening up perspectives to add more functionality and to address much bigger data volumes.

Before attempting any conversion using TripleGeo, a configuration file must be prepared. This file lists crucial properties that define how input data will be accessed, where they will be exported and into which format, as well as optional features (e.g., reprojection into another spatial reference system).

3 https://github.com/SLIPO-EU/TripleGeo.

### 2.3.2 GeoTriples

Similarly to TripleGeo, there is another ETL utility that can extract geospatial features from various sources and transform them into RDF triples, known as GeoTriples[4] [18]. GeoTriples is developed by the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens and it is capable of converting data from various data sources like spatially-enabled relational databases (PostGIS and MonetDB), ESRI shapefiles and XML, GML, KML, JSON, GeoJSON and CSV documents. GeoTriples comes in two forms: a single-node implementation and an implementation that supports the parallelization of the process using Apache Hadoop[5] [21] for dealing with big geospatial data.

4 http://geotriples.di.uoa.gr/
5 https://hadoop.apache.org/

# 3. CONVERSION OF OPENSTREETMAP DATA INTO RDF

This implementation refers to the construction of a wrapper of TripleGeo [19] known as TripleGeo_Forwarder, which simplifies the conversion of OpenStreetMap data into RDF triples. This is accomplished by automating the procedure of collecting the input OSM data. Its main routine is to download the requested OSM datasets from Geofabrik, and then to forward them to TripleGeo in order to convert them into RDF triples. The downloaded datasets are in the PBF file format which is the most complete OSM data source (Section 2.2.2). Also, the information is compressed in PBF datasets and hence processing them faster.

## 3.1 OpenStreetMap data to RDF

Nowadays, geospatial information is incredibly useful. It is required in a wide range of applications like weather pattern predictions and satellite navigation services. However, geospatial information was not available to the public due to restrictions on the use or availability of map information across much of the world. As a result, this motivated many people to construct and to contribute to OpenStreetMap which is a crowd-sourced project that provides geospatial information freely to the public.

OpenStreetMap contains a huge amount of useful information about the whole world such as points of interest (POI), administrative units, geometries, railways systems, waterways, and other geospatial information. This information is accessible to the public to utilize it, but its content is only provided in certain proprietary formats like XML, PBF or ESRI shapefiles. Those formats follow a predefined schema making them hard to integrate with other geospatial or regular data from different data sources. As a result, the integration process becomes time-consuming and complex, and usually requires manual integration and programming efforts to ensure that the meaning of the provided information will be processable. This data integration problem prevents the users to better utilize their data by complicating the process of constructing a unified dataset, that would allow the users to better understand the provided information.

RDF can efficiently solve this problem. Having the data stored in RDF triples makes information seeking easier by allowing exploration, editing, and interlinking of heterogeneous information sources with a spatial dimension. Furthermore, it facilitates the integration process with the view to produce a unified dataset and provides a series of beneficial tools like querying and inferencing which can improve the analysis of the data. Also, users have the capability to use GeoSPARQL [20] in order to perform not only regular queries but also complex spatial calculations similar to those in a Geographic Information System (GIS). Last but not least, RDF triples can be transformed back to any file format using TripleGeo or other relevant software, allowing users to design their desired schema of their produced dataset.

Moreover, OSM data in the form of RDF triples will contribute to the enrichment of the Semantic Web with geospatial information [22]. Traditional GIS offer rigid sets of geographic features, thus integrating external datasets into these systems is a complex task. However, combining the strengths of Linked Data and GIS systems could spur the transition from islands of isolated GIS to a geospatially enriched Linked Data Web where geographic information can easily be integrated and processed. This will enable particularly designed softwares and users to be capable of producing new information by better utilizing the geospatial dimension, like aggregations based on certain area.

This implementation enhances the attempt of providing OSM data in the form of RDF, by utilizing the TripleGeo tool and massively converting OΣM PBF files into RDF triples.

## 3.2 TripleGeo_Forwarder

### 3.2.1 Purpose

TripleGeo_Forwarder's primary purpose is to massively convert OSM data into RDF triples. This is accomplished by downloading the PBF datasets of the requested areas and forwarding them to TripleGeo which will convert them into RDF triples. The PBF datasets contain all the OSM entities and all of their tags, so the produced triples will not lack information. Therefore, it is an add-on of TripleGeo that simplifies the conversion of OSM data by automating the process of collecting and storing it, and by executing TripleGeo for a series of datasets.

### 3.2.2 Geofabrik's Download Server

Geofabrik provides a server known as Geofabrik's Download Server[6] which contains data extracts from the OpenStreetMap project in PBF, XML and ESRI shapefile formats. Its datasets are normally updated every day by an automated process. Geofabrik provides two versions of shapefiles, one free of charge, and a commercial one. The free shapefiles contain just a subset of the feature classes of OSM such as POIs, roads, waterways and railway system, while the commercial ones contain additional feature classes and more information about the entities. In this implementation, we use this server in order to download the PBF datasets of the requested regions, since those contain the most information.

### 3.2.3 Description

The user specifies in the configuration file the names of the regions he wants to convert into triples, separated by semicolons(";"). For those regions, TripleGeo_Forwarder will download from the Geofabrik download server the PBF datasets, since those contain the most information, and then it will forward them to TripleGeo. Before executing TripleGeo for each downloaded dataset, it will construct a temporary configuration file which will contain all the necessary information for the related dataset likes its location and its format, and it will be used as an argument for the execution. The produced triples will be located in the location which was specified in the configuration file.

Since Geofabrik's download server updates its content daily, it is urgent to always use the most recent ones as they could probably hold new or more accurate information. Therefore, if a set of data for a requested region already exists but it was downloaded on a previous day, TripleGeo_Forwarder will ignore it and it will start the process of downloading a new set. Thus, in this way we reassure that TripleGeo will always be executed with the most recent and the most updated datasets.

Regarding the process of downloading, a list of regions must be specified for which their OSM data will be extracted from Geofabrik and then transformed. This list must contain the names of the regions for which Geofabrik must provide a dataset, and the URL to their dataset. This list must be stored in an external file. TripleGeo_Forwarder advises this list in order to find the locations of the requested datasets and to download them. Furthermore, this file can be manually constructed, or otherwise, it will be generated by a procedure

6 https://www.geofabrik.de/data/download.html

which is executed only when it is absent. This procedure implements web scraping techniques that parse recursively all the tables and the sub-tables that exist in the Geofabrik download server, constructing this file. This way we ensure that the produced file contains all the URL of all the existing datasets, but more importantly, it makes the process of its construction automated. Hence, in case Geofabrik enriches its tables by adding new datasets for new regions, for instance, creating a sub table containing all the sub-regions of Greece, it will be effortless to reconstruct the file. However, this procedure is strictly designed and based on the current structure of the website, so in case that it dramatically changes it will render this process incompetent of constructing this file and it will require to be adjusted to the new schema of the website.

Figure 4 describes the whole procedure and the functionality of its modules. The Forwarder is the primary module and gets as its input a configuration file that contains all the necessary information for its execution, such as the requested areas and the location where the downloaded datasets will be exported. Its main job is to download the datasets and then to execute TripleGeo with the necessary arguments. In case geofabrik_areas.ini file doesn't exist, Forwarder will call Ini_Constructor which is responsible for its construction. TripleGeo will convert the input dataset into RDF triples and it will store them to the location that was specified in the configuration file.

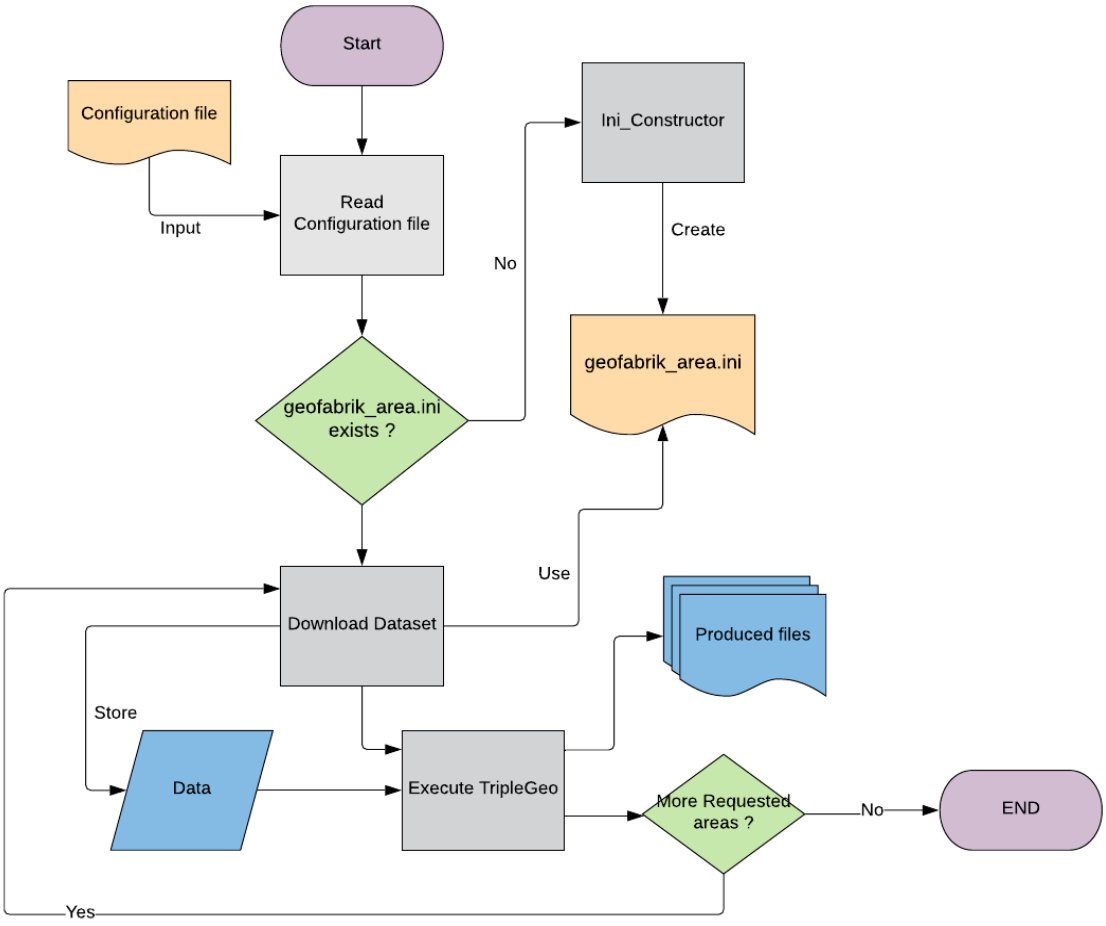


**Figure 4: Flow of the Execution of Fowarder**

### 3.2.4 Performance Results

TripleGeo_Forwarder was tested against OSM data extracted for multiple regions. The virtual machine in which the tests were implemented has 16 CPU cores of 2.3 GHz, 32Gb RAM and 16Gb swap space. “Records” is a database terminology used to express real-world entities, represented in OSM. Table 2 and Figure 5 describes the tests that were implemented and their performances. In Figure 5 we can observe that the execution cost increases proportionally to the input dataset. However, not only the size of the input affects the execution cost, but also the complexity of the content of dataset.

While performing the conversion, TripleGeo indexes all OSM elements in memory. However, if the size of the input dataset is more than 2% of the JVM heap size, it is expected that memory will not be sufficient and the execution could probably lead to memory errors. The developers of TripleGeo were aware of this danger and implemented it in such a way that the indexing will be performed in the Disk in case the memory is expected to be insufficient. Fortunately, the VM that these tests were implemented, contained sufficient resources in order to avoid this situation and use only the RAM.

**Table 2: Resutls of TripleGeo_Forwarder**

| Regions | Dataset's Size (Mb) | Input Records | Output Triples | Execution Time (sec) |
|---|---|---|---|---|
| Cyprus | 12.3 | 13237 | 241578 | 14.327 |
| Latvia | 58 | 98337 | 1746977 | 59.838 |
| Serbia | 64 | 27328 | 605801 | 64.015 |
| Ireland and Northern Ireland | 141 | 74790 | 1340096 | 208.515 |
| Greece | 162 | 103473 | 2032970 | 213.712 |
| Portugal | 180 | 127278 | 2196691 | 264.298 |
| Belgium | 308 | 183863 | 3269995 | 449.314 |

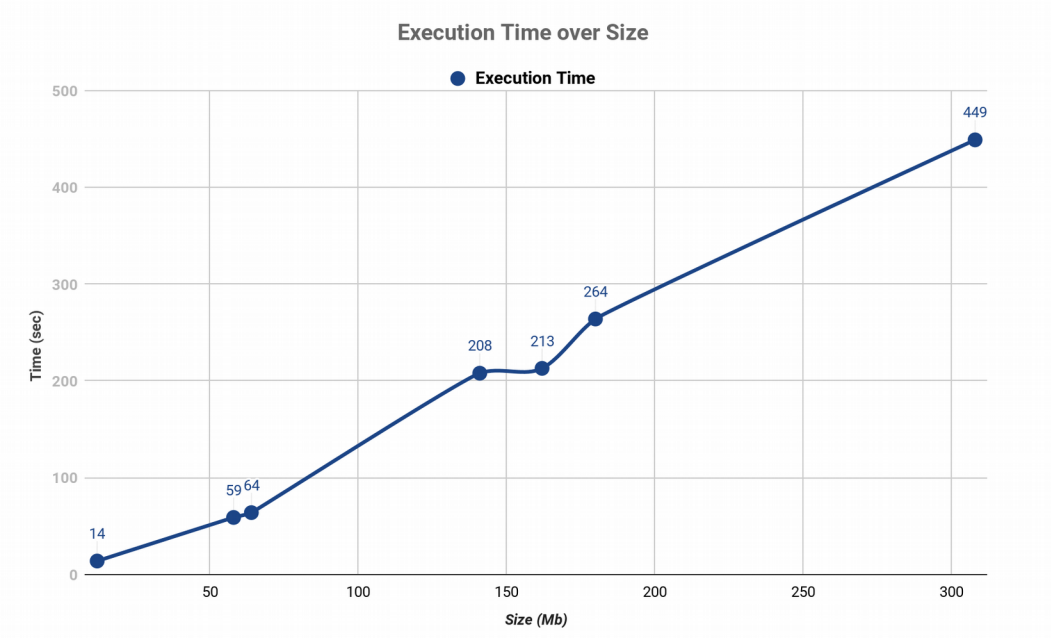


**Figure 5: Performance of TripleGeo_Forwarder**

# 4. SPARK EXTRACTOR

While TripleGeo is very effective in the transformation of geospatial data into RDF triples, its performance regarding the conversion of big files is poor and discouraging. Its multi-threaded execution supports only the conversion of multiple datasets and not the parallelization of a big dataset. In this implementation, TripleGeo is extended with the Spark technology that enables the parallelization of the execution in clusters. This enables the parallelization of the conversion of big geospatial data which led to significantly reduce the transformation time.

## 4.1 The Issue

TripleGeo is very effective in the transformation of several file formats into RDF triples concerning the geospatial information. It is capable of converting CSV, JSON, GeoJSON, PBF, ESRI shapefiles and data exported by widely used Database Management Systems (DBMS). However, TripleGeo does not support the parallelization of the process in order to reduce the execution time, hence large files containing complex geometrical structures require an excessive amount of time to complete their transformation. Nonetheless, TripleGeo offers the possibility to concurrently execute multiple files of the same format, in different Java threads. So users can split their data in separate individual datasets and instruct TripleGeo to convert them simultaneously. This function is provided for every geospatial format but requires from the users to have manually split the data into several pieces which is a hard task especially for complex file formats like ESRI shapefiles. This splitting is not executed by TripleGeo so it must be carried out externally, using other softwares, and therefore the user must be involved which may lead to deformations and corrupted files.

In an attempt to improve the performance of the conversion of big geospatial datasets, we extended TripleGeo to work on top of Spark. This enables the parallelization of the execution without requiring any pre-process by the user. More importantly, it significantly reduced the execution time of big complex datasets.

## 4.2 Spark

Apache Spark[7] [23] is an open-source, distributed, general-purpose, cluster-computing framework, which was developed by the AMP lab[8] of UC Berkeley and it was later donated to the Apache Software Foundation[9] which has been maintaining it ever since. Spark provides an ease of use APIs for many programming languages like Java, Scala, and Python and also powers a stack of libraries including SQL, DataFrames, and Datasets, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. It is able to access diverse data sources including HDFS and can be run in a cluster or in a standalone mode.

Spark uses a master/worker architecture. There is a driver that talks to a single coordinator called *master* which manages *workers* in which executors run. A Spark driver (aka an application's driver process) is a JVM process that hosts the SparkContext for a Spark application and it is the master node. It is responsible to split the job into tasks, to schedule

7 https://spark.apache.org/
8 https://amplab.cs.berkeley.edu/
9 https://www.apache.org/

them to run on executors and to coordinate the overall execution. Executors run in different JVM processes and they are distributed agents that execute the tasks in parallel (or sequentially).

At the core of Apache Spark is the notion of data abstraction as a distributed collection of objects. This data abstraction, called *Resilient Distributed Dataset* (RDD), allows users to write programs that transform these distributed datasets. RDDs are an immutable distributed collection of elements of the data that can be stored in memory or disk across a cluster of machines. The data is partitioned across machines in the cluster, which can be operated in parallel using a high-level API that offers transformations and actions. Transformations are functions that take an RDD as input and produce one or multiple RDDs as output. They do not change the input RDD since RDDs are immutable, but always produce one or more new RDDs by applying computations. The result RDD(s) will always be different from the input. According to the computation, the size can be either bigger (in case of a union) or smaller( in case of a filter) or even the same (in case of a map). By applying transformations the user incrementally builds an RDD lineage with all the parent RDDs of the final RDD(s) and they will not be executed before calling an action since transformations are lazy processes. Actions are RDD operations that produce non-RDD values and evaluate the RDD(s) lineage graph by triggering the execution of the transformations. Also one of their main purposes is to send the data from executors to the driver. RDDs are fault tolerant as they track data lineage information to rebuild lost data automatically on failure. Similarly to RDDs, *Dataframes* are an immutable collection of data but the information is organized into named columns, like a table in a relational database. DataFrame allows developers to impose a structure onto a distributed collection of data allowing them to use higher-level functions like SQL queries.

## 4.3 Why Spark?

Nowadays, the world of Big Data is evolving rapidly leading to the of emerging new technologies that offer the promise of managing and analyzing large volumes of data faster, in a more scalable way, with cheaper implementation and maintenance costs. Apache Spark is one of the most notable of such technologies because it is probably one of the fastest, with a very easy way to use API and with a large active community. Next, we outline the main reasons for choosing Apache Spark as big data infrastructure for TripleGeo.

### Very fast Platform

It is proved that Spark is way faster than its predecessor MapReduce of Hadoop [24]. Spark is designed in a way that it transforms data in-memory and not in disk I/O. Hence, it cuts off the processing time of read/write cycle to disk and storing intermediate data in-memory. This reduces processing time and cost of memory at a time. Moreover, Spark supports parallel distributed processing of data, hence it is almost 100 times faster in memory and 10 times faster on disk. Furthermore, Apache Spark is developed using the Scala programming language which is faster than Java. Scala provides immutable collections rather than Threads in Java that helps in inbuilt concurrent execution.

### Great API

Apache Spark provides a native easy way to use API for Java which contains plenty of transformations and actions allowing programmers to more effectively reach their goals. Also, it comes up with a graphical user interface accessible through the browser, which informs the user about the status of the execution and about helpful information, which can assist in improving the application. Furthermore, it is important to mention that it is

supported by a large active community, which facilitates and encourages learning and the overall engagement with this technology.

**GeoSpark**

GeoSpark[10] [25] is an extension of Spark core implemented by the Data Systems Lab[11], in order to support spatial data types, indexes, and geometrical operations at scale. It is enriched with a set of out-of-the-box *Spatial Resilient Distributed Datasets* (SRDDs) that efficiently load, process, and analyze large-scale spatial data across machines.

## 4.4 Application Description

Before explaining the process of this implementation, it is important to explain the principal routine and the execution flow of TripleGeo. TripleGeo's primary module is Extractor which reads the input configuration file and initializes all the fundamental variables. Then for each input dataset, it creates a *Task* in a different thread which will perform the conversion. The execution of these datasets will be in parallel but the dataset must be of the same file format. If only one file is given as input then the whole procedure will be executed in a single thread and therefore it will be sequential. According to the type of the input file, the *Task* will call the proper converter which will collect the data from the dataset, convert them into triples and store them in the specified destination. As it is already mentioned, the execution will be in parallel only if multiple files of the same format will be given as input. In case of conversion of big files, users are required to split them is separates datasets which is a troublesome procedure.

In this implementation, we support the parallelization of CSV, GeoJSON and ESRI shapefiles and we have refactored the whole procedure by introducing new modules in order to not affect the flow of execution of the current version of TripleGeo. The whole process starts with *SparkExtractor* which performs similarly to *Extractor* but instead of constructing *Tasks* for the given datasets, it constructs a *SparkTask* which implements the Spark functionalities. Firstly, it initializes all the necessary Spark variables and then according to the data source it parses them properly. In cases of GeoJSON and CSV, the information is parsed is such a way in order to create a dataframe which will contain all the features of the initial source. In the case of ESRI shapefiles, in which we will focus on, the information is read using GeoSpark which converts it in a Spatial RDD. This SRDD contains the content of the shapefile in records of Java Topologies Suite (JTS) [26] Geometries instances. Then the procedure is the following:

- A Spark transformation is applied which transforms the input records into Maps.
- Using a Spark action the produced Maps are passed to a new specially designed converter.
- The converter performs the conversion into RDF triples.

The whole execution is performed in parallel because during the initialization of the RDD (or the Dataframe) the input dataset is split into data chunks which are treated individually. Those data chunks known as *Partitions* are logical chunks of a large distributed set of data that an RDD represents. The input dataset will be read and stored in partitions in different Workers and then each Worker will perform an independent conversion. The number of partitions is specified by the user in the configuration file and the performance of the whole procedure is highly related to it. More partitions mean the more parallelize the whole procedure will be and each Worker will have to convert a smaller chunk as the initial

10 http://geospark.datasyslab.org/
11 https://www.datasyslab.net/

dataset will be divided into more partitions of fewer records. However, that does not significantly mean that more partitions will lead to a faster execution. The optimal number of partitions highly depends on the available resources of the cluster or of the machine (in case Spark runs in a standalone mode) the application is executed in.

The transformation of the datasets can be parallelized because each partition can be converted individually from the others. The chunks of the dataset which are held in partitions contain independent records that TripleGeo will convert them into RDF triples and store them in separate files. Therefore the execution of each partition does not affect the execution of the rest, so there is no need for data reshuffling or inner communication during the process. Each partition will produce its own RDF file containing the triples generated by its dataset chunk. These files can be easily concatenated since integration is a simple process for data that follows the RDF model.

The whole process is divided into two stages, as it is represented in Figure 6. GeoSpark, first, loads the shapefile in a distinct partition, so the first stage is dedicated to repartitioning it to the requested number of partitions. In the second stage, the Spark transformations and actions are performed, which in our case are a map transformation and a forEachPartition action. During the map transformation, the input instances are transformed into Maps and during the action, TripleGeo is executed for each partition.

It is important to mention that GeoSpark reads a folder which must contain all the requisite files of a shapefile like the .shp, .shx and .dbf. As a result, the input path must be the location of the folder that contains the shapefiles. Furthermore, each partition will produce a separate RDF file which will hold the triples that were generated by the records of its chunk.

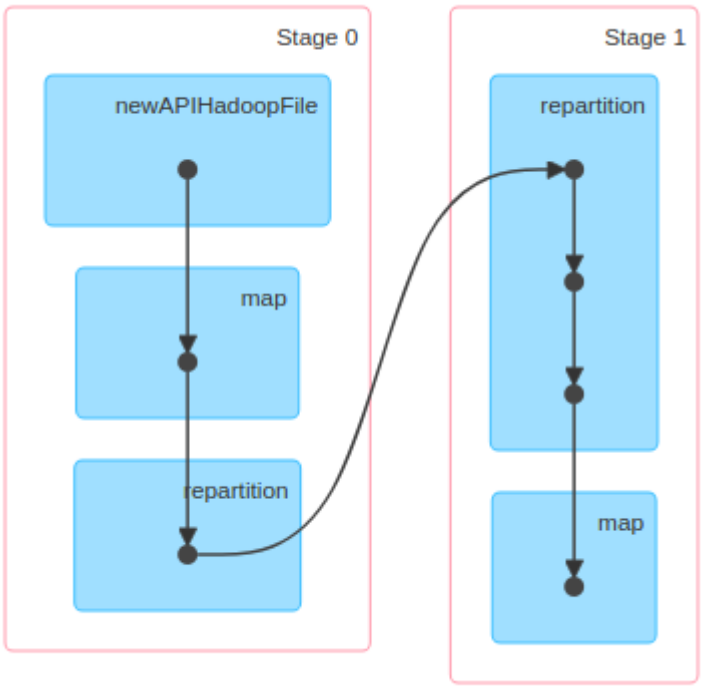


**Figure 6: Directed Acyclic Graph (DAG) of SparkExtractor**

## 4.5 Evaluation

### 4.5.1 Experimental Setup

The application was assessed with multiple files of diverse sizes. The following tests were conversions of ESRI shapefiles provided by the OSM download server. Those tests were implemented in a Virtual Machine (VM) with 4 CPU cores of 2.1GHz, 4Gb memory and 2Gb of swap space. Furthermore, all tests were conducted with “cold cache” which means that the cache memory was empty or contained irrelevant with the procedure data. It is also important to mention that the case of one (1) partition stands for the traditional routine of TripleGeo, the Extractor which does not support the parallelization of ESRI shapefiles and does not apply any Spark operation. The tested datasets and their features are listed in Table 3.

**Table 3: Experimental Setup**

| Name | Description | Type | Size | Records (Input) | | Produced Triples |
|---|---|---|---|---|---|---|
| OSM_POIS_GR | Contains the POIs of Greece. | Shapefile | 15M | 76835 | | 1043040 |
| OSM_POIS_SP | Contains the POIs of Spain. | Shapefile | 106M | 373088 | | 5122008 |
| OSM_ROADS_GR | Contains the road system of Greece. | Shapefile | 453M | 776386 | | 7745024 |
| OSM_ROADS_SP | Contains the road system of Spain. | Shapefile | 1.6G | 3260622 | | 34372473 |
| OSM_ROADS_GER | Contains the road system of Germany. | Shapefile | 3.7G | 11107532 | | 115999413 |

### 4.6.2 Performance Results

The performance of the application was evaluated by executing it with various datasets of diverse sizes and contents as its input. The evaluation was twofold:

- **Scale-up**: The transformation time of each dataset was examined by varying the number of partitions in order to find when the application offers the best utilization of the available system resources (CPU cores).
- **Scale-out**: The scaling of the application was also examined against varying input data sizes by fixing the number of partitions in each test.

## Scale-Up

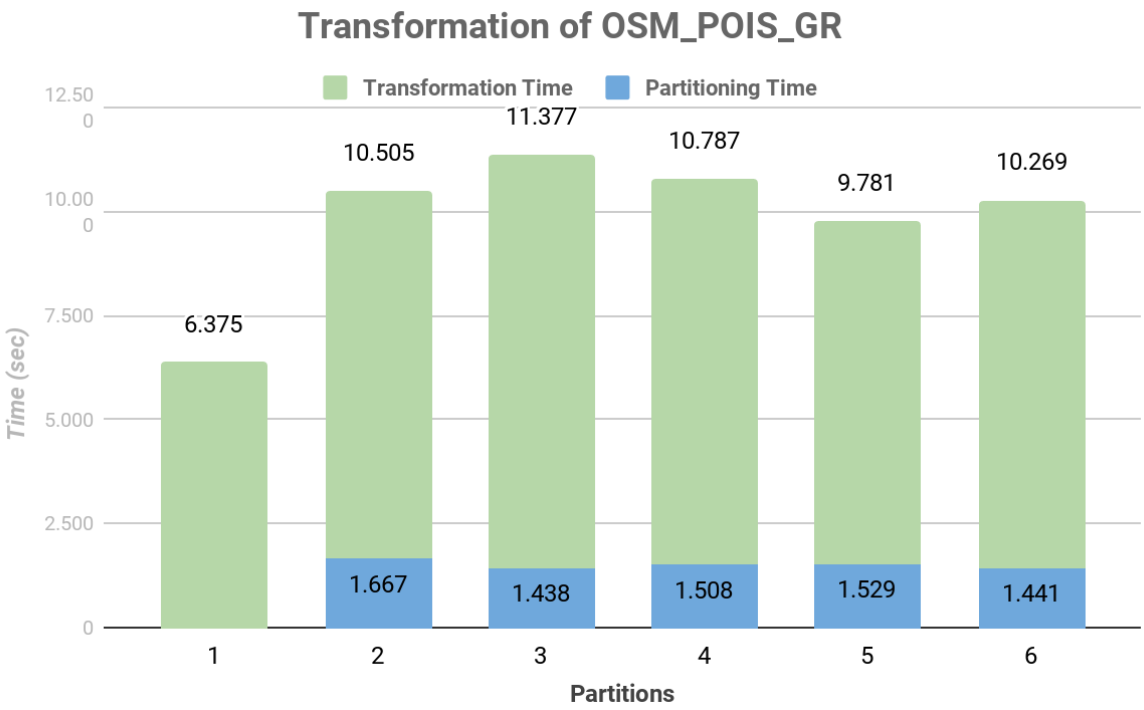


**Figure 7: Scale-up for dataset OSM_POIS_GR**

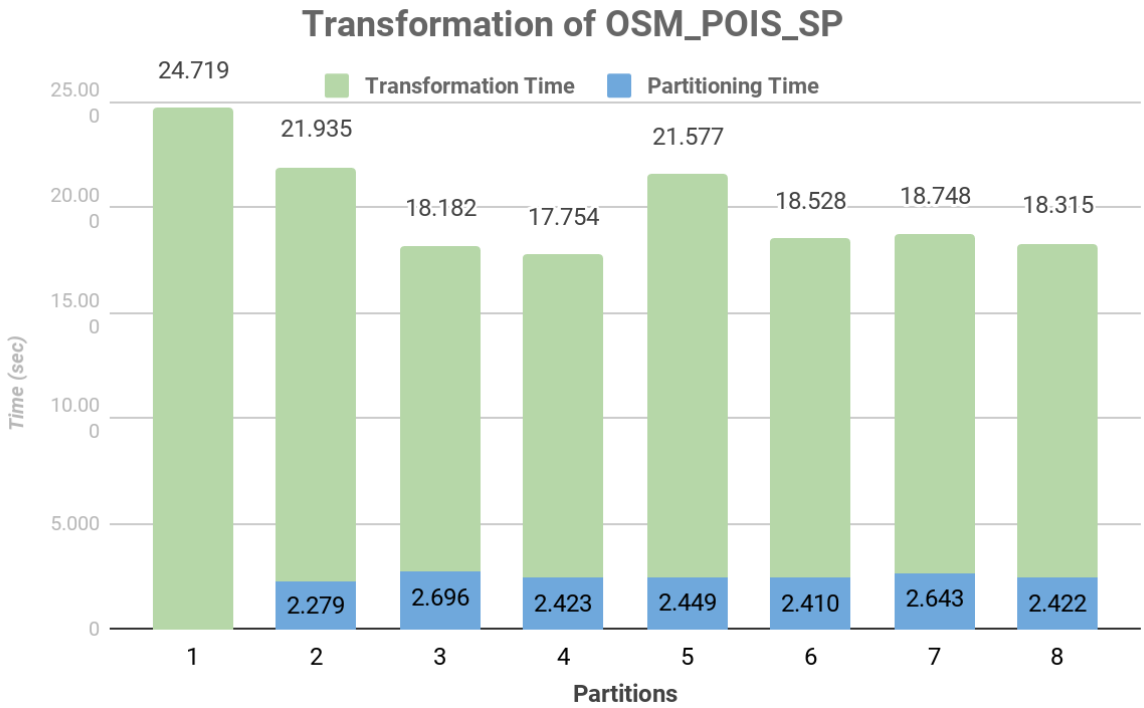


**Figure 8: Scale-up for dataset OSM_POIS_SP**

From Figure 7, we can deduce that the simple *Extractor* (case of one partition) is more effective against the *SparkExtractor* for small datasets. This is probably happening because *SparkExtractor* consumes some time in order to initialize the fundamental variables of Spark and in order to partition and distribute the data. Also, OSM_POIS_GR contains exclusively only POIs where all geometries are simple points and their transformation is an easy task for TripleGeo. However, as the size of the input dataset increases then we can observe the scalability provided by Spark. In Figure 8, we can observe the significant reduction of the execution cost, especially when we use 4 partitions. We reckon that the usage of 4 partitions is the optimal option because it makes the best out of the four CPU cores of the VM and because it performs the conversion of all the partitions completely in parallel.

It is very intriguing and very important to mention that when we split the dataset into more than four partitions (more than the amount of the available CPU cores), Spark doesn't execute all of them concurrently. Instead, Spark assigns the first four partitions to the CPU cores and executes them in parallel, while the rest of the partitions await for a core to complete its operations in order to be assigned to it. Therefore, in the case of five (5) partitions, the first four were executed simultaneously while the fifth was waiting and started when a core was released. This is not utterly a bad thing because it gives us the option to divide the initial dataset into more small partitions and thus to reduce their required execution time. We can observe from Figure 9 and from the following bar charts that the execution time is reduced after 5 partitions and it produces significantly decent results in the case of 8 partitions.

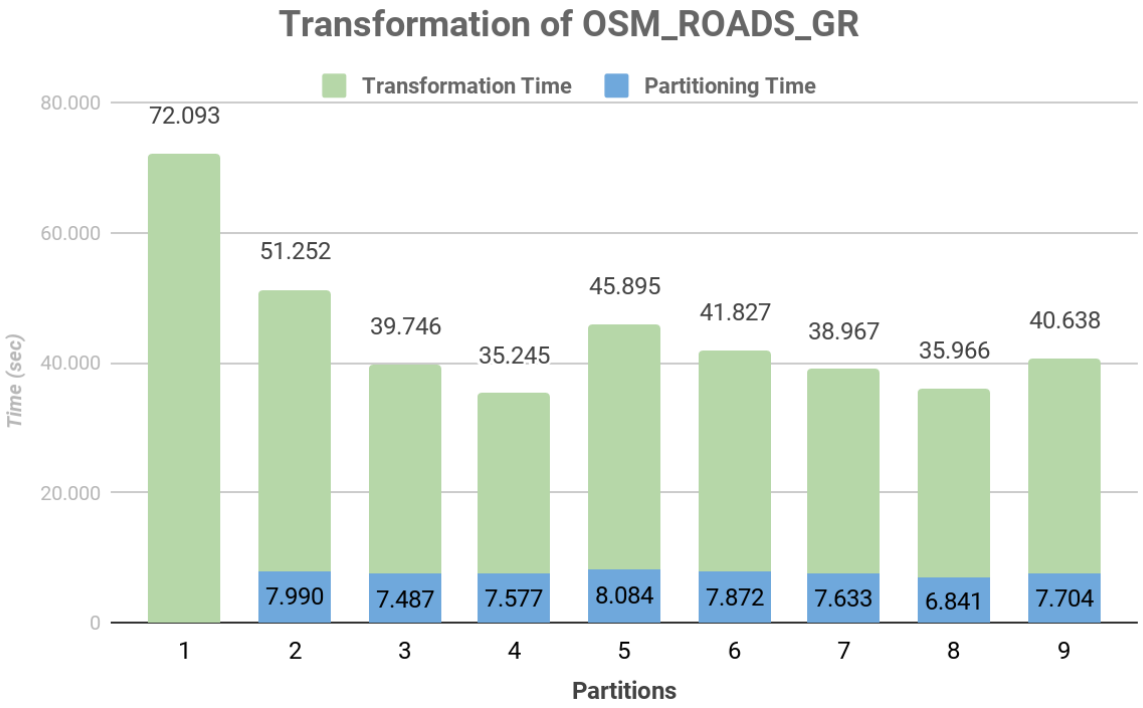


**Figure 9: Scale-up for dataset OSM_ROADS_GR**

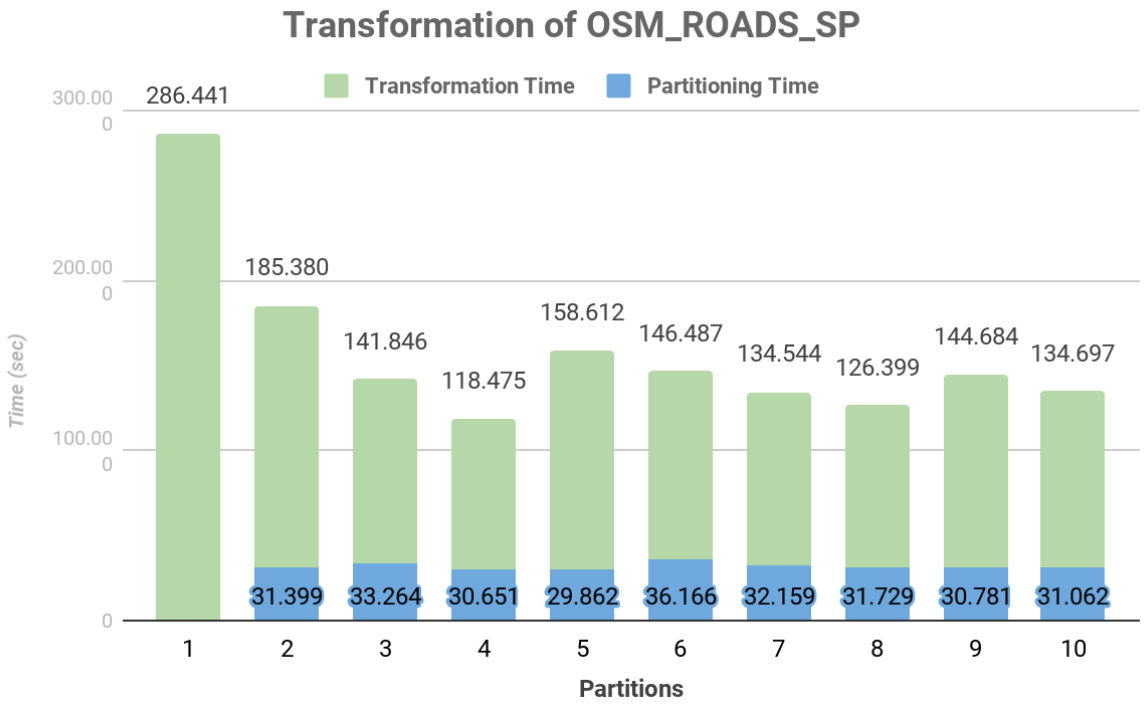


**Figure 10: Scale-up for dataset OSM_ROADS_SP**

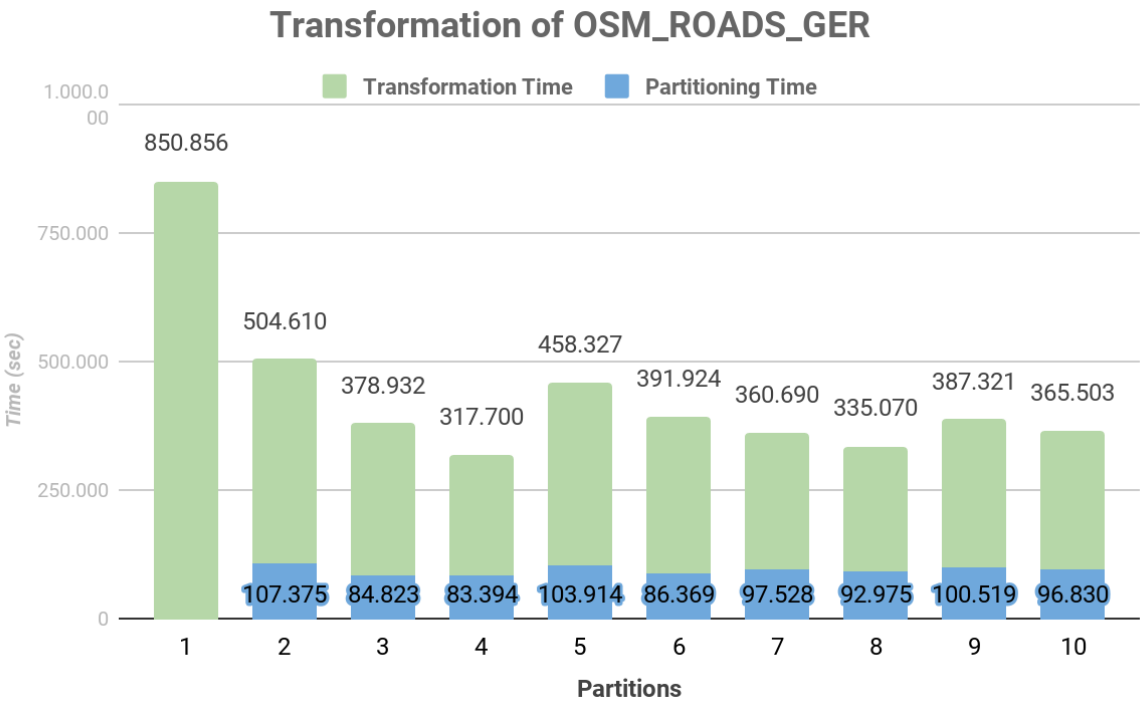


**Figure 11: Scale-up for dataset OSM_ROADS_GER**

It is beyond the shadow of a doubt that the performance of *SparkExtractor* regarding big datasets is remarkable. In Figure 11 where input dataset is relatively big, Spark managed to complete the execution in less than half of the time that it is required by the Extractor.

### Scale-Out

Figure 12 depicts the performance of the application for specific numbers of partitions against datasets of varying sizes. Each bullet represents the performance of the model using a dataset of a corresponding size. The datasets that were used are the ones that were stated in the setup section. Standalone stands for the execution of TripleGeo's main module, Extractor. The performances of the executions using 4 and 2 partitions are displayed because the case of 4 partitions has the optimal outcome and the execution cost using 2 partitions is almost half of the Standalone mode. There is no reason to display the performances of the executions with more than 4 partitions because they do not offer the best utilization of the available CPU cores.

In Figure 12, we can observe that the escalation of the transformation time over the input size is almost linear in every case. However, larger datasets have a more negative impact in the Standalone mode than in the other modes. In the end, the Standalone mode requires almost twice the time needed by the execution that used 2 partitions in order to complete its transformation. Since Spark requires a small period of time partitioning and distributing the input dataset, the performance of the Standalone is superior when the size of the input dataset is relatively small.

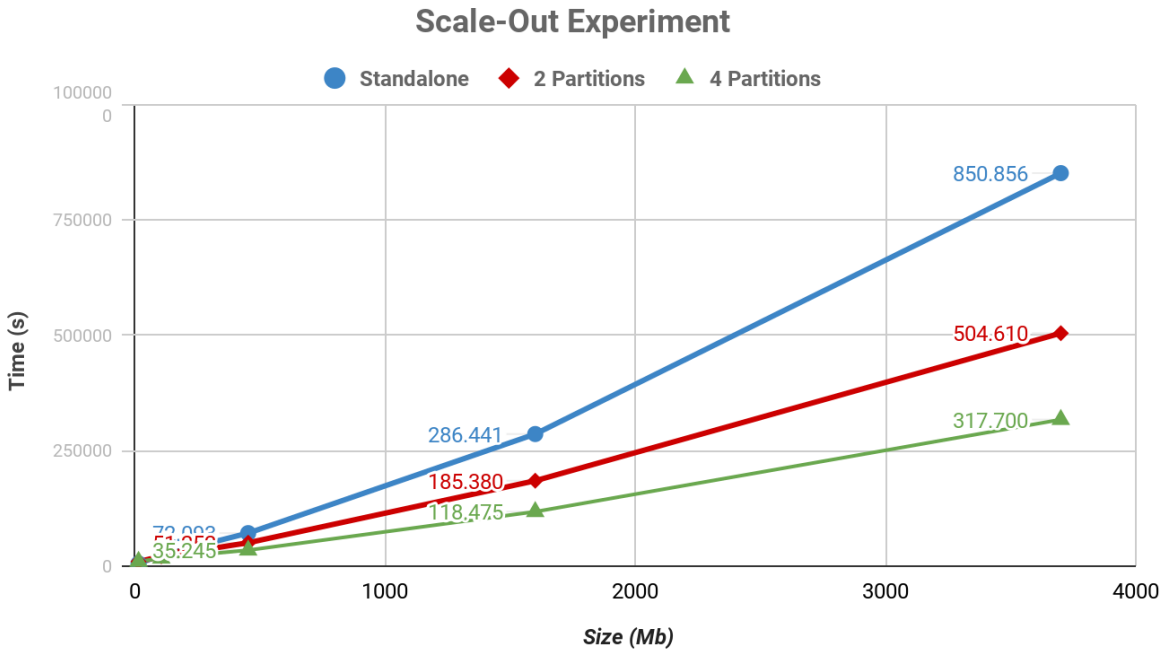


**Figure 12: Scale-Out Experiment**

# 5. YAGO EXTENSION WITH SPATIO-TEMPORAL KNOWLEDGE

In this chapter, we describe an extension of YAGO with temporal and spatial information about the former Greek administrative division that was described in the Kapodistrias plan. The first step was to collect the data from official sources and then to construct the Kapodistrias dataset that follows the RDF data model. Then we use this dataset to extend YAGO with official data about the former administrative divisions of Greece.

## 5.1 Greek Administrative Division

The former administrative division of Greece was based on the Kapodistrias plan and it was in effect from 1997 till 2011. Then it was replaced by the Kallikratis reform which implemented several changes and ıs in effect ever since. The Kapodistrias reform consisted of 910 Municipalities (Δήμοι) and 124 Communities (Κοινότητες), 54 Prefectures (Νομοί) and 13 Regions (Περιφέρειες). In the Kallikratis reform, there was a lot of changes but the most significant was that the majority of the prefectures were transformed into regional units and the rest were split up into multiple regional units. (Περιφερειακές Ενότητες). Moreover, many Municipalities and Communities were merged, while Regions remained intact. In more detail, the Kallikratis reform consists of 325 Municipalities (Δήμοι), 74 Regional Unis (Περιφερειακές Ενότητες) and 13 Regions (Περιφέρειες).

## 5.2 Kapodistrias Dataset Construction

### 5.2.1 Kapodistrias' Administrative Units

The first step towards the extension of YAGO was to collect the information about the former Greek administrative division. The main source from which the information was collected is the Wikipedia page that refers to the Kapodistrias plan. Even though, Wikipedia is known to be a reliable source we wanted to ensure that its information is credible. So its content was manually evaluated using the Greek legislation about the Kapodistrias plan. In the end, it was proved that it is a reliable source as it strictly follows the Greek legislation.

The desired information was collected using a Python script which was recursively scraping the Wikipedia pages collecting all the necessary information. In more detail, the collected are entities (administrative units) were

- 13 Regions (Περιφέρειες)
- 54 Prefectures (Νομοί)
- 1034 Municipalities and Communities (Δήμοι & Κοινότητες)
- 6166 Districts (Διοικητικές διαιρέσεις)

The next step was to construct an RDF graph that contains information about the former Greek administrative units, as stated in the Kapodistrias plan This dataset will follow the RDF data model so each administrative unit will be represented by a unique Uniform Resource Identifier (URI). The properties and relations that are contained in the dataset are:

- *<http://kr.di.uoa.gr/kapodistrias/ontology/officialName>* assigns the entities with their label.

- *<http://kr.di.uoa.gr/kapodistrias/ontology/id>* assigns the entities with a unique id.
- *<http://kr.di.uoa.gr/kapodistrias/ontology/division>* connects the entities with the type of their division (District, Community, Municipality, Prefecture or Region).
- *<http://kr.di.uoa.gr/kapodistrias/ontology/upperLevel>* assigns an entity with the upper level unit it belongs to.
- *<http://www.opengis.net/ont/geosparql#hasGeometry>* connects an entity with a geometry id. The geometry id works as an intermediate between the entities and their geometries.
- *<http://www.opengis.net/ont/geosparql#asWKT>* connects a geometry id with their geographical shape in the form of Well Known Text (WKT).
- *<http://kr.di.uoa.gr/kapodistrias/ontology/OfficialCreationDate>* assigns the entities with their creation date.
- *<http://kr.di.uoa.gr/kapodistrias/ontology/OfficialTerminationDate>* assigns the entities with the date they ceased to exist.

Figure 13 depicts some triples of the Kapodistrias dataset.

```
@prefix kapo_o:  <http://kr.di.uoa.gr/kapodistrias/ontology/>.
@prefix kapo_r: <http://kr.di.uoa.gr/kapodistrias/resource/>.
@prefix geo:  <http://www.opengis.net/ont/geosparql#>.

kapo_r:Περιφέρεια_Αττικής      kapo_o:division                 "Region".
kapo_r:Περιφέρεια_Αττικής      kapo_o:id                       "Kapodistrias_01000000".
kapo_r:Περιφέρεια_Αττικής      kapo_o:officialName             "Περιφέρεια Αττικής".
kapo_r:Περιφέρεια_Αττικής      kapo_o:upperLevel               "Greece".
kapo_r:Περιφέρεια_Αττικής      kapo_o:OfficialCreationDate     "1997-##-##".
kapo_r:Περιφέρεια_Αττικής      geo:hasGeometry    kapo_r:Geometry_Kapodistrias_01000000 .
kapo_r:Geometry_Kapodistrias_01000000      geo:asWKT     "MULTIPOLYGON (((...)))" .

kapo_r:Νομαρχία_Αθηνών      kapo_o:division                 "Prefecture" .
kapo_r:Νομαρχία_Αθηνών      kapo_o:id                       "Kapodistrias_01010000" .
kapo_r:Νομαρχία_Αθηνών      kapo_o:officialName             "Νομαρχία Αθηνών" .
kapo_r:Νομαρχία_Αθηνών      kapo_o:upperLevel               kapo_r:Περιφέρεια_Αττικής .
kapo_r:Νομαρχία_Αθηνών      kapo_o:OfficialCreationDate     "1997-##-##" .
kapo_r:Νομαρχία_Αθηνών      kapo_o:OfficialTerminationDate  "2011-##-##" .
kapo_r:Νομαρχία_Αθηνών      geo:hasGeometry    kapo_r:Geometry_Kapodistrias_01010000 .
kapo_r:Geometry_Kapodistrias_01010000      geo:asWKT      "POLYGON ((...))" .

kapo_r:Δήμος_Αθηναίων(Αθηνών)        kapo_o:division        "Municipality" .
kapo_r:Δήμος_Αθηναίων(Αθηνών)        kapo_o:id              "Kapodistrias_01010100" .
kapo_r:Δήμος_Αθηναίων(Αθηνών)        kapo_o:officialName            "Δήμος Αθηναίων" .
kapo_r:Δήμος_Αθηναίων(Αθηνών)        kapo_o:upperLevel  kapo_r:Νομαρχία_Αθηνών .
kapo_r:Δήμος_Αθηναίων(Αθηνών)        kapo_o:OfficialCreationDate    "1997-##-##" .
kapo_r:Δήμος_Αθηναίων(Αθηνών)  geo:hasGeometry  kapo_r:Geometry_Kapodistrias_01010100 .
kapo_r:Geometry_Kapodistrias_01010100          geo:asWKT          "POLYGON ((..))" .
```

**Figure 13: Triples of the Kapodistrias dataset**

### 5.2.2 The boundaries of Administrative Units

In order to enrich YAGO's spatial information, we extend YAGO with the geographic shapes of the administrative units in the form of polygons instead of points. The ATHENA[12] research centre provided us with an ESRI shapefile containing the geometries of every Municipality of the Kapodistrias plan. The geometries of the Prefectures were constructed by unifying the geometries of the Municipalities of each Prefecture. The same procedure was used in order to construct the geometries of the Regions. We transformed the geometries into Well Known Texts (WKT) and then we mapped them with the entities of our dataset. Unfortunately, we didn't manage to find the geographic shapes of the Districts. Figure 14 depicts the produced geometries.

**Figure 14: Geometries of the Administrative Units**

### 5.2.3 Temporal Information of Administrative Units

The contained temporal information in the data concerns the construction and termination(if it exists) dates of the administrative units. All the administrative units were constructed in 1997 when the Kapodistrias plan was set in effect. In 2011 the Kapodistrias plan was replaced by the Kallikratis reform, and therefore there were several changes in the administrative division of Greece. Many of them remained intact while others significantly changed. Those that altered in any way by the Kallikratis plan are assigned

12 https://www.athena-innovation.gr/

with a termination date. In more details, the date facts of each administrative order are the following:

- The Regions were not affected by the Kallikratis plan, so they are not assigned with a termination date. Therefore the only date fact that was inserted to them is:

  kapo_r:OfficialCreationDate "1997-##-##"^^xsd:date

- All the Prefectures transformed into Regional Units in the Kallikratis plan so they significantly changed. Therefore, all of them are assigned with a creation and a termination date. Their date facts are:

  kapo_r:OfficialCreationDate "1997-##-##"^^xsd:date
  kapo:_rOfficialTerminationDate "2011-##-##"^^xsd:date

- Municipalities are the most complex situation. Most of them were altered by the Kallikratis plan (either merged with other units or they utterly changed), while others remained untouched. In order to assign the Municipalities with their appropriate date facts, we constructed a set containing all the Municipalities that were not affected by the reform. The Municipalities that were in this set were assigned to only a construction date, while the rest were assigned with also a termination date. All the Communities were significantly changed so they are assigned with both a construction and a termination date.

- All the Districts were transformed into Municipal Communities. Similarly to Prefectures, their assigned date facts are

  kapo_r:OfficialCreationDate "1997-##-##"^^xsd:date
  kapo_r:OfficialTerminationDate "2011-##-##"^^xsd:date

### 5.3 YAGO Extension

After the construction of the Kapodistrias dataset, we extend YAGO with its information. However, YAGO already contains entities that represent the administrative units of Greece, so it is urgent to avoid duplicating its existing information. Thus, it is important to find the entities of YAGO that are equivalent to Kapodistrias entities and extend them with their new properties. The matching phase serves exactly this purpose by matching the entities of these two sources that represent the same administrative unit. This is accomplished by comparing them based on two factors, label similarity and geometrical distance, using the methodology proposed in [12]. Similar approaches are proposed in [9] [14]

During the label similarity process, entities that have similar names are matched. The similarity between two labels is expressed as a value and it is calculated using the Levenshtein distance. We accept that the labels refer to nominally the same entity, only when their similarity exceeds a threshold which we set in 0.82 after a lot of experimentation.

After the completion of label similarity follows the geometrical distance comparison, which is applied only to entities that pass the label similarity procedure. Many administrative units around the world share common names, so in this stage, we want to ensure that the matched entities refer to the same administrative units instead of different with the same names. For instance, an entity with the name “Athens” can refer to the Greek Prefecture of Athens or to the Athens city of Alabama of the US. This procedure checks if the Euclidean distance in the WGS:84 coordinate system between the geometry of the administrative units of Kapodistrias and the point provided by YAGO is smaller than a specific threshold,

which is set at 0.2 degrees. In case there are multiple entities of YAGO that are matched with the same resource, the entity that is closest, in terms of distance, to that resource is kept.

This procedure was executed separately for each administrative order in order to prevent matches among entities of different levels (e.g. Kapodistrias Regions were tested against the first-order administrative units of YAGO, etc.). The results are depicted in Table 4. Since the number of the entities was small, the correct matches were evaluated manually

**Table 4: Matching Phase Results**

| Administrative Orders | YAGO geo-class | Kapodistrias Entities | Matches | Correct Matches |
|---|---|---|---|---|
| Regions | first-order_AD | 13 | 12 | 12 |
| Prefectures | second-order_AD | 54 | 51 | 51 |
| Municipalities & Communities | third-order_AD | 1033 | 218 | 216 |
| Districts | populated_places | 6166 | 3471 | 196/200 |

The results are very satisfying for the first and second orders as almost all the entities were matched correctly. Regarding the third order, YAGO's entities about the Greek administrative division follow exclusively the Kallikratis plan, so only the administrative units of the Kapodistrias plan that were not changed by the Kallikratis reform were matched. Furthermore, the two wrong matches in the third order are administrative units that matched with their corresponding units that follow the Kallikratis plan, as they have similar labels and are located in the same place.

Moreover, some of YAGO's entities are declared as second and third order and as a result, they were matched in both cases. For instance, <Grevena_(regional_unit)> which is a YAGO entity, is declared as second and third order and it was matched with Grevena Prefecture and the Municipality of Grevena, which they are different entities. In order to avoid this issue, the priority was given to the highest administrative level, so in this case <Grevena_(regional_unit)> will only be matched with the Prefecture of Grevena.

The next step was to extend the matched entities by inserting them the properties of their matched Kapodistrias entities. Furthermore, we insert the unmatched Kapodistrias entities in YAGO in order to contain complete information about the former Greek administrative division.

```
@prefix kapo_r: <http://kr.di.uoa.gr/kapodistrias/resource/>.
@prefix yagoext_ont: <http://kr.di.uoa.gr/yago-extension/ontology/>.
@prefix yagoext_res: <http://kr.di.uoa.gr/yago-extension/resource/>.
@prefix yago: <http://yago-knowledge.org/resource/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix geo: <http://www.opengis.net/ont/geosparql#>.

kapo_r:Περιφέρεια_Αττικής              owl:sameAs                          yago:Attica_(region) .
yago:Attica_(region)            yagoext_ont:hasKapodistrias_Name      "Περιφέρεια Αττικής" .
yago:Attica_(region)            yagoext_ont:hasKapodistrias_Type      "Region" .
yago:Attica_(region)            yagoext_ont:hasKapodistrias_ID        "Kapodistrias_01000000" .
yago:Attica_(region)            yagoext_ont:hasKapodistrias_OfficialCreationDate "1997-##-##" .
yago:Attica_(region)      geo:hasGeometry      yagoext_res:Geometry_Kapodistrias_01000000 .
yagoext_res:Geometry_Kapodistrias_01000000  geo:asWKT      "MULTIPOLYGON (((...)))” .

kapo_r:Νομαρχία_Αθηνών         owl:sameAs   yago:geoentity_Nomarchía_Athínas_445408 .
yago:geoentity_Nomarchía_Athínas_445408  yagoext_ont:hasKapodistrias_Name
"Νομαρχία Αθηνών" .
yago:geoentity_Nomarchía_Athínas_445408  yagoext_ont:hasKapodistrias_Type  "Prefecture" .
yago:geoentity_Nomarchía_Athínas_445408  yagoext_ont:hasKapodistrias_UpperLevel
yago:Attica_(region) .
yago:geoentity_Nomarchía_Athínas_445408  yagoext_ont:hasKapodistrias_ID
"Kapodistrias_01010000" .
yago:geoentity_Nomarchía_Athínas_445408  yagoext_ont:hasKapodistrias_OfficialCreationDate
"1997-##-##".
yago:geoentity_Nomarchía_Athínas_445408 yagoext_ont:hasKapodistrias_OfficialTerminationDate
"2011-##-##".
yago:geoentity_Nomarchía_Athínas_445408  geo:hasGeometry
yagoext_res:Geometry_Kapodistrias_01010000 .
yagoext_res:Geometry_Kapodistrias_01010000    geo:asWKT          "POLYGON (( … ))”.
```

**Figure 15: Example of Matched Entities**

Figure 15 displays the extension of two YAGO entities with properties from their matched Kapodistrias entities. The YAGO entity *yago:Attica_(region)* was matched with the Kapodistrias entity *wiki:Περιφέρεια_Αττικής*, and therefore it was extended with its properties. Similarly the YAGO entity *yago:geoentity_Nomarchía_Athínas_445408* was matched with *wiki:Νομαρχία_Αθηνών*. The inserted properties are those that were stated in section 5.2.1, however the predicates are different. The predicates changed in such a way in order to indicate their relation to the Kapodistrias plan.

```
@prefix yagoext_ont: <http://kr.di.uoa.gr/yago-extension/ontology/>.
@prefix yagoext_res: <http://kr.di.uoa.gr/yago-extension/resource/>.
@prefix geo: <http://www.opengis.net/ont/geosparql#>.

yagoext_res:kapodistriasentity_Περιφέρεια_Ιονίων_Νήσων   yagoext_ont:hasKapodistrias_Name
"Περιφέρεια Ιονίων Νήσων" .
yagoext_res:kapodistriasentity_Περιφέρεια_Ιονίων_Νήσων   yagoext_ont:hasKapodistrias_ID
"Kapodistrias_08000000" .
yagoext_res:kapodistriasentity_Περιφέρεια_Ιονίων_Νήσων   yagoext_ont:hasKapodistrias_Type
"Region" .
yagoext_res:kapodistriasentity_Περιφέρεια_Ιονίων_Νήσων   yagoext_ont:OfficialCreationDate
"1997-##-##" .
yagoext_res:kapodistriasentity_Περιφέρεια_Ιονίων_Νήσων   geo:hasGeometry
yagoext_res:Geometry_Kapodistrias_08000000 .
yagoext_res:Geometry_Kapodistrias_08000000        geo:asWKT      "MULTIPOLYGON (((...)))".

yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς    yagoext_ont:hasKapodistrias_Name
"Νομαρχία Πειραιώς" .
yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς    yagoext_ont:hasKapodistrias_ID
"Kapodistrias_01020000" .
yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς    yagoext_ont:hasKapodistrias_Type
"Prefecture" .
yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς    yagoext_ont:hasKapodistrias_UpperLevel
yago:Attica_(region) .
yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς
yagoext_ont:hasKapodistrias_OfficialCreationDate              "1997-##-##" .
yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς
yagoext_ont:hasKapodistrias_OfficialTerminationDate           "2011-##-##" .
yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς    geo:hasGeometry
yagoext_res:Geometry_Kapodistrias_01020000 .
yagoext_res:Geometry_Kapodistrias_01020000        geo:asWKT      "MULTIPOLYGON (((..))" .
```

**Figure 16: Example of Unmatched entities**

Figure 16 depicts triples about Kapodistrias entities that did not match with any YAGO entities during the matching phase. Therefore we can deduce that the unmatched entities are unknown to YAGO and it does not contain any information about them. Those entities and their properties will be appended to its knowledge graph in order to further extend its content. We did not use the previous URIs but instead, we constructed and used new ones that indicate their reference to the Kapodistrias plan. Moreover, It is important to mention that even though the entity *yagoext_res:kapodistriasentity_Νομαρχία_Πειραιώς* did not match, its upper level matched with a YAGO entity, hence the upper-level predicate points to the matched YAGO entity.

## 5.4 Temporal Information in YAGO

During the construction of the matched and unmatched triples, we decided to not use the predicates that YAGO provides for the expression of the temporal information. The main reason that led us to this decision is the fact that some of the matched YAGO entities already have date facts. Therefore, in order to avoid collisions among the date facts and to not affect their existing temporal properties we constructed and used the new predicates *yagoext_ont:hasKapodistrias_OfficialCreationDate* and *yagoext_ont:hasKapodistrias_OfficialTerminationDate*. Using these new predicates enabled us to compare our new date facts with the existing ones and to evaluate them. Most of the entities are not assigned with any date fact and the temporal information of those few varies. Some of the date facts are based on the Kallikratis plan, while others are associated with dates that are not related to the Kapodistrias or Kallikratis plans. For instance the entity *yago:Icaria* contains as its creation date the date that it was freed from the Ottoman empire. The following paragraphs explain the origins of some of the date facts of YAGO about the Greek administrative units and mention their new inserted date facts.

An intriguing fact about many entities of YAGO is that even though they are described as Prefectures by their own labels, their creation date is set to “2011-##-##” which is the date that the Kallikratis plan was set in effect and therefore all the Prefectures ceased to exist. Furthermore, some of them contain the string “regional unit” (Περιφιερειακή Ενότητα) in their URIs which strengthens the fact that they refer to the Kallikratis plan. However, they contained multiple labels that indicate that they are Prefectures and therefore they were treated as Prefectures. For instance the entity *yago:Florina_(regional_unit)* contains the labels "Νομός Φλώρινας", "Nomós Florínis"@eng and "Nomos Florinas". As a result, it was matched with the entity kapo_r:*_Νομός_Φλώρινας* and therefore it was assigned with a creation and a termination date. Similar cases were the entities *yago:Euboea_(regional_unit), yago:Lasithi, yago:Lefkada_(regional_unit)* and many more.

Some of the first-order entities of YAGO such as the *yago:Epirus_(region)* and the *yago:Attica_(region)* have as their creation date the year “1987-##-##”. This date fact probably originates from Wikipedia which states that in this year these entities were established as Regions of Greece. We extended these entities with their Kapodistrias creation date only, since Regions did not alter by the Kallikratis plan.

The entity *yago:Magnesia_Prefecture* has as its creation date the year “1899-##-##” and the entity *yago:Samos_Prefecture* has as its creation date the year “1915-##-##”. Both of these dates are stated in the Greek version of these entities’ Wikipedia page and claim that in these dates they were established as Prefectures. There are plenty of other occasions which are quite similar to them. Both of them were extended with a creation and a termination date as all the Prefectures transformed into Regional Units by the Kallikratis plan in 2011.

It is important to mention that there is a lot of date facts for which we could not detect their origins. They do not originate from the Wikipedia pages and we were unable to find something relevant searching in the web.

# 6. CONCLUSION AND FUTURE WORK

To sum up, in this thesis we presented some implementations that enhance the Semantic Web with spatial and temporal information. Its primary purpose is to assist in the extension of YAGO by enriching its knowledge graph with spatiotemporal information about the former Greek administrative division. Moreover, the development of routines that contribute to the conversion of OSM data in RDF triples which will be used in a further extension of YAGO with spatial information. Furthermore, OSM data available in RDF triples are of major importance and it will be probably used by several other applications.

In more details, the first part is dedicated to the conversion of OSM data into RDF triples by constructing a routine that massively collects and converts OSM data into RDF. It is important to mention that the implementation converts the PBF datasets which is the most complete OSM data source, in contrast with other like Geofabrik's non-commercial shapefiles. In the second part, we extended TripleGeo to work on top of Spark. This enables the parallelization of the conversion of big geospatial data which offers magnitude performance gains compared to standalone execution. The third part of this thesis refers to the enrichment of YAGO's knowledge graph with temporal and spatial information about the former Greek administrative division. This is a small part of a bigger project conducted by prof. Koubarakis and N. Karalis, in which they extend YAGO's knowledge graph with spatial information from several data sources.

As future work, we plan to convert a big portion of OSM data into RDF and to use it in order to extend YAGO's knowledge graph more spatial information. Additionally, to further extend YAGO's knowledge graph with more temporal information about other countries administrative division and to develop a question answering system over the extended knowledge graph which will enable querying over the geographical and the temporal dimensions. Last but not least to adjust the data to fit in the SPOTLX data model of YAGO. SPOTLX is a data model in which every fact is directly assigned with its temporal and spatial information.

Regarding the provision of OSM data in RDF, we can construct a platform which will extend Geofabrik's download server by providing OSM data in the RDF data model. This platform will convert daily all the PBF datasets provided by Geofabrik into RDF triples by executing TripleGeo_Forwarder. Then the data will be available for download on a website. However, the transformation of such a big volume of data is a time-consuming process and therefore will require a machine with sufficient resources in order to be capable of completing the process in a reasonable period of time. Regarding to the Spark implementation of TripleGeo, It will be very interested to evaluate it in a cluster and to study its results. Spark is developed to work on a cluster, so it will expose the application to more hardware resources which Spark will utilize in order to improve its performance. This will allow Spark to split the input dataset into more partitions which will be executed concurrently in different machines. Therefore, it is expected that it will lead to a significant reduction of the execution time. Moreover, except for ESRI shapefiles, GeoJSON and CSV, TripleGeo supports the conversion of more geospatial data sources like JSON, XML, PBF and RDBM systems. So this implementation can be extended in order to support the parallelization of those data sources.

# ABBREVIATIONS - ACRONYMS

| | |
|---|---|
| W3C | World Wide Web Consortium |
| RDF | Resource Description Framework |
| SPARQL | SPARQL Protocol and RDF Query Language |
| URI | Uniform Resource Identifier |
| RDBMS | Relational Database Management System |
| LOD | Linked Open Data |
| ETL | Extract, transform, load |
| OSM | OpenStreetMap |
| KB | Knowledge Base |
| GIS | Geographic Information System |
| POI | Point Of Interest |
| RDD | Resilient Distributed Dataset |
| SRDD | Spatial Resilient Distributed Dataset |
| WKT | Well Known Text |

# REFERENCES


[1] T. Berners-Lee, J. Hendler, and O. Lassila, 'The Semantic Web', *Sci. Am.*, vol. 284, no. 5, pp. 34–43, 2001.
[2] 'World Wide Web Consortium', in *SpringerReference*.
[3] S. Powers, *Practical RDF: Solving Problems with the Resource Description Framework*. 'O'Reilly Media, Inc.', 2003.
[4] J. M. Hancock, 'SPARQL (SPARQL Protocol and RDF Query Language)', in *Dictionary of Bioinformatics and Computational Biology*, 2004.
[5] A. Jentzsch, 'Linked Open Data Cloud', in *X.media.press*, 2014, pp. 209–219.
[6] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, 'DBpedia: A Nucleus for a Web of Open Data', in *Lecture Notes in Computer Science*, 2007, pp. 722–735.
[7] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez, and D. Vrandečić, 'Introducing Wikidata to the Linked Data Web', in *Lecture Notes in Computer Science*, 2014, pp. 50–65.
[8] F. M. Suchanek, G. Kasneci, and G. Weikum, 'YAGO: A Large Ontology from Wikipedia and WordNet', *Journal of Web Semantics*, vol. 6, no. 3, pp. 203–217, 2008.
[9] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum, 'YAGO2', in *Proceedings of the 20th international conference companion on World wide web - WWW '11*, 2011.
[10] D. Ahlers, 'Assessment of the accuracy of GeoNames gazetteer data', in *Proceedings of the 7th Workshop on Geographic Information Retrieval - GIR '13*, 2013.
[11] T. Rebele, F. Suchanek, J. Hoffart, J. Biega, E. Kuzey, and G. Weikum, 'YAGO: A Multilingual Knowledge Base from Wikipedia, Wordnet, and Geonames', in *Lecture Notes in Computer Science*, 2016, pp. 177–185.
[12] M. Koubarakis, N. Karalis, 'Extending the YAGO Knowledge Graph with Geospatial Knowledge'.
[13] C. Wootton, 'ISO 8601 Date Format Output', in *Developing Quality Metadata*, 2007, pp. 419–420.
[14] S. Auer, J. Lehmann, and S. Hellmann, 'LinkedGeoData: Adding a Spatial Dimension to the Web of Data', in *Lecture Notes in Computer Science*, 2009, pp. 731–746.
[15] D. Punjani *et al.*, 'Template-Based Question Answering over Linked Geospatial Data', in *Proceedings of the 12th Workshop on Geographic Information Retrieval - GIR'18*, 2018.
[16] A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix, and C. Lange, 'Qanary – A Methodology for Vocabulary-Driven Open Question Answering Systems', in *Lecture Notes in Computer Science*, 2016, pp. 625–641.
[17] M. Koubarakis and K. Kyzirakos, 'Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL', in *Lecture Notes in Computer Science*, 2010, pp. 425–439.
[18] K. Kyzirakos *et al.*, 'GeoTriples: Transforming Geospatial Data into RDF Graphs Using R2RML and RML Mappings', *SSRN Electronic Journal*, 2018.
[19] K. Patroumpas, M. Alexakis, G. Giannopoulos, S. Athanasiou, 'TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples'.
[20] 'GeoSPARQL Query Tool - A Geospatial Semantic Web Visual Query Tool', in *Proceedings of the 10th International Conference on Web Information Systems and Technologies*, 2014.
[21] E. Lee and U. Syed, *Apache Hadoop: Invent the Future*. Consultantsnetwork, 2015.
[22] M. Koubarakis, K. Bereta, C. Nikolaou, and G. Stamoulis, 'Linked Geospatial Data', in *Encyclopedia of Big Data Technologies*, 2018, pp. 1–8.
[23] H. Luu, 'Introduction to Apache Spark', in *Beginning Apache Spark 2*, 2018, pp. 1–13.
[24] Y. Samadi, M. Zbakh, and C. Tadonki, 'Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks', *Concurr. Comput.*, vol. 30, no. 12, p. e4367, 2017.
[25] J. Yu, J. Wu, and M. Sarwat, 'GeoSpark', in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '15*, 2015.
[26] 'Java Topology Suite (JTS)', in *SpringerReference* .