ExtremeEarth

H2020 - 825258

Deliverable

D3.6

Software for interlinking geospatial RDF data sources-v2

George Papadakis, George Mandilaras, Dimitris Bilidas and Manolis Koubarakis

December 30, 2021

Status: Final
Scheduled Delivery Date: 31/12/2021

Executive Summary

The deliverable D3.2 is part of WP3, whose objective is to develop a set of tools for querying, integrating and running extreme analytics over the big information and knowledge that will be mined from Copernicus data and other auxiliary data sources using the techniques of WP2. This information and knowledge will be encoded as linked geospatial data and will be integrated with other open linked data sources to be demonstrated in the two use cases of ExtremeEarth.

In this deliverable, we present the algorithms for Geospatial Interlinking that were developed in the context of Task 3.2 during the second half of the project. They apply to spatial entities, i.e., geometries on Earth's surface, of two types: LineStrings and Polygons. We have developed techniques both for both batch and progressive processing. We also present the second version of JedAI-spatial, which offers an open-source library of our algorithms as well as the main techniques in the literature. JedAI-spatial conveys both serial methods, which run on a single CPU, and massively parallel ones, which run on top of Apache Spark. No other relevant tool has so many capabilities at the moment. We conclude with a discussion about further extensions that are currently under review.

Document Information

Contract Number	H2020 - 825258	**Acronym**	ExtremeEarth
Full title	ExtremeEarth		
Project URL	`http://earthanalytics.eu/`		
EU Project Officer	Riku Leppänen		

Deliverable	**Number**	D3.6	**Name**	Software for interlinking geospatial RDF data sources-v2			
Task	**Number**	T3.2	**Name**	Interlinking big geospatial data sources			
Work package	**Number**			WP3			
Date of delivery	**Contract**	31/12/2021		**Actual**	31/12/2021		
Status	Draft ☐ Final ☑						
Nature	Prototype ☑ Report ☐						
Distribution Type	Public ☑ Restricted ☐						
Responsible Partner	UoA						
QA Partner	NCSR-D						
Contact Person	Prof. Manolis Koubarakis						
	Email	koubarak@di.uoa.gr		**Phone**	+30 2107275213	**Fax**	+30 2107275214

Project Information

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number H2020-825258. The beneficiaries in this project are the following:

Partner	Acronym	Contact
National and Kapodistrian University of Athens Department of Informatics and Telecommunications (Coordinator)	UoA	Prof. Manolis Koubarakis National and Kapodistrian University of Athens Dept. of Informatics and Telecommunications Panepistimiopolis, Ilissia, GR-15784 Athens, Greece Email: (koubarak@di.uoa.gr) Tel: +30 210 7275213, Fax: +30 210 7275214
VISTA Geowissenschaftliche Fernerkundung GmbH	VISTA	Heike Bach Email: (bach@vista-geo.de)
The Arctic University of Norway Deptartment of Physics and Technology	UiT	Torbjørn Eltoft Email: (torbjorn.eltoft@uit.no)
University of Trento Department of Information Engineering and Computer Science	UNITN	Lorenzo Bruzzone Email: (lorenzo.bruzzone@unitn.it)
Royal Institute of Technology	KTH	Seif Haridi Email: (haridi@kth.se)
National Center for Scientific Research - Demokritos	NCSR-D	Vangelis Karkaletsis Email: (vangelis@iit.demokritos.gr)
Deutsches Zentrum für Luft-und Raumfahrt e. V.	DLR	Corneliu Octavian Dumitru Email: (corneliu.dumitru@dlr.de)
Polar View Earth Observation Ltd.	PolarView	David Arthurs Email: (david.arthurs@polarview.org)
METEOROLOGISK INSTITUTT	METNO	Nick Hughes Email: (nick.hughes@met.no)
Logical Clocks AB	LC	Jim Dowling Email: (jim@logicalclocks.com)
United Kingdom Research and Innovation - British Antarctic Survey	UKRI-BAS	Andrew Fleming Email: (ahf@bas.ac.uk)

Contents

1 Introduction **1**

2 Prior work **3**

3 Preliminaries **4**
3.1 Progressive Geospatial Interlinking 6

4 System Architecture **8**

5 Back-end: JS-core **10**
5.1 Serial Algorithms 10
5.1.1 Budget-agnostic algorithms 10
5.1.2 Budget-aware algorithms 12
5.1.3 Parallel Algorithms 14

6 Front-end: JS-gui **16**

7 Comparative Analysis **18**
7.1 Experimental Setup 18
7.2 Budget-agnostic Algorithms 19
7.3 Budget-aware Algorithms 21
7.3.1 Scalability Analysis 21
7.3.2 Discussion 22

8 Conclusions **24**
8.1 Ongoing Work 24
8.1.1 Supervised Filtering 24

List of Figures

1.1 Example of four topologically related geometries. 1

3.1 Examples of the DE-9IM topological relations between (a) pairs of LineStrings, and (b) LineStrings and Polygons [SDSN17]. 5
3.2 The Progressive Geometry Recall of budget-agnostic (batch) and budget-aware (progressive) algorithms. 7

4.1 The solution space of Geospatial Interlinking algorithms that can be constructed by JedAI-spatial. 8
4.2 JedAI-spatial's model-view-controller architecture. 9

5.1 The pipeline of budget-agnostic algorithms. 10
5.2 The pipeline of budget-aware algorithms. 12
5.3 The three-step pipeline of parallel, budget-agnostic algorithms in JedAI-spatial. . . 14

6.1 The workbench window of JedAI-spatial's Graphical User Interface. 16

7.1 Scalability analysis of the serial budget-agnostic algorithms with respect to their Filtering time (in seconds). 19
7.2 Scalability analysis of the serial budget-agnostic algorithms with respect to their Verification time (in minutes). 19
7.3 Scalability of parallel, budget-agnostic algorithms. 20
7.4 The speedup of Parallel Static and Dynamic Progressive GIA.nt as the number of cores increases over D_4, i.e., strong scalability 22
7.5 Scalability of Static (left) and Dynamic (right) Parallel Progressive GIA.nt over D_4. 22

8.1 The Approximate Geospatial Interlinking workflow. 25
8.2 Average recall, precision, training and prediction time of SVM, Naive Bayes, Logistic Regression and C4.5 Decision Trees over D_1-D_4 in combination with area-based features (ABF), boundary-based features (BBF), grid-based features (GBF), cardinality-based features (CBF) and all types of features. In each case, we consider atomic and composite features as well as their combination. 26
8.3 Evolution of average recall, precision, training and testing time over D_1-D_4 when using the atomic candidate-based features in combination SVM, Naive Bayes, Logistic Regression and C4.5 Decision Trees with respect to class size (on the horizontal axis). 27

List of Tables

7.1 Technical characteristics of the real pairs of datasets for Geospatial Interlinking. . 18

7.2 Performance of (a) Static and (b) Dynamic Progressive GIA.nt over D_5 for all weighting schemes in comparison to the optimal (Opt.) and the random (Rnd.) approach for budgets of 5M and 10M verifications. 21

8.1 Performance per classification algorithm. 28
8.2 The relative performance of Batch and Approximate GIA.nt. 28

1. Introduction

Geospatial data has escalated tremendously over the years. The outbreak of Internet of Things (IoT) devices, smartphones, position tracking applications and location-based services has skyrocketed the volume of geospatial data. For example, 100TB of weather-related data is produced everyday[1]; Uber hit the milestone of 5 billion rides among 76 countries already on May 20, 2017[2]. Web platforms like OpenStreetMap[3] provide an open and editable map of the whole world. Earth observation programmes like Copernicus[4] publish tens of terabytes of geospatial data per day on the Web[5]. For these reasons, geospatial data constitutes a considerable part of Semantic Web data, but the links between its data sources and their geometries are scarce in the Linked Open Data cloud [Ngo13, PMMK21].

Geospatial Interlinking aims to cover this gap by associating pairs of geometries with topological relations like those of the Dimensionally Extended 9-Intersection Model (DE-9IM) [EF91, CFvO93, CSE94]. In Figure 1.1 for instance, LineString g_3 intersects LineString g_4 and touches Polygon g_1, which contains Polygon g_2. Two are the main challenges of this task: (i) its inherently quadratic time complexity, because it has to examine every pair of geometries, and (ii) the high time complexity of examining a single pair of geometries, which amounts to $O(N \log N)$, where N is the size of the union set of their boundary points [CN97]. As a result, Geospatial Interlinking involves a high computational cost that does not scale to large Web datasets.

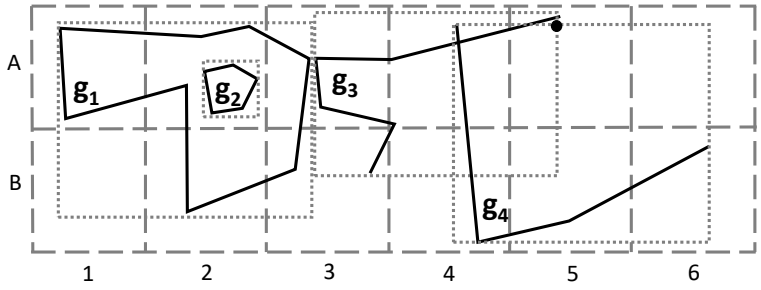

Figure 1.1: Example of four topologically related geometries.

Numerous algorithms aim to address these challenges by enhancing the time efficiency and scalability of Geospatial Interlinking. The most recent ones operate in main memory, reducing the search space to pairs of geometries that are likely to be topologically related according to a geospatial index [SSC+13, SJ14, PKNK18]. However, no open-source system organizes these algorithms into a common framework so as to facilitate researchers and practitioners in their effort to populate the LOD cloud with more topological relations. Systems like Silk [JIB10] and LIMES [NA11] convey only the methods developed by their creators, Silk-spatial [SK16] and RADON [SDSN17] respectively, while systems that could act as a library of established methods, such as stLD [SGD+19, SDVV20], are not publicly available. Moreover, no system supports progressive methods, neither for serial nor for parallel processing, even though they are indispensable for applications with limited or temporal resources [PMMK21].

To address these issues in the context of Task 3.2, UoA extended its open-source JedAI framework for Data Integration with new algorithms for interlinking big geospatial data sources. We actually develop a new module, called **JedAI-spatial**, which serves both as an open-source library of the state-of-the-art works in the field and as an open-source system that implements new methods that go beyond existing works in terms of efficiency and scalability. JedAI-spatial has the following unique characteristics:

[1] https://www.ibm.com/topics/geospatial-data
[2] https://www.uber.com/en-SG/blog/uber-hits-5-billion-rides-milestone
[3] https://www.openstreetmap.org
[4] https://www.copernicus.eu
[5] https://www.copernicus.eu/sites/default/files/Copernicus_DIAS_Factsheet_June2018.pdf

- It organizes the main algorithms for Geospatial Interlinking into a novel taxonomy that facilitates their use and adoption by practitioners and researchers based on three dimensions:
  1. *Space Tiling*, which determines the approach for reducing the search space,
  2. *Budget-awareness*, which distinguishes interlinking algorithms into batch and progressive ones, and
  3. *Execution mode*, which discerns between serial algorithms, running on a single CPU-core, and parallel ones, running on top of Apache Spark[6].
- Its intuitive user interface supports both novice and expert users: they simply have to select one of the available methods per workflow step and optionally configure it. It also simplifies the benchmarking of the main algorithms in the field through the workbench window that summarizes the performance of the algorithms executed so far. This is a crucial task for identifying the best approach for a particular task at hand, given that the experimental analyses in the literature are usually limited with respect to the variety in datasets or the baseline methods.
- Its modular and extensible architecture allows for easily incorporating improvements to all algorithms.
- It optimizes the implementation of existing algorithms, some of which have not been applied to Geospatial Interlinking before.
- It conveys new techniques that achieve competitive performance.
- We have publicly released the code of JedAI-spatial under the Apache License V2.0 at: `https://github.com/GeoLinker/GeoLinker`. In this way, we fulfill the two main challenges that arise in data integration [GHMT17]:
  1. the development of extensible, open-source tools, and
  2. the provision of solutions that apply not only to structured, but also to semi- or even un-structured data, due to its broad coverage of data formats.

The rest of the deliverable is structured as follows: Chapter 2 discusses prior work in the field, Chapter 3 provides background knowledge on Geospatial Interlinking, and Chapter 4 describes the architecture of JedAI-spatial, explaining the role of every component. Chapter 5 delves into its back-end, outlining the functionality of every supported method and highlighting our improvements that lead to significantly higher time efficiency. We briefly describe its front-end in Chapter 6 and perform an experimental analysis over large, real datasets in Chapter 7, providing useful insights into the pros and cons of the main algorithms.

[6] `https://spark.apache.org`

2. Prior work

In the Semantic Web domain, there are three related systems:

1. Silk [JIB10] constitutes an open-source, generic framework for Link Discovery that comprises a specialized component for Geospatial Interlinking, called Silk-spatial [SK16]. It exclusively supports a budget-agnostic, parallel method that runs on top of Apache Hadoop[1]. Its Filtering relies on a static, coarse-grained Equigrid, whose dimensions are defined by the user. Its Verification computes a single topological relation in each run.
2. LIMES [NA11] is an open-source, generic framework for Link Discovery with two algorithms for Geospatial Interlinking: ORCHID [Ngo13], which detects proximity relations in an efficient way, and RADON [SDSN17], which detects topological relations. RADON's Filtering employs a dynamic Equigrid, whose granularity depends on the input data, while its Verification relies on a hash map that stores all examined pairs in order to avoid repeated computations. Due to this data structure, RADON has been parallelized as a multi-core, shared-memory process, rather than a shared-nothing, MapReduce-based approach. Originally, its Verification computed a single topological relation per run, but was later extended to compute all relations at once, through the Intersection Matrix [ASN18].
3. stLD [SGD+19, SDVV20] is a tool that is crafted for Geospatial Interlinking. It is limited to budget-agnostic approaches, conveying a variety of algorithms, such as R-Tree, static Equigrid as well as hierarchical grid. Similar to JedAI-spatial and GIA.nt [PMMK21], its algorithms are capable of loading only the source dataset in main memory, reading the target one on-the-fly. stLD also supports massive parallelization on top of Apache Flink[2]. Its Verification supports both proximity and topological relations, but computes a single relation per run. Most importantly, though, its code has not been publicly released.

Note that none of these systems supports budget-aware algorithms. Such algorithms are only examined in [PMMK21], which proposed Progressive RADON and GIA.nt. Both have been integrated into JedAI-spatial, just like the proposed budget-agnostic algorithm, GIA.nt.

Note also that these systems are of limited scope, as the open-source systems (Silk and LIMES) convey only the algorithms developed by their creators, while stLD, being a proprietary software, cannot be used as a library of the state-of-the-art tools in Web applications nor can it be extended with novel techniques and pipelines.

To the best of our knowledge, no other systems similar to JedAI-spatial have been publicly released. The most relevant tools, which support parallelization on top of Apache Hadoop or Spark, are analyzed in [PKNK18]. They all support a variety of spatial queries, such as distance (range) and kNN queries, but in the context of Geospatial Interlinking, only their spatial join is applicable. Each tool essentially offers a single parallel algorithm for this join. The most recent and advanced systems are GeoSpark [YWS15] (a.k.a., Apache Sedona[3]), Spatial Spark [YZG15], Location Spark [TYM+16, TYM+20] and Magellan[4]. Their algorithms have been integrated into JedAI-spatial.

Finally, related to JedAI-spatial are two works that examine the relative performance of 10 serial, budget-agnostic algorithms that run in main memory [SSC+13, SJ14]. However, their experimental analyses focus on answering distance (range) queries about moving objects.

[1] http://hadoop.apache.org
[2] http://flink.apache.org
[3] https://sedona.apache.org
[4] https://github.com/harsha2010/magellan

3. Preliminaries

JedAI-spatial supports the two types of geometries in Figure 1.1:

1. the one-dimensional *LineStrings* or *Polylines*, which comprise a sequence of points and the line segments that connect the consecutive ones. E.g., a river or a road is represented by the set of points in its route, connected with straight line segments.
2. the two-dimensional *Polygons*, which usually comprise a sequence of connected points, where the first and last one coincide. E.g., the borders of a country are described by a polygon, usually a complex one.

Both types of geometries consist of an interior, a boundary and an exterior (i.e., all points that are not part of the interior or the boundary). These three parts are used by the *Dimensionally Extended nine-Intersection Model* (**DE-9IM**)[1], which has been standardized by the Open Geospatial Consortium (OGC), to define 10 topological relations between two geometries A and B:

1. `equals`(A, B): the interiors and boundaries of A and B are identical.
2. `disjoint`(A, B): A and B have no point in common, as the interior and boundary of A intersect neither with the interior nor with the boundary of B.
3. `intersects`(A, B): A and B have at least one point in common, i.e., their interiors or boundaries are not `disjoint`.
4. `touches`(A, B): the boundaries of A and B intersect but their interiors do not.
5. `within`(A, B): A is located inside the interior of B.
6. `contains`(A, B): `within`(B, A).
7. `covers`(A, B): all points of B lie in A's interior or boundary.
8. `covered-by`(A, B): `covers`(B, A).
9. `crosses`(A, B): A and B have some interior points in common but not all, while $dim(A) < dim(B)$ or $dim(B) < dim(A)$.
10. `overlaps`(A, B): A and B have some points in common but not all, while $dim(A) = dim(B)$.

Examples of these relations are shown in Figures 3.1(a) and (b). Note that $dim(g)$ amounts to 0, 1 or 2 if geometry g is a point, a line segment or an area, respectively. Note also that JedAI-spatial disregards redundant relations. For example, if $Contains(s, t) = true$, it does not materialize the equivalent $Within(t, s) = true$. Most importantly, JedAI-spatial, disregards the relation `disjoint` for two reasons [PMMK21]:

1. It provides no positive information for the relative location of two geometries.
2. It is impractical to compute it in the case of large input data, because it scales quadratically with the input size, given that the vast majority of pairs are disjoint.

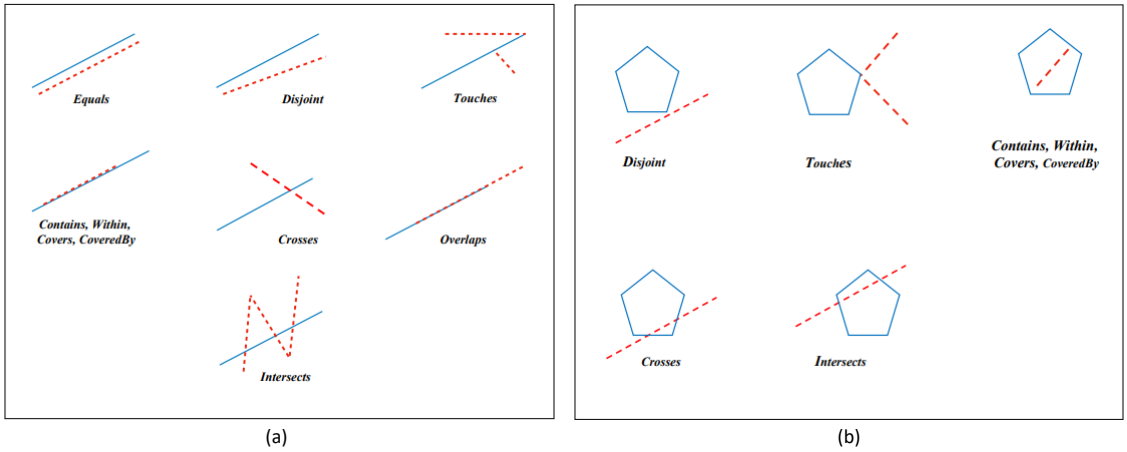

Figure 3.1: Examples of the DE-9IM topological relations between (a) pairs of LineStrings, and (b) LineStrings and Polygons [SDSN17].

Yet, JedAI-spatial is based on a closed-world assumption: the lack of the relation `intersects` between two geometries implies that they satisfy the relation `disjoint`.

Following [PMMK21, ASN18], JedAI-spatial considers *Holistic Geospatial Interlinking*, which simultaneously computes all **positive** topological relations (i.e., all DE9IM relations except `disjoint`). In practice, for each pair of geometries it computes the Intersection Matrix, from which all relations can be extracted with simple boolean expressions. We didn't follow the approach of Silk, LIMES and stLD, where every run computes an individual topological relation, because they repeat the entire processing to the same data multiple times in order to produce all topological links.

Overall, Geospatial Interlinking is formally defined as:

Problem 1 (Geospatial Interlinking)**.** *Given a source and a target dataset, S and T, together with the set of positive topological relations R, compute the set of links $L_R = \{(s, r, t) \subseteq S \times T \times R : r(s, t)\}$ from the Intersection Matrix of all related geometry pairs.*

Given that all batch Geospatial Interlinking algorithms produce an exact solution, yielding the same links, their performance is exclusively assessed with respect to time efficiency: the lower their running time is, the better.

Challenges & Solutions. The main challenge for Geospatial Interlinking is its quadratic time complexity, $O(n^2)$. It essentially requires that every possible pair of geometries should be analytically processed to examine whether a topological relation is satisfied. As a result, this task does not scale to the large volumes of data that lie at the focus of ExtremeEarth.

This is especially true, when considering the high complexity of examining a topological relation for a single pair of entity descriptions. The geometries are typically complex, thus turning the comparison of their interior, boundary and exterior into a rather time-consuming process.

To address the high time complexity of both tasks, we consider three approaches for increasing the time efficiency and scalability of Geospatial Interlinking:

- *Filtering* is the task of grouping together geometries that are highly likely to satisfy a particular relation. These are geometries with intersecting Minimum Bounding Rectangles (**MBR**s). To efficiently identify them, Filtering performs Space Tiling, dividing the Earth's surface into a set of rectangles, i.e., tiles, of the same dimensions. As an example, consider the gray dashed

[1] `https://en.wikipedia.org/wiki/DE-9IM`

rectangles in Figure 1.1. Every geometry is then placed into the tiles that intersect its MBR, i.e., the dotted gray rectangle in Figure 1.1. By considering only the pairs of geometries that co-occur inside every tile, we significantly restrict the search space, avoiding the brute-force approach of examining all possible pairs. We do not introduce any novel filtering methods, but we combine the state-of-the-art existing methods with the following two features for the first time in the literature.

- *Massive parallelization* on top of Apache Spark[2]. Our goal is to exploit the high computational power of the Hops platform by distributing all computations to the available nodes. This pertains not only to Filtering, but to the entire end-to-end workflow of Geospatial Interlinking. Several frameworks in the literature focus on massive parallelization over Apache Spark, such as Apache Sedona[3] (a.k.a., GeoSpark) and Magellan[4]. We have integrated all of them into JedAI-spatial through a common three-step procedure. We perform an analytical scalability analysis, demonstrating that our parallelization approach outperforms the main relevant techniques in the literature to a significant extent.
- *Progressive functionality* aims to schedule the processing of the input data such that the results are produced in a pay-as-you-go fashion. Even though the final outcome is identical with a batch process, the progressive process produces significantly more verified relations at any early point in time. Thus, progressive methods are ideal when the computational resources are available for a limited period of time, as is the case with the Hops platform. They are also ideal for tasks that can operate with partial results. Note that no other tool in the literature offers progressive techniques for Geospatial Interlinking. See below for a formal definition of this task.

3.1 Progressive Geospatial Interlinking

An approximate solution to Geospatial Interlinking is provided by progressive algorithms, which run for a limited time or number of calculations. Compared to batch algorithms, their goal is twofold [PMMK21]:

1. they should produce the same results if they process the entire input data, and
2. they should detect a significantly larger number of related geometry pairs, if their operation is terminated earlier.

These requirements are reflected in Figure 3.2, where the horizontal axis corresponds to the number of examined pairs and the vertical one to the number of related pairs. Essentially, the progressive algorithms should define a processing order that examines the related pairs before the non-related ones, unlike batch algorithms, which examine pairs in an arbitrary order. Hence, the progressive algorithms should maximize the area under their curve, an evaluation measure called **Progressive Geometry Recall** that is defined in $[0, 1]$, with higher values indicating higher effectiveness.

Given a budget BU on the maximum calculations or running time, progressive algorithms tackle the following task [PMMK21]:

Problem 2 (Progressive Geospatial Interlinking)**.** *Given a source and a target dataset, S and T, the positive topological relations R and a budget BU, maximize Progressive Geometry Recall within BU.*

In addition to Progressive Geometry Recall, progressive algorithms are evaluated with respect to the following three measures, too:

[2] `https://spark.apache.org`
[3] `https://sedona.apache.org`
[4] `https://github.com/harsha2010/magellan`

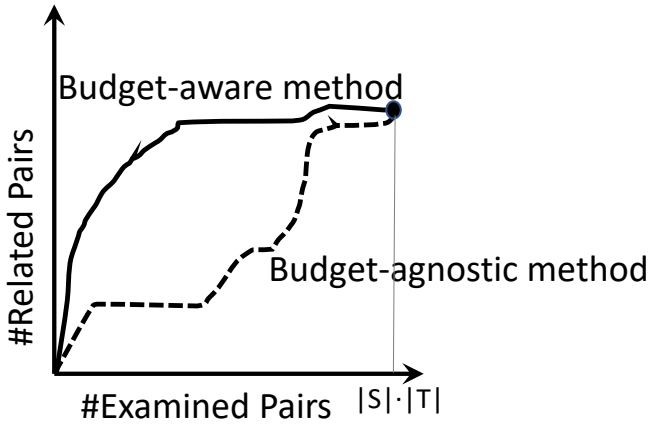

Figure 3.2: The Progressive Geometry Recall of budget-agnostic (batch) and budget-aware (progressive) algorithms.

1. run-time,
2. *precision*, i.e., the number of detected related pairs divided by the number of examined pairs, and
3. *recall*, i.e., the number of detected related pairs divided by the maximum number of related pairs that would be found within BU examinations in the optimal case – after placing all related pairs before the non-related ones.

4. System Architecture

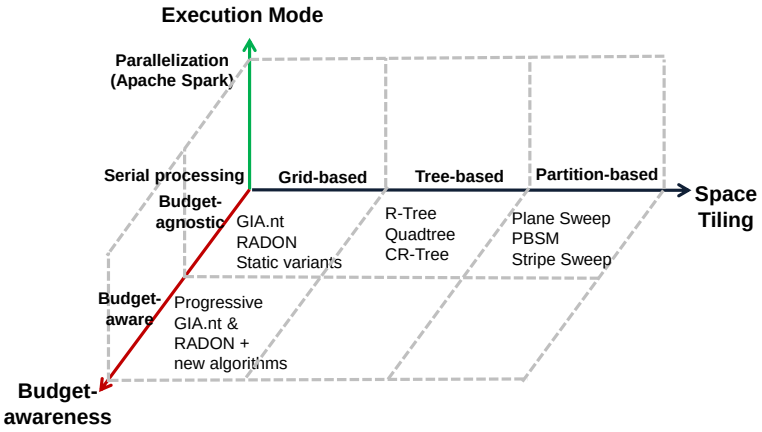

Figure 4.1: The solution space of Geospatial Interlinking algorithms that can be constructed by JedAI-spatial.

JedAI-spatial organizes the Geospatial Interlinking algorithms into a novel taxonomy formed by three dimensions, as shown in Figure 4.1:

1. *Space Tiling* distinguishes the Geospatial Interlinking algorithms into *grid-*, *tree-* and *partition-based* ones. The first category includes Semantic Web techniques that define a static or dynamic Equigrid, the second one encompasses main-memory spatial join techniques from the database community, and the third one conveys variations of plane sweep, a cornerstone algorithm of computational geometry.
2. *Budget-awareness* categorizes algorithms into *budget-agnostic* and *budget-aware* ones. The former are executed in a batch manner that processes the input data in no particular order, producing results only upon completion of the entire process. Budget-aware algorithms are suitable for applications with limited computational or temporal resources, producing results progressively, in a pay-as-you-go manner.
3. *Execution mode* distinguishes between serialized algorithms, which run on a single CPU core, and massively parallel ones, which run on top of Apache Spark.

JedAI-spatial creates any end-to-end pipeline that is defined by these three dimensions. This is achieved by the model-view-controller architecture in Figure 4.2: JS-gui offers two interfaces for user interaction (view), JS-core conveys numerous algorithms and pipelines (controller), and the Data Model component provides the data structures that lie at its core (model).

This architecture serves the following goals:

- *Broad data coverage.* Through its *Data Reading* component, JedAI-spatial supports the most popular structured and semi-structured formats that are used for encoding geometries: Well Known Text (WKT) files, CSV and TSV files, GeoJSON files, RDF dumps, JsonRDF as well as SPARQL endpoints. In this way, JedAI-spatial is able to interlink heterogeneous datasets, e.g., WKT data with GeoJSON files. Special care has been taken to detect and ignore corrupted data as well as to remove noise, which is quite common in automatically or user-generated data sets, especially as their size grows.
- *Broad algorithmic coverage.* JedAI-spatial serves as a library of the state-of-the-art algorithms in the literature, even if they haven't been applied directly to Geospatial Interlinking before. These algorithms are implemented by JedAI-spatial-core (cf. Chapter 5).

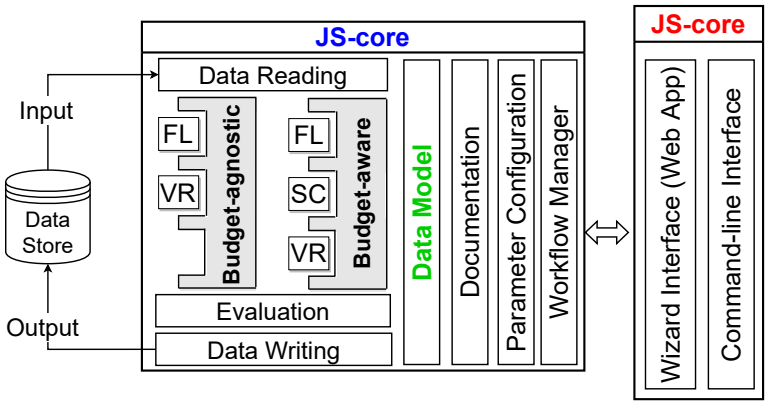

Figure 4.2: JedAI-spatial's model-view-controller architecture.

- *Broad application coverage.* JedAI-spatial accomodates both academic and commercial applications, as its code is released under Apache License V2.0. It also supports both batch and progressive applications. For the latter, the available run-time and/or computations are limited, e.g., cloud-based applications with a limited budget for AWS Lambda functions[1], which charge whenever they are called. In any type of applications, it is crucial to detect the most suitable algorithm for the data at hand (e.g., different batch algorithms might be faster in a LineString-to-LineString scenario and in a Polygon-to-LineString one). To cover this need, JedAI-spatial's benchmarking functionality evaluates easily the relative performance of a large variety of pipelines.

- *High usability.* JedAI-spatial supports both novice and expert users. The former can apply complex, high performing pipelines to their data simply by choosing among the available algorithms, without any knowledge about their internal functionality or their configuration. See Chapter 6 for more details. Power users can use JedAI-spatial as a library or a Maven dependency, can manually fine-tune the selected methods and can extend it with more algorithms or pipelines according to their needs.

- *Extensibility.* Every algorithm in JedAI-spatial implements the interface of its pipeline, which determines its input and output. Hence, new methods can be seamlessly integrated into JedAI-spatial as long as they implement the corresponding interface so that they are treated like the existing ones. Similarly, new pipelines be added as long as they define a new interface that defines their input and output. All additions should implement the `IDocumentation` interface (see Chapter 6 for more details).

- *Efficiency and scalability.* JedAI-spatial scales well to large datasets both in commodity/standalone systems and in computer clusters that run Apache Spark. For more details, refer to the implementation improvements in Chapter 5 and the experiments in Chapter 7.

[1] `https://aws.amazon.com/lambda/`

5. Back-end: JS-core

All methods have been re-implemented in JedAI-spatial's common framework, thus minimizing the dependencies to other systems and libraries. For most algorithms, we have incorporated improvements that significantly enhance their original performance.

5.1 Serial Algorithms

Two are the main types of serial algorithms: the batch ones, which are described in Section 5.1.1, and the progressive ones, which are presented in Section 5.1.2.

5.1.1 Budget-agnostic algorithms

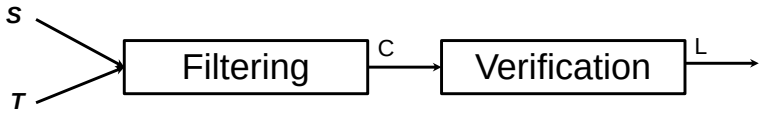

Figure 5.1: The pipeline of budget-agnostic algorithms.

The methods of this category address Problem 1. To compute all positive topological relations between the source and the target geometries, S and T, respectively, they follow the two-step pipeline in Figure 5.1: initially, the Filtering step indexes the source dataset and, if necessary, the target one, based on the minimum bounding rectangle (MBR) of each geometry – in Figure 1.1, the MBRs are the dotted rectangles surrounding each geometry. The resulting index is used to generate C, the set of candidate pairs, which are likely to satisfy at least one topological relation. Next, the Verification step examines every pair in C as long as their MBRs are intersecting. The detected topological relations are added to the set of triples L, which is returned as output.

JedAI-spatial organizes these algorithms into the following three subcategories, based on the type of the index used in Filtering.

Grid-based Algorithms. The input geometries are indexed by dividing the Earth's surface into cells of the same dimensions. The index is called *Equigrid* and its cells *tiles*. Every geometry is placed into all tiles that intersect its MBR. JedAI-spatial conveys four state-of-the-art algorithms of this type, which differ in the definition and use of the Equigrid during Filtering and Verification.

RADON [SDSN17]. Filtering loads both input datasets into main memory and defines an Equigrid index by setting the horizontal and vertical dimensions of its tiles equal to the average width and height, respectively, over all geometries. Verification computes the Intersection Matrix for all candidate pairs [ASN18], taking special care to avoid the ones repeated across different tiles of the Equigrid.

GIA.nt [PMMK21]. Filtering loads into main memory only the input dataset with the fewest geometries. The granularity of the Equigrid index is determined by the average dimensions of this dataset. Verification reads the geometries of the other dataset from the disk, one by one; for each geometry g, it gathers the candidates, whose MBR intersects the same tiles as $MBR(g)$ as well as $MBR(g)$ itself, and computes their Intersection Matrix, adding the detected links to L.

Static variants. Unlike the dynamic Equigrid of the above algorithms, whose granularity depends on the input data, *Silk-spatial* [SK16] employs a static Equigrid, whose granularity is predetermined

(by the user), independently of the input characteristics. Even though the resulting index might be too fine- or coarse-grained for the input datasets, this approach is based on the idea that the resulting candidate pairs are filtered during Verification, which discards those with disjoint MBRs. To put this approach into practice, JedAI-spatial includes the custom methods *Static RADON* and *Static GIA.nt*, where the index granularity is manually defined.

Implementation improvements. RADON's implementation is publicly available through LIMES[1]. However, we re-implemented it in JedAI-spatial so as to significantly improve its performance. First, we reduce the run-time of Filtering by skipping the swapping strategy, which is used to identify the input dataset with the smallest overall volume. This has no impact on its Filtering, given that the Equigrid granularity considers both input datasets. Second, we reduce RADON's memory footprint to a significant extent. Instead of a hashmap that stores all examined pairs in main memory to avoid verifying the same candidate pairs more than once, we use the **reference point technique** [DS00], verifying every candidate pair only in the tile that contains the top left corner of their intersection; as an example, consider the geometries g_3 and g_4 in Figure 1.1: they co-occur in tiles (4,A), (4,B), (5,A), (5,B), but are verified only in (5,A), where their reference point (black dot) lies. Moreover, unlike the original implementation, which refers to all geometries by their URL (of type `String`), we use ids for this purpose (of type `int`). We also use the data structures of the GNU Trove library [FE13], which work with primitive data types (e.g., the 4-bytes `int` instead of the 16-bytes `Integer`). The same improvements apply to the methods corresponding to Silk-spatial, namely the static variants. For GIA.nt, we use the open-source implementation provided by the authors of [PMMK21], which already involves these optimizations[2].

Partition-based Algorithms. They rely on a (usually vertical) sweepline that moves across the Earth's surface, stopping at some points. Filtering sorts all input geometries in ascending order of their lower boundary on the horizontal axis, x_{min}. Verification is restricted to pairs of source and target geometries whose MBRs simultaneously intersect the sweepline whenever it stops. The process terminates once the sweepline passes over all geometries.

Plane Sweep [BKS93]. This cornerstone algorithm applies the above process to all source and target geometries. Before verifying a pair of geometries, it ensures that they overlap on the y-axis.

PBSM [PD96]. This algorithm splits the given geometries into a manually defined number of orthogonal partitions and applies Plane Sweep inside every partition. Filtering defines the partitions, assigns every geometry to all partitions that intersect its MBR and sorts all geometries per partition in ascending x_{min}. Verification goes through the partitions and in each of them, it sweeps a vertical line l, computing the Intersection Matrix for each pair of geometries that simultaneously intersect l and overlap on the y-axis. To avoid repeated verifications of the same geometry pairs across different partitions, it uses the *reference point technique* [DS00].

Stripe Sweep. To lower the time complexity of Plane Sweep, this new algorithm sorts only the source geometries during Filtering. These geometries are then partitioned into several vertical stripes, whose length is equal to their average width. Every source geometry is placed in all stripes that intersect its MBR. Verification probes every target geometry t against the stripes and aggregates the set of the source geometries that intersect the same stripes with t; this way, it gathers the distinct candidate geometries, avoiding redundant verifications. This set is further refined by retaining only the candidate pairs with intersecting MBRs.

Implementation improvements. Plane Sweep and PBSM employ a dynamic data structure, called *sweep structure*, which stores in main memory the active geometries, i.e., the ones whose MBR intersects the sweep-line in its current position. JedAI-spatial supports two different sweep structures: (i) *List Sweep* maintains one linked list for each input dataset. In every move of the sweepline l, the contents of both lists are updated, inserting the geometries with an intersecting MBR and removing the expired ones, i.e., the geometries with $x_{max} < l_x$. (ii) *Striped Sweep* splits

[1] `https://github.com/dice-group/LIMES`
[2] `https://github.com/giantInterlinking/prGIAnt`

the given datasets into n stripes and uses a different List Sweep per stripe. After preliminary experiments, the length of each stripe on the horizontal axis was set to the average width of the source geometries.

Stripe Sweep can use two different data structures for storing the source geometries per stripe: (i) a hash map, which associates every stripe id with the corresponding source geometry ids, and (ii) an STR-Tree [LEL97], which indexes the source geometries in each stripe. The hash map does not ensure the overlap on the y-axis before checking the MBR intersection of candidate pairs, unlike the STR-Tree. For the implementation of the STR-Tree, we use the optimized implementation provided by the JTS library.[3]

Tree-based Algorithms. As suggested by their name, these algorithms rely on state-of-the-art spatial tree indices. During Filtering, they index the smallest input dataset. During Verification, every geometry g from the other dataset queries the tree index; its candidates are located in the leaf nodes whose MBR intersects with $MBR(g)$. For all candidate geometries with an MBR that intersects $MBR(g)$, the Intersection Matrix is computed.

R-Tree [Gut84]. In this index, every non-leaf node contains pointers to its child nodes along with an MBR that encloses the span of all the MBRs in its children; every leaf node contains up to M geometries. When an entry is added to a full node, the node is split into two new ones, which are initialized with the two largest geometries. The remaining geometries are added to the node whose MBR expands the least after insertion.

Quadtree [FB74]. In this index, every non-leaf node has exactly four children, dividing the space into four quadrants: NorthEast (NE), NorthWest (NW), SouthEast (SE) and SouthWest (SW). Again, every node has a maximum capacity M. When M is reached, the corresponding cell is split into four new ones, its children.

CR-Tree [KCK01]. This index compresses the R-Tree so that it leverages the L1 and L2 cache memory of CPUs, which have faster access times. Using the *Quantized Relative Representation of MBR*, it minimizes the size of the MBRs, which dominate the space requirements. CR-Trees are usually wider and smaller than R-Trees, achieving higher time efficiency, while occupying ∼60% less memory.

Implementation improvements. For R-Tree and CR-Tree, we employ the data structures of GNU Trove, which work with primitive data types, offering high time efficiency and low memory footprint. For Quadtree, we use the optimized implementation of the JTS library.

5.1.2 Budget-aware algorithms

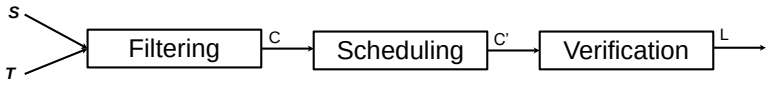

Figure 5.2: The pipeline of budget-aware algorithms.

This category encompasses methods that address Problem 2. Their goal is to maximize the number of related geometry pairs that are detected after consuming the available budget BU, which determines the maximum number of verifications. To this end, they follow the three-step pipeline in Figure 5.2. Filtering is identical with that of budget-agnostic methods, producing a set of candidate pairs C. Scheduling first refines C by discarding the pairs with non-overlapping MBRs. Then, it defines the processing order of the remaining pairs so that the likely related ones are placed before the unlikely ones. The new set of candidate pairs C' is forwarded to Verification, which carries out their processing and returns the set of detected links, L.

[3] `https://locationtech.github.io/jts/`

The gist of budget-aware algorithms is the combination of Scheduling with Filtering, as Verification remains the same in all cases. Based on the co-occurrence patterns of grid-based Filtering, Scheduling assigns a score to every pair of candidates with intersecting MBRs. The higher this score is, the more likely the constituent geometries are to satisfy at least one topological relation. JedAI-spatial offers the following weighting schemes [PMMK21]:

- *Co-occurrence Frequency* (CF) measures how many tiles simultaneously intersect the MBRs of the source and the target geometry.
- *Jaccard Similarity* (JS) normalizes CF by the number of tiles intersecting each geometry.
- *Pearson's χ^2 test* extends CF by assessing whether the given geometries appear independently in the tiles.
- *Minimum Bounding Rectangle Overlap* (MBRO) returns the normalized overlap of the MBRs of the two geometries.
- *Inverse Sum of Points* (ISP) amounts to the inverse sum of boundary points in the two geometries, thus promoting the simpler candidate pairs.
- *Composite schemes.* They combine two of the aforementioned, atomic schemes in the following way: the *primary* one is used for scheduling all pairs, while the *secondary* one is used for resolving the ties.

These weighting schemes are leveraged by the following algorithms:

Progressive GIA.nt [PMMK21]. It applies the same Filtering as its budget-agnostic counterpart and, then, its Scheduling retains the top-BU weighted valid candidate pairs.

Progressive RADON [PMMK21]. It applies RADON's Filtering and defines the processing order of the resulting tiles by sorting them in increasing or decreasing number of candidate pairs. Inside every tile, it identifies the non-redundant candidate pairs using the reference point technique. Those with intersecting MBRs, are processed in decreasing score, as determined by the selected weighting scheme.

Iterative Algorithm. To ensure that every source or target geometry is represented in the BU retained candidate pairs, this new algorithm applies the same Filtering as GIA.nt and its Scheduling retains $\lceil BU/|S| \rceil$ or $\lceil BU/|T| \rceil$ candidates per source or target geometry, respectively. The top-weighted BU candidates are then forwarded to Verification.

Geometry-ordered Algorithm. This is another new progressive algorithm. It assumes that the larger the average weight of a geometry is, the more likely it is to be related to its candidates. Thus, it applies the same Filtering as GIA.nt and then, it estimates the average weight per target or source geometry. After sorting the geometries in decreasing average weight, it iterates once more over the target dataset to select the BU retained pairs from the candidates of the top-weighted geometries. Optionally, the retained candidates can be sorted in decreasing weight.

Dynamic Progressive GIA.nt [PMMKar]. The core idea of this approach is that whenever a new pair of geometries (s, t) is detected as qualifying, we boost the weight of all candidate pairs that are associated with s and t and are still located in the priority queue so that they are verified earlier. This is useful for example in cases where the source dataset involves long LineString geometries like roads, whereas the target dataset involves Polygon geometries buildings: the more buildings a road touched so far, the higher should be the weight of the rest of the candidate buildings, as it is likely a main road.

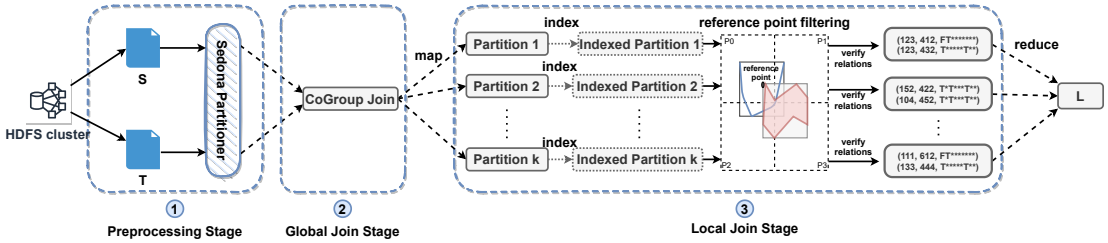

Figure 5.3: The three-step pipeline of parallel, budget-agnostic algorithms in JedAI-spatial.

5.1.3 Parallel Algorithms

To scale to voluminous datasets, JedAI-spatial exploits the massive parallelization functionalities offered by the state-of-the-art framework of Apache Spark. JedAI-spatial has aggregated all relevant algorithms in the literature that are crafted for the same parallelization framework and the same types of geometries, i.e., LineStrings and Polygons [PKNK18] (we exclude SIMBA [XLY+16], which exclusively applies to points).

To integrate the main parallel algorithms into JedAI-spatial, we adapted their functionality so that they implement the three-step pipeline that is shown in Figure 5.3:

1. The *Preprocessing Stage* reads the source and target datasets from HDFS, transforms them into Spark RDDs and splits each of them into partitions according to a predetermined approach (e.g., QuadTree).
2. The *Global Join Stage* joins the source and target partitions with a spatial overlap and assigns every pair of overlapping partitions to a different worker for processing.
3. In the *Local Join Stage*, each worker interlinks every pair of overlapping source and target partitions.

Below, we explain how the main parallel interlinking algorithms are adapted to this three-step pipeline.

Budget-agnostic algorithms. JedAI-spatial conveys the following four state-of-the-art parallel approaches:

GeoSpark [YWS15]. This algorithm is now part of *Apache Sedona*[4]. During Preprocessing, it uses sampling to partition the input data with a KDB-Tree or a Quadtree. The geometries that are not covered by the index are added to an overflow partition. The overlapping source and target partitions are assigned to the workers, during the Global Join Stage. The Local Join Stage verifies the candidate pairs through a nested loop join or indexes the source geometries with an R-Tree or a Quadtree that is queried by the target ones.

Spatial Spark [YZG15]. This algorithm supports two functionalities:

1. The *broadcast join* supports up to 2GB of source data. During the Preprocessing Stage, the source dataset is indexed by an R-Tree, which is then broadcast as a read-only variable to all workers. Every worker also receives a disjoint partition of the target geometries. The Global Join Stage is skipped. During the Local one, every worker iterates over its target geometries, retrieves the candidate source ones from the R-Tree and verifies those intersecting the target MBR.

[4] `http://sedona.apache.org`

2. The *partition join* overcomes the size limit of the broadcast join by implementing all three steps of JedAI-spatial's framework. Depending on the user's choice, the Preprocessing Stage indexes the entire input data (in case of a Fixed Grid Partition, whose dimensions, $dim_X \times dim_Y$ are defined by the user) or a sample of the source and target data (in case of Binary Split or Sort Tile Partitions, which use an R-Tree). The Global Join Stage assigns the overlapping source and target partitions to the same worker so that their candidate pairs are verified locally, during the third stage.

Magellan.[5] This algorithm relies on the Z-Order Curves, which define an Equigrid on the Earth's surface during the Preprocessing Stage. The number of tiles in this grid is determined as 2^p, where p is the precision parameter that is set by the user. Apparently, the higher the precision is, the more fine-grained is the resulting Equigrid index. The Global Join Index sends to the same workers the source and target geometries that intersect the same tiles. During the Local Join Index, every worker checks every candidate pair and verifies those with intersecting MBRs.

Location Spark [TYM+16, TYM+20]. Its Preprocessing partitions the source and target datasets using a Grid, R-Tree or Quadtree index. Then, its Query Plan Scheduler performs a skew analysis in order to partition the data as evenly as possible, balancing the workload among the workers. In essence, it repartitions the skewed partitions, which include at least twice as many geometries as the smallest one. After joining the overlapping source and target partitions during the Global Join Stage, a local index is constructed for the source geometries of every worker using an R-Tree, QuadTee or EquiGrid. The Local Join Stage queries the index with the target geometries and verifies the candidate pairs with intersecting MBRs.

Parallel GIA.nt [PMMK21]. The Preprocessing estimates the average width and height of the source geometries. These dimensions, which are broadcast to all workers, define the Equigrid that partitions both input datasets. The next stage joins the overlapping source and target partitions, while the Local Join Stage creates an Equigrid of the source geometries inside every worker, using the broadcast dimensions. The target geometries query the index to retrieve the candidates with intersecting MBRs, which are then verified. The reference point technique eliminates all repeated verifications.

Implementation Improvements. The most important enhancement to all parallel algorithms is the use of the reference point technique during the Local Join Stage in order to avoid all repeated verifications. This has replaced GeoSpark's `groupBy`, Spatial Spark's call to `distinct`, Magellan's `dropDuplicates` and Location Spark's call to `reduceByKey`. All these functions shuffle the output data along the cluster, imposing significant overhead.

We also replaced Magellan's Extended Spark SQL with Spark RDDs for higher efficiency. For Location Spark, we removed the Spatial Bloom Filter; it is mainly used for spatial range queries, but in our case, where every geometry is assigned to multiple partitions, it imposes an unnecessary overhead. From GeoSpark, we removed the R-Tree from the indexes supported by the Global Join Stage: due to sampling, its overflow bucket typically contains geometries from the entire input datasets and, thus, it is not disjoint from the rest of the partitions; as a result, the reference point technique cannot be used to eliminate redudant verifications. For Parallel GIA.nt, we employ the publicly available code that was used in [PMMK21].

Budget-aware algorithms. JedAI-spatial parallelizes all serial budget-aware algorithms described in Section 5.1.2. The Preprocessing and Global Join Stage are identical with Parallel GIA.nt. Then, the overall budget BU is split among the partitions assigned to every worker based on the portion of candidate pairs it involves. The Local Join Stage applies the budget-aware algorithm to the data assigned to every worker, using the corresponding local budget.

[5] `https://github.com/harsha2010/magellan`

6. Front-end: JS-gui

JedAI-spatial support users of any experience level, offering two wizard-like user interfaces that simplify its use to a great extent:

1. The command line interface. JS-core produces an executable jar, which when run, guides users in applying the desired pipeline to their data.
2. The Web application interface. JS-gui is available as a Docker image, which, when deployed, runs a Web application that seamlessly supports both serial and parallel execution (the latter relies on *Apache Livy*[1]).

It both cases, users do not need to write code in order to interlink their spatial data. They are merely required to provide the following input:

1. the paths of their datasets,
2. reading parameters for the dataset files (e.g., the separator character in CSV files),
3. the desired pipeline, i.e., serial or parallel, budget-aware or budget-agnostic, and
4. the desired algorithm among the available ones for the selected pipeline.

This applies even to the parallel pipelines that run on top of Apache Spark. In both interfaces, users can also inspect the input data and store the detected links to a specific path.

Special care has also been taken to facilitate the comparison between the available algorithms, regardless of their space tiling and budget-awareness category. JedAI-spatial acts as a workbench, encompassing a special menu that summarizes the performance of the latest runs with respect to the effectiveness measures (i.e., recall, precision, f-measure and progressive geometry recall, if applicable) as well as the efficiency ones (i.e., the run-time in total and in every step of the selected pipeline). This workbench functionality, which is shown in Figure 6.1, also allows for examining the impact of configuration parameters on the performance of a particular algorithm (e.g., by changing the granularity of the grid index).

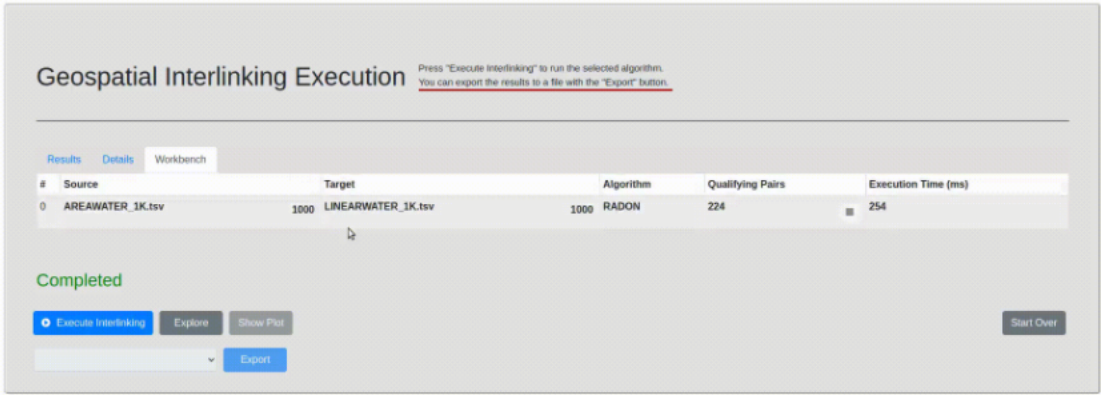

Figure 6.1: The workbench window of JedAI-spatial's Graphical User Interface.

Auxiliary Components. In the following, we briefly describe the rest of the components in Figure 4.2, which play an important role in the characteristics offered by JedAI-spatial's user interfaces.

[1] `https://livy.apache.org`

Data Model. This component implements the classes and the data structures that lie at the core of JedAI-spatial. The cornerstone is the `GeometryProfile` class, which supports all heterogeneous data formats mentioned in Section 4. This is accomplished by representing every geometry as a set of name-value pairs, which capture the textual information about an entity, coupled with a `Geometry` object of the JTS library, which is accompanied by its MBR and the method for computing an Intersection Matrix. Note that the simple, yet versatile `GeometryProfile` class facilitates the visualization and inspection of input data through JS-gui.

Documentation. This component essentially corresponds to a Java interface that is implemented by all algorithms. The interface conveys methods providing textual information about the most important aspects of each algorithm: its name, a summary of its functionality, the name of every configuration parameter, a short description of every parameter, the domain of every parameter (i.e., its default, minimum and maximum values) as well as the configuration of the current algorithm instantiation. This information is provided to the user through tooltips in the Web application and through the help option of the command line interface.

Parameter-configuration. JedAI-spatial facilitates the fine-tuning of any supported algorithm, as a poor parameterization invariably leads to poor performance. To this end, this component supports three modes:

1. *Default configuration* a-priori sets all parameters of each algorithm to values that empirically achieve reasonable performance across different datasets. This mode allows lay users to apply the desired pipeline to their data simply by choosing among the available methods.
2. *Manual configuration* enables power users to fine-tune an algorithm themselves, based on their own experience or on the information provided by the `Documentation` component.
3. *Grid search* automatically identifies the optimal configuration through a brute-force approach that tries all values in the domain of each parameter. In the case of budget-agnostic pipelines, the parameterization that minimizes the run-time is selected as the optimal one, while for budget-aware pipelines, the optimal parameterization is the one maximizing Progressive Geometry Recall.

Table 7.1: Technical characteristics of the real pairs of datasets for Geospatial Interlinking.

	$\mathbf{D_1}$	$\mathbf{D_2}$	$\mathbf{D_3}$	$\mathbf{D_4}$	$\mathbf{D_5}$
Source Dataset	AREAWATER	AREAWATER	Lakes	Parks	Roads
Target Dataset	LINEARWATER	ROADS	Parks	Roads	Buildings
#Source Geometries	2,292,766	2,292,766	8,326,942	9,831,432	72,339,926
#Target Geometries	5,838,339	19,592,688	9,831,432	72,339,926	114,796,567
Cartesian Product	$1.34 \cdot 10^{13}$	$4.49 \cdot 10^{13}$	$8.19 \cdot 10^{13}$	$7.11 \cdot 10^{14}$	$8.30 \cdot 10^{15}$
Candidate Pairs	6,310,640	15,729,319	19,595,036	67,336,808	257,075,645
#Qualifying Pairs	2,401,396	199,122	3,841,922	12,145,630	1,041,562
#`Contains`	806,158	3,792	267,457	5,147,704	274,953
#`CoveredBy`	0	0	1,944,207	47,253	82,828
#`Covers`	832,843	4,692	267,713	5,284,672	274,966
#`Crosses`	40,489	106,823	217,198	5,700,257	313,566
#`Equals`	0	0	61,712	2,047	18,909
#`Intersects`	2,401,396	199,122	3,841,922	12,145,630	1,037,153
#`Overlaps`	0	0	488,814	42,331	54,810
#`Touches`	1,554,749	88,507	986,522	1,210,230	331,166
#`Within`	0	0	1,943,643	47,155	81,567
Total Relations	5,635,635	402,936	10,019,188	29,627,279	2,481,027

7. Comparative Analysis

7.1 Experimental Setup

All serial methods and experiments were implemented in Java 8. The experiments were ran on a server with Intel Xeon E5-4603 v2 @ 2.2GHz, 128 GB RAM, running Ubuntu 14.04.5 LTS. For all time measurements, we used a single physical core and performed three repetitions, reporting the average. For the verification of geometry pairs, we used the JTS Topology Suite, version 1.16.[1]

All parallel methods and experiments were implemented in Scala 2.12 using Spark 2.4.4. Most of the experiments were performed on a Hadoop cluster consisting of a single node with 32 cores Intel(R) Xeon(R) CPU E5-4603 v2 @ 2.20GHz[2] and 128GB DDR3 RAM, 1.6 Tb mechanical disk. Unless stated otherwise, the experiments performed on the single node used 16 Executors with 2 cores each and 7GB of memory, and the experiments performed on the two nodes, used 15 Executors with 4 cores each and 10GB of memory. For each time measurement, we performed 5 repetitions and took the average run-time.

Datasets. The technical characteristics of the real datasets we use in our experiments are reported in Table 8.2. All of them have been widely used in the literature [TBMT19, EM15] and are publicly available.[3] They contain public data about area hydrography (AREAWATER), linear hydrography (LINEARWATER), roads (ROADS) and all edges (EDGES) in USA. They also contain the boundaries of all lakes (Lakes), parks or green areas (Parks), roads and streets (Roads) as well as of all buildings (Buildings) around the world. Each column of Table 8.2 shows statistics for a pair (D_1–D_5) of interlinked datasets. Note that in D_1, D_2 and D_4, the source geometries are Polygons and the target ones LineStrings, and vice versa for D_5. In contrast, D_3 is homogeneous, as it exclusively pertains to Polygons.

[1] `https://github.com/locationtech/jts`
[2] The system uses hyper-threading. Hence, it has 16 physical cores.
[3] `http://spatialhadoop.cs.umn.edu/datasets.html`

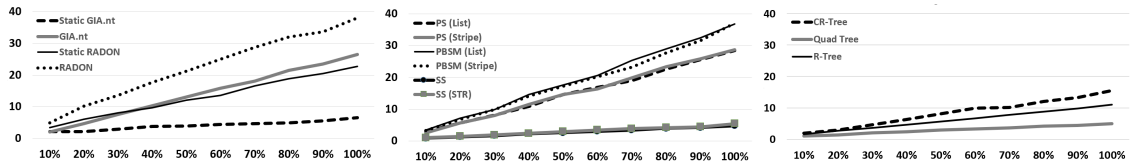

Figure 7.1: Scalability analysis of the serial budget-agnostic algorithms with respect to their Filtering time (in seconds).

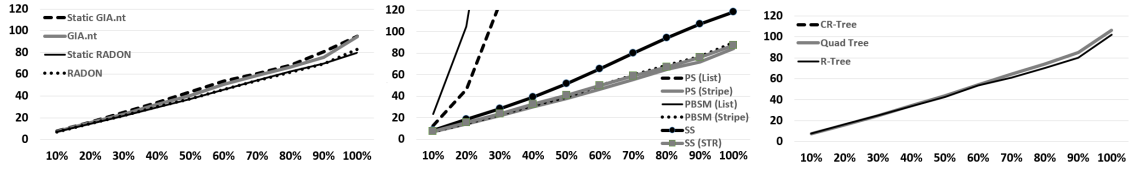

Figure 7.2: Scalability analysis of the serial budget-agnostic algorithms with respect to their Verification time (in minutes).

7.2 Budget-agnostic Algorithms

In this analysis, we assess the scalability of each method over the AREAWATER-LINEARWATER dataset from the US Census Bureau TIGER files [EM15, TBMT19, PMMK21], which includes USA's Area and Linear Hydrography – D_1 in Table 8.2. We split this dataset into 10 subsets of increasing size, from 10% of source and target geometries to 100% with a step of 10%. Special care was taken to ensure that the number of related pairs increases in proportion to the size of the input data. We examine the relative performance of JedAI-spatial's budget-agnostic algorithms. Given that these methods produce the same results, we assess their relative performance with respect to their efficiency, estimating the time required to complete each step in the pipeline of Figure 5.1: (i) the *filtering time*, $\mathbf{t_f}$, and (ii) the *verification time*, $\mathbf{t_v}$.

The resulting filtering and verification times appear in Figures 7.1 and 7.2, respectively. Each figure encompasses a separate diagram for each algorithm category, but all diagrams use the same scale in order to facilitate the comparisons between the three categories.

Starting with Figure 7.1, we observe that for all algorithms, the Filtering step is completed within a few seconds, even when processing the entire D_1. The reason is that Filtering constitutes a quick process that considers exclusively the MBR of the input geometries, thus disregarding their actual complexity. Yet, it manages to reduce the number of candidates by a whole order of magnitude; e.g., for the entire dataset, it reduces them from $1.34{\cdot}10^{13}$ (Cartesian Product) to 6,310,640 candidate pairs with intersecting MBRs (among them, 2,401,396 pairs of source and target geometries are topologically related, satisfying at least one relation).

As expected, the algorithms that consider only the source dataset when building their index are much faster than those iterating over both input datasets. The former category includes (Static) GIA.nt, Stripe Sweep and the tree-based algorithms. All these algorithms scale sublinearly with the increase of the input data: from 10% to 100%, their t_f raises by 5 (Stripe Sweep) to 8 (CR-Tree) times. The rest of the algorithms scale linearly with the increase of the input data. We also observe that the static variants of the grid-based algorithms are significantly faster, as they save the cost of deriving the index granularity from the characteristics of the input datasets – they merely index them. Finally, we notice that the filtering time of each partition-based algorithm is practically stable, regardless of the underlying data structure (List Sweep or Striped Sweep for Plane Sweep and PBSM, hash map or STR-Tree for Stripe Sweep). Overall, *Quadtree and Stripe Sweep exhibit the fastest Filtering.*

Regarding the Verification time, we notice in Figure 7.2 that it constitutes the bottleneck of Geospatial Interlinking, being two orders of magnitude larger than Filtering time. This is caused by

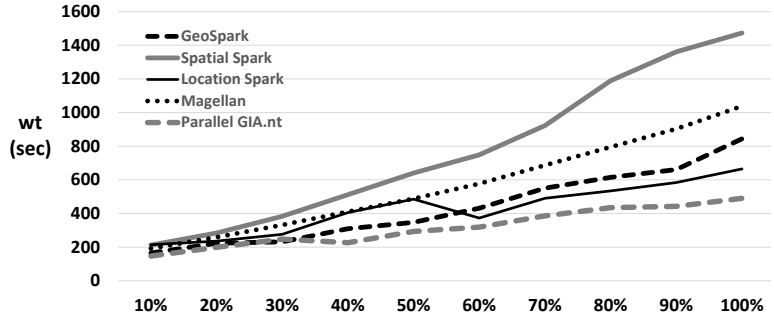

Figure 7.3: Scalability of parallel, budget-agnostic algorithms.

the complexity of the input geometries, which determines the cost of calculating each Intersection Matrix. We also observe that the algorithms with the fast, source-based Filtering are now slower, because they read the target geometries from the disk, one by one. This overhead is included in their t_v, increasing linearly with the size of the input data, hence the larger deviations over larger subsets. *The algorithms (Static) RADON, Plane Sweep and PBSM are faster (in this order)*, because they a-priori load the target geometries into main memory. Note, though, that Plane Sweep and PBSM are by far the slowest algorithms when using a Linked List to maintain their active geometries, due to the high cost of its operations. The performance of both algorithms is significantly improved when using Stripes to reduce the maintenance overhead. The situation is even worse for CR-Tree, which is excluded from Figure 7.2, because its run-time over the smallest subset is 235 minutes, exceeding the time required by most other algorithms even for the largest subset. The reason is the high cost of retrieving the candidates for every target geometry, due to the compression of MBRs. Finally, we should stress that (Static) RADON is faster than (Static) GIA.nt by 5% to 12%, which is in contrast with their relative performance in [PMMK21], due to the significant impact of the implementation improvements we have incorporated in JedAI-spatial. *Yet, GIA.nt involves the fastest verification among the algorithms that read the target geometries from the disk, with Stripe Sweep being slightly slower.*

Figure 7.3 reports the overall wall-clock time (in seconds) of the parallel budget-agnostic algorithms, which are fine-tuned as follows: GeoSpark (Apache Sedona) with KDB-Tree partitioning and local indexing with R-Tree, Spatial Spark with a 512x512 Fixed Grid Partitioning, Location Spark with Quadtree partitioning and local indexing with R-Tree and Magellan with precision 20; for Parallel GIA.nt, we have interchanged the source with the target datasets. *Parallel GIA.nt is consistently the fastest algorithm, with Location Spark following in close distance over the largest subsets*, where its skew analysis bears fruit, *and GeoSpark in the third place.* These three algorithms require less than half the overall run-time of Spatial Spark, with Magellan lying in the middle of these two extremes. All algorithms scale sublinearly with the size of the input data: from 10% to 100%, their run-time increases by 3 (Location Spark, Parallel GIA.nt) to 7 (Spatial Spark) times. All are significantly faster than the serial algorithms, especially over the larger subsets, where the overhead of Apache Spark pays off: for 100%, the slowest parallel algorithm (Spatial Spark) is 3.2 times faster than the best serialized algorithm (RADON).

Evaluating JedAI-Spatial on Hopsworks and CREODIAS. Regarding the experimental evaluation of JedAI-Spatial on the Hopsworks cluster that has been set up in CREODIAS, we have successfully detected all topological relations for the D_5 dataset, which almost 190 million geometries. The original size of the TSV files is 50 GB, which corresponds to almost 400 GB when transformed into RDF Ntriples. For this dataset, we used 25 executors with 6120 MB of memory and 2 virtual cores per executor. The entire processing was completed in less than 12 minutes. The total size of the input datasets and the produced output is almost 1 TB in Ntriples format.

Table 7.2: Performance of (a) Static and (b) Dynamic Progressive GIA.nt over D_5 for all weighting schemes in comparison to the optimal (Opt.) and the random (Rnd.) approach for budgets of 5M and 10M verifications.

	BU = 5M							**BU = 10M**						
	Opt.	Rnd.	Progressive GIA.nt					Opt.	Rnd.	Progressive GIA.nt				
			CF	JS	χ^2	$MBRO$	ISP			CF	JS	χ^2	$MBRO$	ISP
PGR	0.890	0.039	0.025	0.065	0.064	**0.159**	0.014	0.948	0.068	0.040	0.110	0.112	**0.224**	0.028
Recall	1.000	0.072	0.049	0.144	0.141	0.246	0.028	1.000	0.115	0.076	0.227	0.228	0.338	0.060
Precis.	0.207	0.015	0.010	0.030	0.029	0.051	0.006	0.103	0.012	0.008	0.024	0.024	0.035	0.006
t_r (min)	-	-	23.4	23.6	23.3	23.4	23.4	-	-	23.4	22.5	23.5	23.2	23.3
t_s (min)	-	-	17.6	19.0	18.8	17.7	17.7	-	-	18.5	17.1	18.1	18.0	17.6
t_v (min)	-	-	2.3	4.3	1.4	2.5	2.0	-	-	3.5	4.0	3.8	3.0	3.4
t_w (min)	-	-	43.3	46.9	43.5	43.6	43.2	-	-	45.4	43.6	45.4	44.2	44.3

(a) Static Parallel Progressive GIA.nt

PGR	0.890	0.039	0.146	0.158	**0.241**	0.211	0.111	0.948	0.068	0.167	0.182	0.270	**0.295**	0.127
Recall	1.000	0.072	0.182	0.183	0.277	0.323	0.133	1.000	0.115	0.195	0.231	0.324	0.423	0.159
Precis.	0.207	0.015	0.038	0.038	0.057	0.067	0.028	0.103	0.012	0.020	0.024	0.034	0.044	0.016
t_r (min)	-	-	23.4	23.6	23.4	23.4	23.4	-	-	23.3	22.5	23.5	23.2	23.8
t_s (min)	-	-	17.6	18.6	17.8	18.1	18.2	-	-	17.5	19.4	18.4	17.5	17.8
t_v (min)	-	-	3.5	1.5	2.2	3.0	3.1	-	-	4.2	2.5	3.1	4.2	3.4
t_w (min)	-	-	44.5	43.7	43.4	44.5	44.7	-	-	45.0	44.8	45.0	44.9	45.0

(b) Dynamic Parallel Progressive GIA.nt

7.3 Budget-aware Algorithms

Table 7.2 reports the performance of static and dynamic Parallel Progressive GIA.nt over D_5 for two different budgets, BU=5M and BU=10M. In addition to the scheduling ($\mathbf{t_s}$) and verification ($\mathbf{t_v}$) time, we report the *overhead time*, $\mathbf{t_r}$, which is the aggregate time needed for all other computations, such as the initialization of Spark context, the spatial partitioning and the computation of the granularity of the tiles. The *overall wall-clock run-time* ($\mathbf{t_w}$) corresponds to the sum of these three time measurements.

Notice that both scheduling and verification are lower than the overhead time, mostly because of spatial partitioning. Spatial partitioning collects a random sample of the source geometries in order to built the spatial partitioner and then redistributes all the geometries, invoking data shuffling. This procedure remains the same regardless the size of the budget and adds significant overhead to the overall execution. Moreover, the Verification step is faster than the Scheduling step, as most of the geometries in D_5 are small and simple, thus allowing for the quick computation of the intersection matrix. On the contrary, the Scheduling step needs to examine all the candidate pairs that passed the Filtering step, which comprises of millions of geometry pairs.

Most importantly, we observe that both algorithms are capable of processing the ∼190 million geometries of D_5 in around 45 minutes. Both algorithms underperform the optimal approach, but significantly outperform the random ordering with respect to all measures. The recall remains low, due to the small maximum budget, which was determined by the available main memory resources. Larger budgets would allow us to detect more topologically related geometries. Note also that Dynamic Progressive GIA.nt consistently outperforms its static counterpart, producing results twice as good in certain cases. MBRO also proves to be the most suitable weighting scheme.

7.3.1 Scalability Analysis

Figure 7.4 displays the strong scaling experiments of the parallel implementation of the static and dynamic progressive algorithms. In strong scaling, we examine how the overall computational time of the job scales as we increase the number of available processing units. In this experiment, the job is defined by the performance of progressive algorithms in combination with MBRO weights and BU=20M, over the most time-consuming dataset pair, D_4. We observe that both algorithms

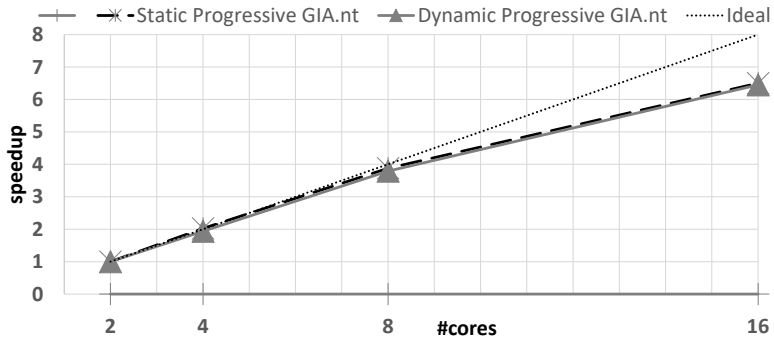

Figure 7.4: The speedup of Parallel Static and Dynamic Progressive GIA.nt as the number of cores increases over D_4, i.e., strong scalability .

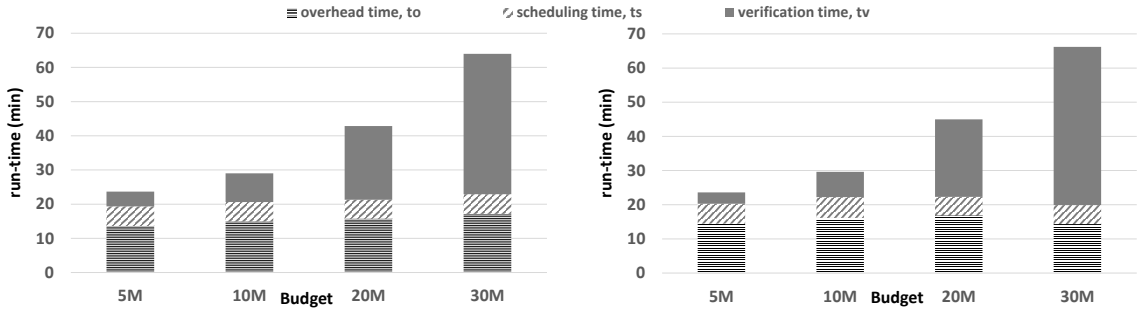

Figure 7.5: Scalability of Static (left) and Dynamic (right) Parallel Progressive GIA.nt over D_4.

scale sub-linearly and close to the ideal speedup, and we can only notice a small deceleration in the case of 16 processing units. This is because both algorithms invoke data shuffling in certain points in order to redistribute the geometries (i.e., spatial partitioning) and to compute the granularity of the tiles. By increasing the number of the processing units, more extensive data shuffling is performed, adding the extra overhead to the execution that leads to a sub-linear speedup. Overall, the total wall-clock time of Static (Dynamic) Parallel Progressive GIA.nt is reduced from 335.2 (337.5) min for 2 cores to 51.6 (52.3) min for 16 cores.

In Figure 7.5, we report the scalability of Static and Dynamic Parallel Progressive GIA.nt by keeping the same number of the processing units, while increasing the size of the job, i.e., by gradually increasing their budgets: $BU \in \{5M, 10M, 20M, 30M\}$. Every algorithm is combined with MBRO weights and applied to D_4. The overall wall-clock time is divided into the scheduling (t_s), the verification (t_v) and the overhead (t_r) time. We observe that both t_s and t_r remain constant, while t_v increases in proportion of the size of the budget. This is because the computations included in t_r are irrelevant to the budget size, and budgets of such sizes are not enough to significantly affect the performance of the Scheduling step. For D_4, both algorithms use around 2,000 partitions with small local budgets and, hence, with small local priority queues. As a result, the time needed to push/pop elements from these queues is negligible and does not have a significant impact to the overall performance of the Scheduling step. On the contrary, the size of the budget is decisive for the duration of the Verification step. It is worth noting that both algorithms perform quite similarly, especially with respect to t_r and t_s, but we can notice that the Verification step of the Dynamic Progressive GIA.nt is slightly slower, especially in BU=30M, due to the continuous update of the processing order.

7.3.2 Discussion

We now summarize the main findings of the experiments presented above.

The progressive methods rely on the heuristics of weighting schemes, thus providing no performance guarantees. Instead, they depend on two factors: (i) the characteristics of the input data (e.g., the higher the proportion of qualifying pairs over the candidate ones, the better the performance), and (ii) the size of the budget. Regarding the latter, there is a clear trade-off between recall and precision in Table 7.2: larger budgets increase recall at the cost of lower precision and vice versa, for smaller budgets.

To assess the performance of progressive methods, we compared them with two baseline methods: the Optimal Scheduling, which places all qualifying pairs before the non-qualifying ones, and the Random Scheduling, which produces an arbitrary ordering of all candidate pairs. The closer the progressive methods are to the former baseline method, the better. Our experimental results demonstrate that even though there is still room for improving their performance, they significantly outperform the latter baseline. In fact, $MBRO$ consistently outperforms Random Scheduling in combination with Static Progressive GIA.nt, as shown in Table 7.2. The same applies to the rest of the weighting schemes when they are combined with Dynamic Progressive GIA.nt, especially when they form composite schemes with $MBRO$ as the secondary one [PMMKar].

Regarding time efficiency, the filtering time, t_f, accounts for a negligible portion of the overall run-time, which is dominated by the verification time, t_v. In Table 7.2, we notice that the scheduling time, t_s, is much higher than t_f, albeit significantly lower than t_v in most cases. In general, the scheduling time is significant with respect to t_v, if (i) the budget is very low, yielding very low t_v, (ii) the selected geometries are very small and simple, or (iii) the number of candidate pairs with intersecting MBRs is very high. *The more candidate pairs have intersecting MBRs, the more time consuming is Scheduling*, because its time complexity depends linearly on their number. Note that t_s is not affected by the tiles intersecting the MBR of each target geometry, because their contents, i.e., the source geometry ids they contain, are efficiently added to the current set of candidate pairs and those with disjoint MBRs are later discarded. Most importantly, though, t_s involves the time required to read the target geometries from the disk, unlike t_v, since progressive verification merely processes the top-weighted target geometries, which have already been stored in main memory during Scheduling.

In the case of Dynamic Progressive GIA.nt, the overhead time t_o, which is required for weight updating, is rather low when comparing its value to the corresponding verification time (seconds or minutes in comparison to hours). The reason is that the algorithm re-ranks only the pairs among the top-BU that have not been verified so far. In fact, very few pairs are re-weighted whenever a new topologically related pair is detected.

Regarding the relative run-time of the five weighting schemes, we observe that CF consistently yields the highest verification time, followed by $MBRO$. All other schemes are much faster, yielding similar verification times, with ISP being consistently the fastest one. This is explained by the complexity of the top-weighted candidate pairs in terms of their area as well as the number of their boundary points [PMMKar]. The smaller both measures are for both source and target geometries, the faster is the corresponding progressive verification time. Regarding the scheduling time, though, it remains relatively stable across all weighting schemes, as it is dominated by the time required to read the target geometries from the disk.

Considering the relative effectiveness of the weighting schemes, we observe that CF exhibits the lowest performance in most cases, because it produces scores of very low distinctiveness and has a limited scope, as its search space is reduced to geometries with large MBRs. The characteristics of the other weighting schemes are highly correlated, with $MBRO$ excelling in distinctiveness, thus being ideal for the secondary role in a composite scheme. In this way, it manages to boost the performance of all other weighting schemes, even CF, in the context of Dynamic Progressive GIA.nt [PMMKar].

8. Conclusions

In this deliverable, we presented JedAI-spatial, an open-source system that acts as a library of the state-of-the-art algorithms for Geospatial Interlinking. For the existing algorithms, some of which have not been applied to Geospatial Interlinking before, JedAI-spatial incorporates optimized implementations, while also including new, high-performing techniques. JedAI-spatial facilitates their application by organizing them into a novel 3D taxonomy and by offering two wizard-like interfaces that assume no expert knowledge. Their benchmarking functionality allows for evaluating the relative performance of the available algorithms and for examining the impact of their internal configuration on their performance. We elaborated on JedAI-spatial's architecture, describing the components of its back- and front-end, and performed scalability analyses, highlighting the relative performance of all serial and parallel budget-agnostic algorithms.

8.1 Ongoing Work

The following work is currently in progress, under review at the VLDB 2022 conference.[1]

8.1.1 Supervised Filtering

Similar to the budget-aware (i.e., progressive) workflow in Figure 5.2, this work turns Geospatial Interlinking into an approximate process by adding an intermediate step between Filtering and Verification, called *Supervised Filtering*, as shown in Figure 8.1. It receives as input the set of candidates C and classifies every pair in C as "`likely related`" or "`unlikely related`". The pairs assigned to the latter label are discarded so that the refined set of candidate pairs C', which is returned as output, consists only of the "`likely related`" ones. Then, Verification examines all pairs in C'.

In this context, we use the following notation to measure the performance of Supervised Filtering:

- $TP(C)$ denotes the true positive candidate pairs, i.e., topologically related geometries correctly classified as "`likely related`".
- $FP(C)$ stands for the false positive candidate pairs, which consist of disjoint geometries but are classified as "`likely related`".
- $TN(C)$ denotes the true negative candidate pairs, which involve disjoint geometries correctly categorized as "`unlikely related`".
- $FN(C)$ designates the false negative candidate pairs, i.e., topologically related geometries that are categorized as "`unlikely related`".

In this context, the output of Supervised Filtering is defined as: $C' = TP(C) \cup FP(C)$. Given that Verification is an exact process, the overall performance of the workflow in Figure 8.1 is determined by C'. To quantify its quality, we define the following measures:

- *Recall* expresses the portion of existing topological relations that are detected: $Re = |TP(C)|/(|TP(C)| + |FN(C)|)$.

[1] `https://vldb.org/2022`

Figure 8.1: The Approximate Geospatial Interlinking workflow.

- *Precision* expresses the portion of verified geometry pairs that are topologically related: $Pr = |TP(C)|/(|TP(C)| + |FP(C)|)$.

Both measures are defined in $[0, 1]$, with higher values indicating better performance. Note that the higher the precision, the lower the run-time is, as fewer unrelated geometry pairs are verified, which is a time-consuming process [CN97].

Overall, Approximate Geospatial Interlinking is formalized as:

Problem 3 (Approximate Geospatial Interlinking)**.** *Given a source and a target dataset, S and T respectively, process their geometries such that precision is maximized for recall ≥ 0.85.*

Without loss of generality, we consider 0.85 as an acceptable recall level for most applications that process voluminous datasets. We emphasize recall over precision, because a Geospatial Interlinking process that does not retrieve the vast majority of topological relations is of little use, regardless of its precision.

Features for Supervised Filtering. Unlike the progressive methods in [PMMK21], which associate every pair of geometries with a single score, Supervised Filtering addresses Problem 3 by associating every pair with a feature vector, where every dimension is a separate numerical score.

The desiderata of the features used by our approach are:

1. They should be *generic*, applying seamlessly to LineStrings and Polygons and ideally, to any indexing scheme used by the Filtering step in Figure 8.1.
2. They should be *effective* with high discriminatory power.
3. They should be *efficient*, involving a low extraction cost so that the classification of a geometry pair is much faster than its verification. As a result, they cannot rely on a detailed examination of a geometry pair, e.g., by counting the boundary points they share.

In this context, we propose 31 features for Supervised Filtering. To facilitate their description and understanding, we organize them into four complementary categories:

1. The *area-based features* consider the space occupied by the MBR of each geometry.
2. The *boundary-based features* stem from the characteristics of each geometry's boundary.
3. The *grid-based features* emanate from the indexing scheme of Filtering.
4. The *candidate-based features* rely on the candidates associated with every geometry after Filtering.

The first two categories depend exclusively on the characteristics of the geometries comprising every candidate pair, but the remaining two rely on the Filtering step. For Filtering, we use the space tiling of the state-of-the-art algorithm GIA.nt [PMMK21], which builds an Equigrid, where the dimensions of each cell correspond to the average width and height of the source geometries. Thus, it loads in main memory only the source dataset and reads the target one from the disk,

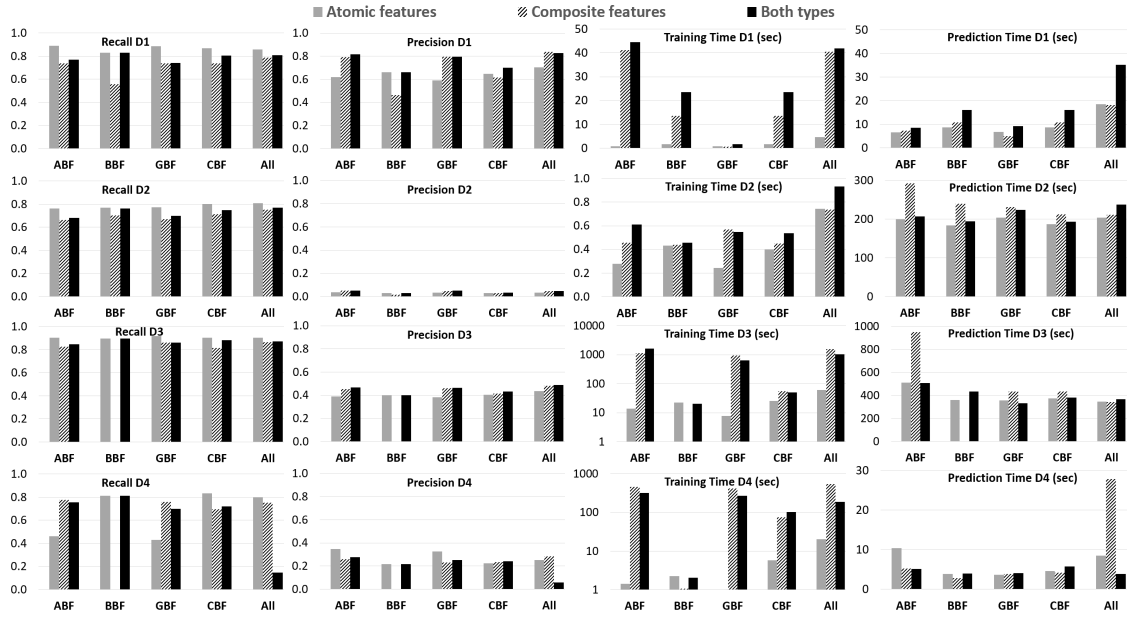

Figure 8.2: Average recall, precision, training and prediction time of SVM, Naive Bayes, Logistic Regression and C4.5 Decision Trees over D_1-D_4 in combination with area-based features (ABF), boundary-based features (BBF), grid-based features (GBF), cardinality-based features (CBF) and all types of features. In each case, we consider atomic and composite features as well as their combination.

reducing the space requirements to a significant extent so as to scale to large datasets with limited memory resources.

Every category includes two types of features: (i) the *atomic*, and (ii) the *composite* ones. The former includes individual, core characteristics of a single geometry, while the latter encompasses combinations of atomic features that typically normalize their values in $[0, 1]$, with higher values implying a stronger likelihood for topological relatedness. These two types allow for exploring the impact of feature complexity on Supervised Filtering.

We omit the detailed definition of features for brevity.

Feature Selection. The more features describe a labelled instance, the more complex and time-consuming is the resulting classification model. To minimize the features used by Supervised Filtering, we perform analytical experiments about the performance of every category and type of feature with respect to recall, precision and run-time, which is broken into training and prediction time. In these experiments, we assume that all candidate pairs have been labelled.

We used the dataset pairs D_1-D_4, excluding D_5, because it cannot be processed within the available memory resources (128 GB), when running on a single CPU – the parallelization of Supervised Filtering is work in progress. For each dataset, we formed a balanced training set that comprises a random sample with 1% of the positive instances and an equal number of randomly selected negative ones. The remaining candidate pairs formed the testing set. All features were rescaled with min-max normalization. We considered four state-of-the-art classifiers [HKP11], namely SVM, Naive Bayes, C4.5 Decision Trees and Logistic Regression. We repeated every experiment 5 times and took the average for every evaluation measure. The resulting performance is reported in Figure 8.2.

We observe that only only two categories of features satisfy the recall threshold in all cases: the atomic candidate-based features and the atomic all features. Their difference in terms of precision is insignificant. However, the candidate-based features excel in time efficiency, as both their training

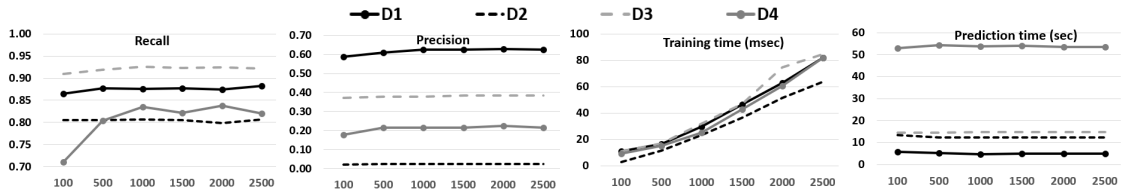

Figure 8.3: Evolution of average recall, precision, training and testing time over D_1-D_4 when using the atomic candidate-based features in combination SVM, Naive Bayes, Logistic Regression and C4.5 Decision Trees with respect to class size (on the horizontal axis).

and their prediction times are significantly lower in all cases. The reason is that they involve just 6 features, much few than the 16 atomic features. As a result, the candidate-based ones yield much simpler and faster, yet equally effective classifiers. For this reason, we exclusively consider these features in the following.

Class size selection. We now examine how sensitive is our feature set with respect to the size of the training set. Even though the labelled instances are generated automatically, restricting their number lowers the cost of Supervised Filtering for three reasons: (i) the training time gets lower, (ii) the resulting classifier is simpler and, thus, the prediction time is lower, and (iii) the candidate pairs that need to be labelled are fewer, thus reducing the time required for building the training set.

To assess the impact of these two parameters, we performed a series of experiments over D_1-D_4, assuming that the labels of all candidates pairs are available. For the training set size, we consider six values: 100 and 500-2,500 instances per class with a step of 500. In every case, the training set is balanced, due to undersampling. We use the same four classification algorithms and report the average performance per evaluation measure in Figure 8.3.

We observe that 500 labelled instances per class suffice for satisfying the recall level across all datasets. These instances are labelled automatically, by randomly selecting pairs of geometries to be verified during a first pass over the input data.

Algorithm Selection. Table 8.1 reports the performance of the individual classification algorithms, when combined with the atomic candidate-based features and 500 labelled instances per class. Every algorithm was applied to every dataset 5 times and we consider the average and the standard deviation for every evaluation measure.

Regarding effectiveness, we observe that in every dataset, SVM achieves the highest recall, which lies consistently well above 0.9, at the cost of the lowest precision. In contrast, Naive Bayes (NB) and Decision Trees (C4.5) emphasize precision at the cost of lower recall. Compared to SVM, their recall drops between 12% and 35%, but their precision almost doubles. For this reason, their F-Measure is consistently higher than that of SVM. In the middle of these two extremes lies Logistic Regression (LR), whose recall is consistently very close or higher than 0.85, while maintaining a very high precision, too. As a result, its F-Measure is significantly higher than SVM and close to that of NB and C4.5, if not higher, as in D_4.

Regarding time efficiency, LR and NB are the slowest algorithms in terms of training (t_r) and prediction (t_p) time, respectively (Table 8.1 excludes the cost of feature extraction, which is common to all algorithms). However, the overall overhead for building the classification model and applying it to all candidate pairs remains below 1 minute in all cases, except for NB over D_4, where it raises up to 2.5 minutes. This time is negligible when compared to the verification times in Table 8.2. For these reasons, we select Logistic Regression as the best classification algorithm for Supervised GIA.nt.

Final performance. The overall performance of the Approximate GIA.nt, whose workflow is presented in Figure 8.1, is reported in Table 8.2. For each dataset, we performed 5 iterations and

Table 8.1: Performance per classification algorithm.

		Recall	Precision	$F1$	t_r (ms)	t_p (sec)
D_1	SVM	0.995±0.001	0.417±0.002	0.588	10±1	3±1
	NB	0.808±0.013	0.718±0.020	0.761	5±1	12±3
	C4.5	0.811±0.033	0.714±0.031	0.759	17±7	1±0
	LR	0.892±0.016	0.591±0.024	0.711	34±4	5±0
D_2	SVM	0.951±0.008	0.015±0.000	0.030	10±1	7±3
	NB	0.656±0.019	0.027±0.003	0.052	4±2	28±1
	C4.5	0.765±0.045	0.038±0.004	0.072	13±2	3±0
	LR	0.847±0.023	0.019±0.000	0.038	18±1	11±0
D_3	SVM	1.000±0.000	0.234±0.001	0.379	10±2	8±1
	NB	0.841±0.034	0.472±0.011	0.604	6±4	34±0
	C4.5	0.892±0.064	0.446±0.036	0.595	10±4	2±0
	LR	0.942±0.024	0.360±0.013	0.521	41±2	14±1
D_4	SVM	0.940±0.056	0.188±0.007	0.313	15±4	36±2
	NB	0.821±0.039	0.219±0.004	0.345	6±1	123±2
	C4.5	0.620±0.083	0.229±0.006	0.335	15±7	12±2
	LR	0.834±0.018	0.220±0.001	0.348	24±9	4±0

considered the average value of each measure. $|C'|$ stands for the number of candidate pairs that were labelled as "`likely related`" by Supervised Filtering. t_f corresponds to the Filtering time, t_s to the run-time (i.e., overhead) of Supervised Filtering, and t_v to the verification time. Note that during Supervised Filtering, at least 500 random candidate pairs are verified per class in order to automatically build the training set with the atomic cardinality-based features that is used for learning the classification model of Logistic Regression.

We observe that compared to Batch GIA.nt, which performs exact Geospatial Interlinking, Approximate GIA.nt sacrifices recall by 8% to 18%, but increases precision by more than 50% in most cases, while the number of candidate pairs is reduced by at least 30%. As a result, the overall run-time is reduced by 46% (D_4) to 74% (D_3).

Table 8.2: The relative performance of Batch and Approximate GIA.nt.

Measure	$\mathbf{D_1}$	$\mathbf{D_2}$	$\mathbf{D_3}$	$\mathbf{D_4}$
Recall	1.000	1.000	1.000	1.000
Precision	0.381	0.013	0.196	0.180
t_f (sec)	27	27	115	90
t_v (hrs)	1.4	3.4	9.3	33.3

(a) Batch GIA.nt

Recall	0.907 ± 0.012	0.824 ± 0.039	0.951 ± 0.016	0.845 ± 0.021
Precision	0.585 ± 0.015	0.020 ± 0.001	0.376 ± 0.011	0.217 ± 0.000
$\lvert C'\rvert$ ($\times 10^6$)	3.72 ± 0.13	8.43 ± 0.83	9.73 ± 0.12	47.26 ± 1.24
L	1,263 ± 41	13,700 ± 74	2,551 ± 67	2,712 ± 67
t_f (sec)	28 ± 3	28 ± 3	114 ± 1	91 ± 1
t_s (min)	4.4 ± 0.2	6.1 ± 0.5	9.3 ± 0.3	34.4 ± 1.3
t_v (hrs)	0.4 ± 0.0	1.2 ± 0.2	2.3 ± 0.2	17.4 ± 0.2

(b) Approximate GIA.nt

Bibliography

[ASN18] Abdullah Fathi Ahmed, Mohamed Ahmed Sherif, and Axel-Cyrille Ngonga Ngomo. RADON2 - a buffered-intersection matrix computing approach to accelerate link discovery over geo-spatial RDF knowledge bases: OAEI2018 results. In *International Workshop on Ontology Matching*, pages 197–204, 2018.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, pages 237–246, 1993.

[CFvO93] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. A small set of formal topological relationships suitable for end-user interaction. In *SSD*, pages 277–295, 1993.

[CN97] Edward P. F. Chan and Jimmy N. H. Ng. A general and efficient implementation of geometric operators and predicates. In *SSD*, pages 69–93, 1997.

[CSE94] Eliseo Clementini, Jayant Sharma, and Max J. Egenhofer. Modelling topological spatial relations: Strategies for query processing. *Comput. Graph.*, 18(6):815–822, 1994.

[DS00] Jens-Peter Dittrich and Bernhard Seeger. Data redundancy and duplicate detection in spatial join processing. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, pages 535–546, 2000.

[EF91] Max J Egenhofer and Robert D Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information System*, 5(2):161–174, 1991.

[EM15] Ahmed Eldawy and Mohamed F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman, editors, *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1352–1363. IEEE Computer Society, 2015.

[FB74] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[FE13] Eric Friedman and Rob Eden. Gnu trove: High-performance collections library for java. `http://trove4j.sourceforge.net/html/overview.html`, 2013.

[GHMT17] Behzad Golshan, Alon Y. Halevy, George A. Mihaila, and Wang-Chiew Tan. Data integration: After the teenage years. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 101–106, 2017.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011.

[JIB10] Anja Jentzsch, Robert Isele, and Christian Bizer. Silk - generating RDF links while publishing or consuming linked data. In *Proceedings of the ISWC 2010 Posters & Demonstrations Track: Collected Abstracts*, 2010.

[KCK01] Kihong Kim, Sang Kyun Cha, and Keunjoo Kwon. Optimizing multidimensional index trees for main memory access. In *SIGMOD*, pages 139–150, 2001.

[LEL97] Scott T. Leutenegger, J. M. Edgington, and Mario Alberto López. STR: A simple and efficient algorithm for r-tree packing. In *ICDE*, pages 497–506, 1997.

[NA11] Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2312–2317, 2011.

[Ngo13] Axel-Cyrille Ngonga Ngomo. ORCHID - reduction-ratio-optimal computation of geo-spatial distances for link discovery. In *ISWC*, pages 395–410, 2013.

[PD96] Jignesh M. Patel and David J. DeWitt. Partition based spatial-merge join. In *SIGMOD*, pages 259–270, 1996.

[PKNK18] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. How good are modern spatial analytics systems? *Proc. VLDB Endow.*, 11(11):1661–1673, 2018.

[PMMK21] George Papadakis, Georgios M. Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. Progressive, holistic geospatial interlinking. In *WWW*, pages 833–844, 2021.

[PMMKar] George Papadakis, Georgios M. Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. Static and dynamic progressive geospatial interlinking. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 2022 (to appear).

[SDSN17] Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. Radon - rapid discovery of topological relations. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 175–181, 2017.

[SDVV20] Georgios M. Santipantakis, Christos Doulkeridis, Akrivi Vlachou, and George A. Vouros. Integrating data by discovering topological and proximity relations among spatiotemporal entities. In *Big Data Analytics for Time-Critical Mobility Forecasting, From Raw Data to Trajectory-Oriented Mobility Analytics in the Aviation and Maritime Domains*, pages 155–179. Springer, 2020.

[SGD+19] Georgios M. Santipantakis, Apostolos Glenis, Christos Doulkeridis, Akrivi Vlachou, and George A. Vouros. stld: towards a spatio-temporal link discovery framework. In *Proceedings of the International Workshop on Semantic Big Data, SBD@SIGMOD*, pages 4:1–4:6, 2019.

[SJ14] Darius Sidlauskas and Christian S. Jensen. Spatial joins in main memory: Implementation matters! *Proc. VLDB Endow.*, 8(1):97–100, 2014.

[SK16] Panayiotis Smeros and Manolis Koubarakis. Discovering spatial and temporal links among RDF data. In *Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, co-located with 25th International World Wide Web Conference (WWW 2016)*, 2016.

[SSC+13] Benjamin Sowell, Marcos Antonio Vaz Salles, Tuan Cao, Alan J. Demers, and Johannes Gehrke. An experimental analysis of iterated spatial joins in main memory. *Proc. VLDB Endow.*, 6(14):1882–1893, 2013.

[TBMT19] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. Parallel in-memory evaluation of spatial joins. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2019, Chicago, IL, USA, November 5-8, 2019*, pages 516–519, 2019.

[TYM+16] MingJie Tang, Yongyang Yu, Qutaibah M. Malluhi, Mourad Ouzzani, and Walid G. Aref. Locationspark: A distributed in-memory data management system for big spatial data. *Proc. VLDB Endow.*, 9(13):1565–1568, 2016.

[TYM+20] Mingjie Tang, Yongyang Yu, Ahmed R. Mahmood, Qutaibah M. Malluhi, Mourad Ouzzani, and Walid G. Aref. Locationspark: In-memory distributed spatial query processing and optimization. *Frontiers Big Data*, 3:30, 2020.

[XLY+16] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. Simba: Efficient in-memory spatial analytics. In *SIGMOD*, pages 1071–1085, 2016.

[YWS15] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: a cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*, pages 70:1–70:4, 2015.

[YZG15] Simin You, Jianting Zhang, and Le Gruenwald. Large-scale spatial join query processing in cloud. In *CloudDM workshop*, pages 34–41, 2015.