NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

MSc THESIS

Supervised Progressive Geospatial Interlinking

Maria Despoina D. Siampou

Supervisor: **Manolis Koubarakis,** Professor

Co-Supervisors: **George Papadakis,** Associate Researcher
Nikolaos Mamoulis, Professor, University of Ioannina

ATHENS

JULY 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εποπτευόμενη Βαθμιαία Διασύνδεση Γεωχωρικών Δεδομένων

Μαρία Δέσποινα Δ. Σιάμπου

Επιβλέπων: **Μανόλης Κουμπαράκης,** Καθηγητής

Συνεπιβλέποντες: **Γεώργιος Παπαδάκης,** Συνεργαζόμενος Ερευνητής
Νικόλαος Μαμουλής, Καθηγητής, Πανεπιστήμιο Ιωαννίνων

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2022

MSc THESIS

Supervised Progressive Geospatial Interlinking

Maria Despoina D. Siampou
S.N.: CS2200017

SUPERVISOR: **Manolis Koubarakis,** Professor

COSUPERVISORS: **George Papadakis,** Associate Researcher
Nikolaos Mamoulis, Professor, University of Ioannina

THESIS COMMITTEE: **Dimitrios Gounopoulos,** Professor
Manolis Koubarakis, Professor
Alexandros Ntoulas, Professor

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εποπτευόμενη Βαθμιαία Διασύνδεση Γεωχωρικών Δεδομένων

Μαρία Δέσποινα Δ. Σιάμπου
Α.Μ.: CS2200017

ΕΠΙΒΛΕΠΩΝ: **Μανόλης Κουμπαράκης,** Καθηγητής

ΣΥΝΕΠΙΒΛΕΠΟΝΤΕΣ: **Γεώργιος Παπαδάκης,** Συνεργαζόμενος Ερευνητής
Νικόλαος Μαμουλής, Καθηγητής, Πανεπιστήμιο Ιωαννίνων

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Δημήτριος Γουνόπουλος,** Καθηγητής
Μανόλης Κουμπαράκης, Καθηγητής
Αλέξανδρος Ντούλας, Καθηγητής

ABSTRACT

Geospatial Interlinking constitutes a useful task that associates pairs of geometries with topological relations. Its high computational cost, though, scales poorly to voluminous datasets. Progressive methods were recently proposed in order to reduce this cost by sacrificing recall to an affordable extent. However, these methods operate in a learning-free manner that relies on mere heuristics, which can be conservative (i.e., retaining too many unrelated pairs) or aggressive (i.e., discarding too many related pairs). In this work, we introduce Supervised Scheduling as a quick and principled way of defining the processing order of the candidate geometry pairs that are likely to be topologically related, based on their classification probability. Our approach leverages generic features with low extraction cost but high discriminatory power that can be automatically labelled. Thorough experiments verify the high performance and robustness of our features as well as the limited size of the training set that suffices for learning an accurate classification model. We integrate Supervised Scheduling into a progressive end-to-end algorithm, which has both a serial and a parallel version. Each version is integrated into JedAI-Spatial and DS-JedAI systems, respectively. Lastly, we compare the performance of our algorithm to the other progressive ones over six real, large datasets.

SUBJECT AREA: Semantic Web, Database Systems, Machine Learning

KEYWORDS: Geospatial Interlinking, Supervised Learning, Binary Classification

ΠΕΡΙΛΗΨΗ

Η Διασύνδεση Γεωχωρικών Δεδομένων αποτελεί μια χρήσιμη διαδικασία, κατά την οποία ζεύγη γεωμετριών συνδέονται σύμφωνα με τις τοπολογικές τους σχέσεις. Η διαδικασία αυτή, ωστόσο, έχει υψηλό υπολογιστικό κόστος, το οποίο κλιμακώνεται ελάχιστα σε μεγάλο όγκο δεδομένων. Πρόσφατα προτάθηκαν αλγόριθμοι βαθμιαίας διασύνδεσης γεωχωρικών δεδομένων, οι οποίοι -για να ελαττώσουν το υπολογιστικό κόστος- θυσιάζουν έως ένα επιτρεπτό όριο το βαθμό ανάκλησης. Παρ'ολα αυτά, οι αλγόριθμοι αυτοί βασίζονται είτε σε απλές ευρετικές μεθόδους, οι οποίες μπορεί να χαρακτηριστούν είτε ως "συντηρητικές" (διατηρόντας, δηλαδή, πολλά τοπολογικά μη-συσχετιζόμενα ζεύγη) είτε ως "επιθετικές" (απορρίπτοντας, δηδαδή, πολλά τοπολογικά συσχετιζόμενα ζεύγη). Λαμβάνοντας υπόψην τα παραπάνω, προτείνουμε την Εποπτευόμενη Δρομολόγηση, σύμφωνα με την οποία τα ζεύγη γεωμετριών διαχωρίζονται σε δυο κατηγορίες: τα ζεύγη που είναι πιθανό να σχετίζονται τοπολογικά και τα ζεύγη που είναι εξαιρετικά απίθανο να σχετίζονται τοπολογικά. Κατά την Εποπτευόμενη Δρομολόγηση, η σειρά επεξεργασίας των υποψήφιων ζευγών, ορίζεται σύμφωνα με την πιθανότητα ταξινόμησης τους σε μια από τις δυο κατηγορίες. Η προσέγγισή αυτή, αξιοποιεί χαρακτηριστικά των γεωμετριών, που συμβάλουν συνολικά στην διάκριση των ζευγών, και των οποίων η εξαγωγή έχει αρκετά χαμηλό κόστος. Τα πειράματά που παρουσιάζουμε, αναδεικνύουν την υψηλή απόδοση και ευρωστία των χαρακτηριστικών μας, έχοντας μάλιστα ένα μικρό σε μέγεθος σετ εκπαίδευσης, το οποίο επαρκεί για την εκμάθηση ενός ακριβούς μοντέλου ταξινόμησης. Παρουσιάζουμε δύο εκδοχές του αλγορίθμου, μια σειριακή και μια παράλληλη, τις οποίες ενσωματώνουμε στα συστήματα JedAI-Spatial και DS-JedAI, αντίστοιχα. Τέλος, συγκρίνουμε την απόδοση της προσέγγισής μας με τις υπόλοιπους βαθμιαίους αλγόριθμους σε έξι υπάρχοντα, μεγάλα σύνολα δεδομένων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σημασιολογικός Ιστός, Συστήματα Βάσεων Δεδομένων, Μηχανική Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Διασύνδεση Γεωχωρικών Δεδομένων, Εποπτευόμενη Μάθηση, Δυαδική Ταξινόμηση

To my family.

ACKNOWLEDGEMENTS

I would like to thank Professor Manolis Koubarakis for his guidance throughout the year, as well as for giving me the opportunity to become a member of the AI Research Group, and therefore meet and collaborate with great researchers. I would also like to thank Dr. George Papadakis and Prof. Nikolaos Mamoulis for their continuous guidance and help during the preparation of this thesis.

The present work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant” (Project Number: HFRI-FM17-2351).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Geospatial data constitutes the cornerstone in numerous applications, especially on the Web. For example, OpenStreetMap alone contains data about the entire globe which amounts to 1.5 terabyte[1]. Geonames describes more than 12 million locations[2], while WorldKG[3] contains around 113.4 million geographic entities [6].

Despite the prominence of geospatial data, its sources are inadequately interlinked on the Linked Open Data cloud. Even though the geospatial data corresponds to almost 20% of the LOD cloud triples, only 7% of the links between the various data sources pertain to geometries [12]. For example, only 0.52% of the OpenStreetMap geometries were linked to Wikidata as of April, 2021 [6].

To address this shortage, Geospatial Interlinking aims to automatically connect the geometric entities between the various data sources of the Semantic Web [16, 15]. In more detail, Geospatial Interlinking takes as input a source and a target dataset, S and T, respectively, and its objective is to interlink S and T by identifying all geometry pairs in $S \times T$ that satisfy a topological relationship, except for the trivial *disjoint* one. As an example, consider Figure 1.1, where LineString g_4 `intersects` LineString g_3, which `touches` Polygon g_1, which `contains` Polygon g_2.

Yet, Geospatial Interlinking is a demanding task that involves two main challenges [16, 15]:

1. Its inherently quadratic time complexity.
2. The time-consuming processing for a single pair of geometries.

Both shortcomings are based on the fact that the brute-force approach has to examine all pairs of geometries as well as that all topological relations of the DE-9IM model need to be extracted from the intersection matrix (cf. Chapter 2). The time complexity of the latter is $O(N \cdot \log N)$, where N stands for the total number of boundary points in the two geometries [3].

The first challenge is addressed through the Filtering-Verification framework that lies at the core of all relevant techniques [2, 14, 15, 16]. The source and target datasets, S and T respectively, are fed to the Filtering step, which produces a set $C \subseteq S \times T : |C| \ll |S| \times |T|$ with pairs that are *candidates*, i.e., likely to have a non-trivial topological relation, because their Minimum Bounding Rectangles (MBRs) are intersecting. C is then forwarded to the Verification step, which examines every geometry pair.

The second challenge is addressed through progressive approaches [14], which turn Geospatial Interlinking into an approximate procedure – they sacrifice recall to a small extent in order to reduce the run-time by orders of magnitude for applications with limited resources (e.g., cloud-based applications with a limited budget for AWS Lambda functions[4], which charge whenever they are called). In essence, progressive methods try to determine the processing order of candidate pairs such that the topologically related ones take precedence. After Filtering, they associate every candidate pair with a score that is proportional to the likelihood that its geometries are topologically related. During Verification, only the

[1] `https://wiki.openstreetmap.org/wiki/Planet.osm`
[2] `https://www.geonames.org/about.html`
[3] `https://www.worldkg.org`
[4] `https://aws.amazon.com/lambda`

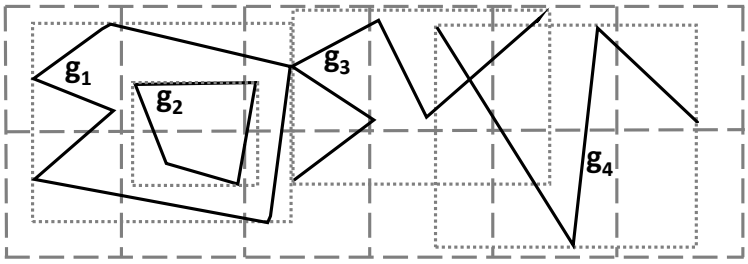

Figure 1.1: An example of topologically related geometries.

top-k weighted pairs are examined, with k configured according to the available temporal or computational resources.

On the downside, the existing progressive methods weigh the candidate pairs according to learning-free heuristics that are based on the tiles that intersect the MBR of each geometry: the number of common tiles, their Jaccard similarity or their Pearson χ^2 test [14]. As a result, they prune the search space in a way that might be too aggressive, missing many related pairs, or too conservative, verifying many non-related pairs.

In this work, we argue that supervised learning provides a principled framework for distinguishing between related and non-related geometry pairs. We formalize *Supervised Scheduling* as a probabilistic binary classification task that orders in decreasing classification probability the candidate pairs that are labelled as "`likely related`". To solve this task, we propose 31 features that represent every pair of geometries. They are generic enough to apply to both LineStrings and Polygons, while their extraction cost is very low. Our objective is to identify the smallest set of features that achieves high accuracy. We experimentally verify its robustness, showing that its high performance is independent of the classification algorithm, while requiring a very small training set. Finally, we propose *Supervised Progressive GIA.nt*, an end-to-end algorithm that uses the Filtering and Verification steps as provided by Progressive GIAnt, but instead of using a simple Scheduling step, it utilizes Supervised Scheduling. The latter, builds the training set on the fly, without any human intervention, learns the classification model and uses it to feed Verification with the best candidates. Extensive experiments show that it significantly outperforms the best progressive method.

Overall, this work makes the following contributions:

- We define Supervised Scheduling, a probabilistic binary classification task that trades slightly lower recall for significantly higher precision and lower run-time.
- We define four categories of classification features, with each one further divided into two subcategories. Each (sub)category includes generic, efficient and effective features.
- We propose Supervised Progressive GIA.nt, a learning-based algorithm that builds a small training set tr on-the-fly, after the first pass over all input data, learns a binary classifier $\mathcal{M}$ over tr and applies $\mathcal{M}$ during the second pass over the input data, significantly reducing the number of candidate pairs and the overall run-time.
- We perform feature selection and use the resulting features in four established classifiers, demonstrating their robustness even when trained over very few labelled instances.

- We experimentally compare our approach to the state-of-the-art progressive method, demonstrating its superiority in terms of effectiveness, robustness and memory efficiency.
- We present a parallelized version of our approach on top of Apache Spark[5], and we evaluate all approaches through a thorough experimental analysis that involves six real, large scale datasets.
- We make our work publicly available.[6]

The rest of the thesis is structured as follows: Section 2 provides the necessary background knowledge, while 2.3 discusses related work in the field. In Section 3, we describe the features and the serial and parallel implementation of the algorithm of our approach. We experimentally fine-tune and compare them with the state-of-the-art progressive method in Section 4. We conclude the thesis in Section 5 along with directions for future work.

[5] `https://spark.apache.org/`
[6] `https://github.com/msiampou/DS-JedAI`

2. BACKGROUND AND RELATED WORK

A large part of the information we deal with on a daily basis has some kind of a geographic dimension. An example of such information can be searching for the opening hours of a restaurant close to our location. The goal of the geospatial Semantic Web research is to make seeking such information easier by allowing exploration, editing and interlinking of heterogeneous information sources with a spatial dimension.

Traditional Geographic Information Systems (GIS) offer rigid sets of geographic features. However, integrating external data sets into these systems typically requires manual integration and programming efforts to ensure that the meaning of the provided information will be processable. Yet, combining the strengths of Linked Data and GIS systems could spur the transition of numerous isolated geographic information to a geospatially enriched Linked Data Web where such information can easily be integrated and processed.

The large-scale adoption of Linked Data[1] started in 2006 and a number of the largest spatial datasets, like LinkedGeoData, already contain spatial information. A considerable number of users and organizations contribute and work with structured geospatial data on the Web. The idea of large-scale collaborative spatial data management is one of the biggest challenges in the area of intelligent information management, aiming the exploitation of the Web as a platform for geospatial data integration as well as for searching and querying for geographic information.

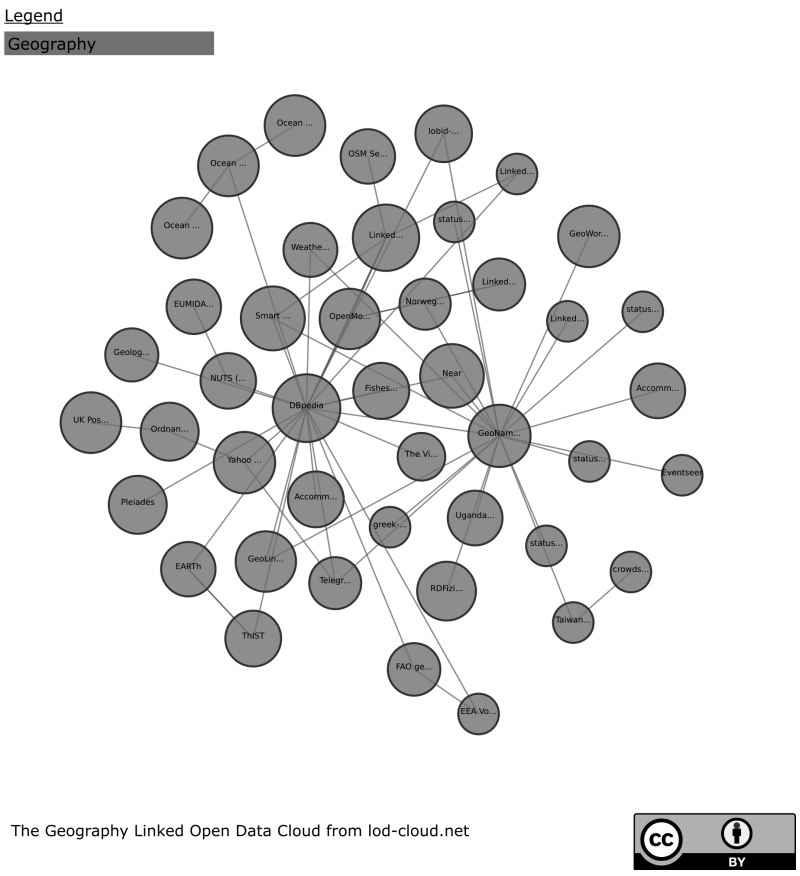

Figure 2.1: The Geography Linked Open Data Cloud.

At the moment, though, the geospatial data is underrepresented in the LOD Cloud. Figure

[1] http://lod-cloud.net

2.1 presents the datasets that have been published in the Linked Data format. Geospatial data corresponds to almost 20% of the LOD cloud triples, however 7% of the triples linking different datasets pertain to geometries [12]. Geospatial Interlinking aims to overcome this limitation.

In this work, we focus on two types of geometries:

- *LineStrings* or *Polylines*, which are sequences of connected line segments
- *Polygons*, which, in the simplest case, are two-dimensional geometries specified by a loop sequence of points, where the first and the last one coincide.

In Figure 1.1, examples of LineStrings are the geometries g_3 and g_4, while Polygons are represented by g_1 and g_2. Both types of geometries consist of three parts; the *interior*, the *boundary*, and the *exterior* (i.e., the rest of the points).

For most types of geometries, the *Dimensionally Extended Nine-Intersection Model* (DE-9IM) [4, 5, 7] defines the following topological relations between two geometries A and B:

1. `Equals`(A, B): A and B have identical interiors & boundaries.
2. `Intersects`(A, B): A and B are not `disjoint`, sharing at least one point of their interiors or boundaries.
3. `Touches`(A, B): A and B have common points in their boundaries, but not in their interiors.
4. `Within`(A, B): A resides in the interior of B.
5. `Contains`(A, B): `within`(B, A).
6. `Covers`(A, B): B resides in A's interior or boundary.
7. `Covered-by`(A, B): `covers`(B, A).
8. `Crosses`(A, B): A and B share part of their interior points, and $dim(A)<dim(B)$ or $dim(B)<dim(A)$.
9. `Overlaps`(A, B): A and B share part of their interior and boundary points, and $dim(A) = dim(B)$.
10. `Disjoint`(A, B): A and B share no point of their boundaries nor of their interiors.

Note that $dim(g)$ is 0, 1 or 2 if g is a point, a line segment or an area, respectively. Note also that all topological relations can be extracted from the intersection matrix of the given geometry pair [2, 14].[2] Following [14], though, we disregard the relation `Disjoint`, because it is not informative, as it typically applies to the vast majority of geometry pairs. We denote the set of the nine *non-trivial* topological relations by $\mathcal{R}$. If a geometry pair is found to satisfy none of these relations, it is assumed to satisfy `Disjoint`.

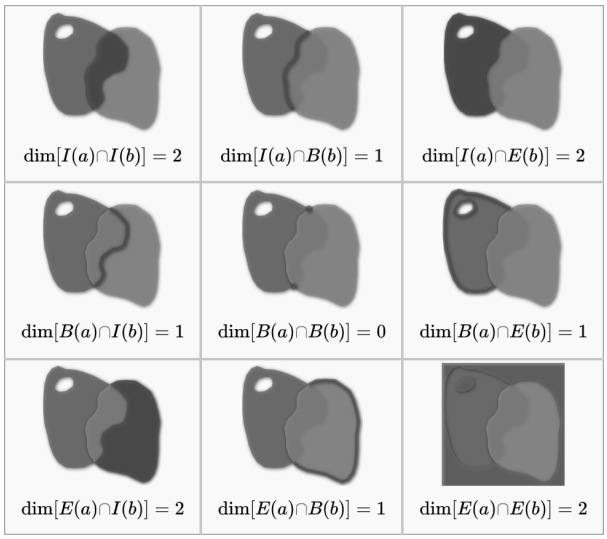

Figure 2.2: Topological Relations between Spatial Objects in Two-Dimensional Space.

2.1 Problem Definition

In this context, Geospatial Interlinking is an exact task that misses no topological relations between the given geometries [14]:

Problem 1 (Geospatial Interlinking)**.** *Given a source and a target dataset,* S *and* T *respectively, compute all non-trivial topological relations between their geometries* $L_{\mathcal{R}} = \{(s, r, t) \subseteq S \times \mathcal{R} \times T : r(s, t)\}$*.*

In the context of Linked Data, the goal is to estimate all topological relations (excluding `Disjoint`) between the source and the target datasets. These can be derived with simple logical conditions from the Intersection Matrix (IM), which is defined as:

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

where dim denotes the dimension of the intersection of the *interior I*, *boundary B*, and *exterior E* of the geometries s and t. For empty intersections, dim is -1 or F (False), while for non-empty ones, dim is equal to 0 in the case of a point, 1 for a line segment and 2 for an area. The values $\{0, 1, 2\}$ are collectively represented by T (True). Figure 2.2 illustrates the aforementioned details.

2.2 Progressive Geospatial Interlinking

Progressive algorithms offer an approximate solution to Problem 1, while operating in a pay-as-you-go manner [14]. In this way, they maximize the throughput of applications with limited resources, e.g., due to the cost of AWS Lambda functions, which charge whenever they are called [19].

[2]See examples in `https://en.wikipedia.org/wiki/DE-9IM#Matrix_model`.

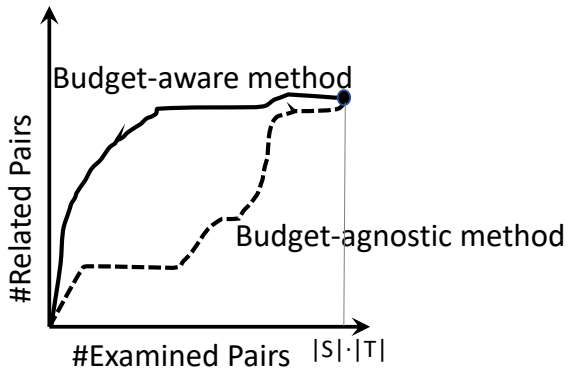

Figure 2.3: PGR for batch and progressive algorithms.

Compared with the exact (batch) solutions to Problem 1, the goal of progressive algorithms is twofold [14]:

1. They should yield the same outcome when processing all input data.
2. They should yield a substantially higher number of related geometry pairs, when terminating prematurely.

These requirements are reflected in the diagram of Figure 2.3, which is formed by the number of verifications on the horizontal axis and the number of related geometries on the vertical one. The gist of progressive algorithms is that they place the related pairs before the non-related ones in the processing order. This is in contrast to the batch algorithms, which define an arbitrary processing order. More formally, the progressive algorithms aim to maximize the area under their curve in Figure 2.3. This evaluation measure is called *Progressive Geometry Recall* (**PGR**) and is defined in $[0, 1]$, with higher values indicating higher effectiveness.

In this context, progressive algorithms tackle the following task [14]:

Problem 2 (Progressive Geospatial Interlinking)**.** *Given a source and a target dataset, S and T, along with a budget BU on the maximum calculations (or running time), compute as many non-trivial topological relations between S and T as possible so that the Progressive Geometry Recall is maximized within BU.*

The progressive algorithms are also evaluated with respect to: (i) *run-time*, (ii) *precision*, i.e., the ratio between the detected related pairs and the number of verifications, and (iii) *recall*, i.e., the ratio between the detected and the existing related pairs.

2.3 Related Work

Geospatial Interlinking is a core task in the process of populating the Semantic Web with links between its geospatial entities [2, 14, 16].

The first relevant technique is Silk-spatial [17]. Its Filtering divides the surface of the Earth into a user-defined number of tiles, creating a fixed EquiGrid. As a result, its tiles are usually coarse-grained, in the sense that they involve a large number of geometry pairs. Therefore, too many pairs are verified, incurring a computational cost that is close to that

of a brute-force approach. Its Verification examines the candidate pairs inside every tile in parallel, leveraging Apache Hadoop[3]. The latter step also considers a single topological relation, even though it uses the same Filtering for all relations, resulting in the repetition of the algorithm when another relation needs to be examined over the same data.

To go beyond Silk-spatial, RADON [16] builds a dynamic EquiGrid, based on the input data. The Filtering step also involves a swapping strategy, which goes through all source and target geometries to identify the dataset with the smallest Estimated Total Hypervolume, in an effort to minimize the size of the Equigrid. Due to this strategy, however, RADON needs to maintain both input datasets in memory, and thus demanding high space requirements. Given that every geometry is assigned to all tiles intersecting its MBR, the contents of the resulting tiles are overlapping. To avoid duplicate verifications, RADON maintains a main memory a hash-table with all geometry pairs verified so far. Yet, this renders its massive parallelization non-trivial: special care should be taken to partition the input data among the available workers in a way that avoids all redundant verifications (broadcasting all geometries to all workers is not an option for large datasets, due to the high memory requirements). The Verification step operates at the level of individual topological relations, incorporating specialized filters for some of them. For instance, equals is verified only after ensuring that the source and target geometries have identical MBRs. Similar to Silk-spatial, though, the entire algorithm is repeated whenever another relation is examined over the same data.

stLD [15] enhances RADON in four ways: (i) its Filtering supports a series of indices, such as R-Tree, Equigrid and a hierarchical grid; (ii) only one of the two input datasets is indexed, reducing the memory requirements and treating the second input dataset as a stream of geometries; (iii) MaskLink estimates the overlap of every geometry with every tile to check whether it is contained in the space left empty by the rest of the geometries in the tile. If this is true, the entire tile is skipped, without generating any candidate pairs; (iv) Apache Flink is used for massive parallelization.

These works operate at the level of a single topological relation, repeating the entire processing for every relation of interest. This is addressed by RADON2 [2], which simultaneously extracts all topological relations from the intersection matrix of two geometries.

GIA.nt [14] combines the advantages of all the above works. During Filtering, it loads only the smallest dataset in main memory and indexes it with an Equigrid, whose granularity depends on the characteristics of the smallest input dataset. During Verification, it reads the largest dataset from the disk, one geometry at a time. For each geometry, it retrieves the candidate pairs from the Equigrid and verifies those with intersecting MBRs. Massive parallelization on Apache Spark[4] minimizes the run-time.

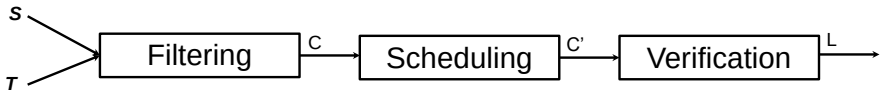

Figure 2.4: Learning-free Progressive Geospatial Interlinking.

The above approaches operate in a batch manner that produces results (in an arbitrary order) only after processing the entire input. This is incompatible with geospatial applications of limited computational and/or temporal resources (e.g., cloud-based apps). To accommodate them, progressive methods were proposed in [14], turning Geospatial Interlinking into an approximate process that promotes precision at the expense of recall. As

[3] http://hadoop.apache.org
[4] http://spark.apache.org

shown in Figure 2.4, they extend the Filtering-Verification framework with an intermediate step, called *Scheduling*, which orders the set of candidate pairs C such that the related ones are processed before the non-related. This is achieved through a weighting scheme that considers the tiles intersecting the MBR of every geometry, assigning higher scores to pairs that are more likely to be related. Given a user- or application-defined budget of k verifications, Progressive GIA.nt verifies the top-k weighted pairs, while Progressive RADON orders the tiles according to their size and applies Progressive GIA.nt inside every tile, until consuming the budget. None of them leverages machine learning to enhance the time efficiency of Geospatial Interlinking.

3. APPROACH

In this chapter we present Supervised Progressive GIAnt, a system that transforms the scheduling step of the learning-free Progressive methods to a supervised one, leveraging the power of probabilistic binary classification procedures. In Supervised Progressive GIAnt, every pair of geometries that their Minimum Bounding Rectangles are intersecting, is associated with a n-dimensional feature vector, where $n = 31$. In the following chapters we refer to these pairs as *candidate pairs*. The total candidate pairs are order according to their classification probability, in terms of their likelihood to be topologically related.

3.1 Features for Supervised Scheduling

The Scheduling step in Figure 2.4 applies a learning-free progressive method that associates every pair of geometries with a single score [14]. We argue that this is not sufficient for addressing Problem 2. Instead, we introduce a Supervised Scheduling step that performs *probabilistic binary classification*, 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:

- *They should be generic*, applying seamlessly to LineStrings and Polygons and ideally, to any indexing scheme used by the Filtering step in Figure 2.4.
- *They should be effective* with high discriminatory power.
- *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* that consider the space occupied by the MBR of each geometry.
2. The *boundary-based features* that stem from the characteristics of each geometry's boundary.
3. The *grid-based features* that emanate from the indexing scheme of Filtering.
4. The *candidate-based features* that 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 [14], which builds an Equigrid, where the dimensions of each cell correspond to the average width and height of the source geometries.

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.

Next, we delve into the features of every category and type.

3.1.1 Area-based features

To be generic, this category considers the area occupied by the MBR of a geometry, rather than the area occupied by the geometry itself (this is not true for LineStrings, which interior coincides with their boundary).

In this context, the atomic features are the following:

- (F1) Source MBR Area
- (F2) Target MBR Area
- (F3) Intersection MBR Area

The first two features assume that the larger the MBR of a geometry is, the more likely it is to be related with another geometry lying within the same index. The third feature assumes that the larger the overlap of two MBRs is, the more likely are the respective geometries to satisfy at least one non-trivial topological relation.

The composite features normalize the atomic ones in $[0, 1]$:

- (F4) Intersection MBR normalized by Source MBR = $F3/F1$
- (F5) Intersection MBR normalized by Target MBR = $F3/F2$
- (F6) Jaccard MBR Overlap = $F3/(F1 + F2 - F3)$

3.1.2 Boundary-based features

This category includes the two features characterizing the border of LineStrings and Polygons, i.e., their points, and their length. We defined four atomic features:

- (F7) Number of Source Boundary Points
- (F8) Number of Target Boundary Points
- (F9) Source Boundary Length
- (F10) Target Boundary Length

Note that (F7) and (F8) capture the complexity of a geometry, as higher values indicate more complicated boundaries. Therefore, the rationale behind (F7)-(F10) is that the more complex and longer the boundary of a geometry is, the more likely it is to satisfy at least one topological relation.

The composite features form normalized measures of complexity, expressing the average number of boundary points per length unit:

- (F11) Normalized Source Boundary Complexity = $F7/F9$
- (F12) Normalized Target Boundary Complexity = $F8/F10$

For both features, higher values indicate higher complexity and possibly greater chances for topological relations.

3.1.3 Grid-based features

Using GIA.nt's Filtering, a uniform grid is built, based on the average dimensions of the source geometries. Every geometry is then placed into all tiles that intersect its MBR, defining the following atomic features:

- (F13) Number of Tiles Intersecting the Source MBR
- (F14) Number of Tiles Intersecting the Target MBR
- (F15) Number of Common Tiles

We implicitly assume that all features are proportional to the likelihood that a geometry (pair) is topologically related.

The composite features normalize the atomic ones:

- (F16) Common Tiles Normalized by Source Tiles = $F15/F13$
- (F17) Common Tiles Normalized by Target Tiles = $F15/F14$
- (F18) Jaccard Common Tiles = $F15/(F13 + F14 - F15)$
- (F19) Pearson's χ^2 test [14] receives as input the atomic features F13-F15 and returns a value proportional to the dependency of the candidate pair. In other words, it checks whether the distribution of tiles intersecting the source MBR remains the same if we exclude the tiles intersecting the target MBR, and vice versa.

3.1.4 Candidate-based features

This category considers the contents of the tiles intersecting the MBR of a geometry g through: (i) the total number of candidates, i.e., the total number of geometries of the other input dataset that participate in the same tiles, (ii) the number of *distinct* candidates, i.e., the cardinality of the *set* of candidates, which disregards multiple appearances of the same geometry, and (iii) the number of distinct *real* candidates, which intersect MBR(g), too.

Overall, the following atomic features are defined:

- (F20) Total Candidates for Source Geometry
- (F21) Distinct Candidates for Source Geometry
- (F22) Real Candidates for Source Geometry

- (F23) Total Candidates for Target Geometry
- (F24) Distinct Candidates for Target Geometry
- (F25) Real Candidates for Target Geometry

For all these features, we assume that higher values correspond to a stronger likelihood for topological relatedness.

The composite features normalize the atomic ones in $[0, 1]$:

- (F26) Source Distinct Candidates Normalized by Total = $F21/F20$
- (F27) Source Real Candidates Normalized by Total = $F22/F20$
- (F28) Source Real Candidates Normalized by Distinct = $F22/F21$
- (F29) Target Distinct Candidates Normalized by Total = $F24/F23$
- (F30) Target Real Candidates Normalized by Total = $F25/F23$
- (F31) Target Real Candidates Normalized by Distinct = $F25/F24$

3.2 Supervised Progressive GIA.nt

We now describe the algorithm that implements the pipeline in Figure 2.4, by replacing Scheduling with Supervised Scheduling. Following GIA.nt [14], it first indexes the smallest input dataset, i.e., the source dataset. The dimensions of the grid cells are specified in Line 1 of Algorithm 1 as $\Delta_x = mean_{s \in S} MBR(s).width$ and $\Delta_y = mean_{s \in S} MBR(s).height$. Based on these dimensions, the lower left MBR point $(x_1(s), y_1(s))$ and the upper right MBR point $(x_2(s), y_2(s))$ of every source geometry s are estimated in Lines 2-3. Together with Δ_x and Δ_y, these points determine the tiles that intersect $MBR(s)$ and should contain s (Lines 4-10). The Equigrid index I is ready after Line 11.

As an example of this indexing, assume that $\Delta_x = 4$ and $\Delta_y = 3$. For a geometry $POLYGON(20\ 90,\ 20\ 93,\ 16\ 93,\ 16\ 90,\ 20\ 90)$, the lower left MBR point is $(16,\ 90)$ and the upper right one is $(20, 93)$. Hence, this geometry participates in the tiles defined by $\lfloor 16/\Delta_x \rfloor = 4 \leq i \leq 5 = \lceil 20/\Delta_x \rceil$ and $\lfloor 90/\Delta_y \rfloor = 30 \leq j \leq 31 = \lceil 93/\Delta_y \rceil$.

The training of the binary probabilistic classification model is carried out in Lines 12-46. For every target geometry t, the tiles that intersect its MBR are inferred from its lower left and the upper right MBR points (Lines 15-17). The source geometries participating in these tiles are aggregated into the *set* of candidates C_S (Line 18). Even though every source geometry appears in C_S just once, a counter measures its actual frequency across the tiles intersecting $MBR(t)$ (we ommit the details for brevity). This counter is used for updating the candidate-based features F20 and F21 for every candidate source geometry $s \in C_S$ (Lines 23-24). If $MBR(s)$ intersects $MBR(t)$, feature F22 is updated, too (Lines 25-26). Then, the two geometries are added to the random sample of pairs to be verified if their id is among the selected ones (Lines 27-29).

The random selection of pairs is carried out in Line 12, through the *randomGenerator* function, which receives two arguments: the maximum number m of pairs to be verified

and labelled during training, and the range D, within which it searches for these pairs. Ideally, the former is set to $2 \cdot N$, where N is the input parameter that specifies the required number of labelled instances per class, while the latter should be set to $|C|$, i.e., the number of candidate pairs in the given datasets. In practice, though, $|C|$ is a-priori unknown, while m should be much larger than $2 \cdot N$, due to the class imbalance, as most candidates entail disjoint geometries (cf. Section 4).

Several approaches address the class imbalance problem [10, 11]: (i) *oversampling* randomly resamples the minority class until both classes have the same size, (ii) *undersampling* randomly samples a subset of equal size from both classes, (iii) *cost-sensitive learning* trains a classifier with a high misclassification cost for the minority class, and (iv) *ensemble learning* trains several classifiers such that they collectively label every instance. However, oversampling yields very large training sets that foster overfitting, due to the repetition of the minority class instances, while cost-sensitive and ensemble learning produce complex and, thus, time-consuming classification models. In contrast, undersampling allows for minimizing the training and the prediction time, as it works well with small training sets that learn simple, fast, but effective classifiers. For these reasons, *Supervised Progressive GIA.nt relies on undersampling*.

In this context, in Line 12, D is set to the maximum possible range of candidate pairs, i.e., the Cartesian product $|S| \times |T|$ (or to the maximum integer value supported by Java) and m to two orders of magnitude larger than N, i.e., $m = 100 \cdot N$, to make up for class imbalance and the sampled ids that exceed $|C|$.

Next, the sampled pairs are shuffled, to randomize their order (Line 34) and verified in order to extract their labels (Lines 35-45). The topologically related ones are added to the set of positive pairs and the rest to the negative pairs (Lines 37-41). In practice, only the first N from each class are taken into account, but we omit this for brevity. As soon as the necessary number of instances is gathered for both classes, the loop terminates (Lines 42-44).

Subsequently, the feature vectors of the selected candidate pairs are generated in Line 46. Most features rely on inherent characteristics of the geometries in each pair. However, features F15-F19, F23-F25 and F29-F31 require that Lines 14-22 are repeated for every sampled *target* geometry. The resulting training set L is then fed to the selected algorithm to learn the classification model $\mathcal{M}$ (Line 46).

Then, the algorithm iterates once more over the target dataset and for each geometry t, it gathers the source candidates from the tiles intersecting $MBR(t)$, as in Lines 14-22. During this process, the features F23-F25 are computed for t, if necessary (we omit the details). Subsequently, for every source candidate s with $MBR(s)$ intersecting $MBR(t)$, a feature vector v is generated (Lines 50-52). The vector is fed to $\mathcal{M}$, which predicts the classification probability for the pair $\{s, t\}$, $w_{s,t}$ (Line 53). If $w_{s,t}$ exceeds the probability corresponding to min_w, $\{s, t\}$ is added to the priority queue T_C, which maintains the most likely related pairs that fit within the given budget BU (Lines 54-55). Note that min_w is initialized to 0.5 in Line 47 so as to exclude pairs classified as unlikely related. Note also that min_w is updated to the probability of the $(BU + 1)^{th}$ top-weighted pair, whenever the size of T_C exceeds the specified budget (Lines 56-59). Finally, the overall top-BU weighted pairs are verified in decreasing classification probability (Lines 64-65); their intersection matrix IM is computed and its topological relations are added to the output set of links $L_{\mathcal{R}}$ (Lines 66-67).

Note that the pairs verified in Line 36 are also included in the output $L_{\mathcal{R}}$. To avoid redundant verifications, their ids are maintained in a hash map that is checked between Lines

50 and 51, but we omit these details for brevity. Note also that Lines 1-11 correspond to the Filtering step, Lines 12-63 to the Supervised Scheduling step and Lines 64-68 to the Verification step of Figure 2.4.

Overall, Supervised Progressive GIA.nt has the same space complexity as Progressive GIA.nt, which is linear with respect to the input – the space occupied by the learned model and the candidate pairs that are automatically labelled is constant, due to the parameter configuration in Section 4. Its time complexity is also equivalent to Progressive GIA.nt, amounting to $O(|S|+|T|\cdot|\bar{C}_S|\cdot\log|BU|+|BU|)$, where $|\bar{C}_S|$ stands for the average number of source candidates per target geometry and $\log|BU|$ for the maximum cost of inserting a candidate pair in the priority queue. The first part corresponds to Filtering, the second one to Supervised Scheduling and the last one to Verification. Note that the time required by Lines 34-46, which label the sample of candidate pairs and train the classification model, is negligible, as demonstrated in Section 4.

3.3 Massive Parallelization

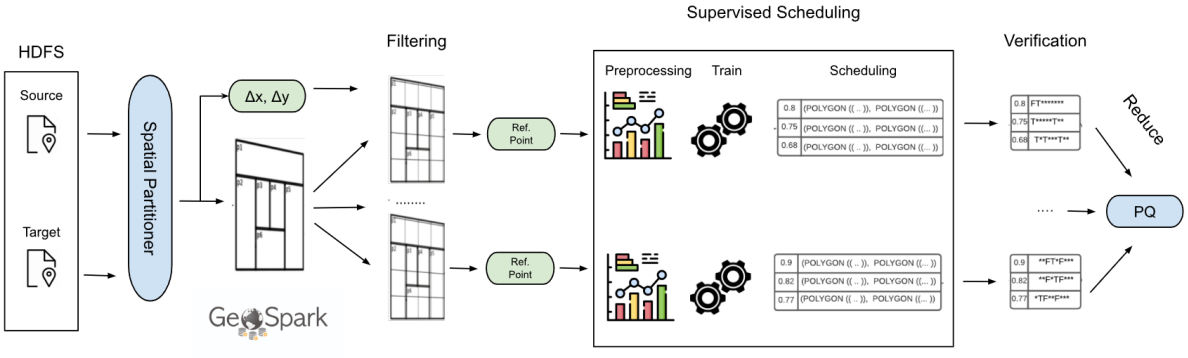

Figure 3.1: Integration of Supervised Scheduling in DS-JedAI.

We have integrated our approach in the Distributed-Spatial JedAI System[1] [2] (DS-JedAI). DS-JedAI is implemented on top of Apache Spark and can run in any distributed or standalone environment that supports the execution of Apache Spark jobs. Our approach is outlined in Figure 3.1.

Following the system's specifications, both datasets are loaded as RDDs[3] and are spatially partitioned based on GeoSpark's Quad Tree, which is built from a sample of the source geometries. Both source and target RDDs are partitioned using the same partitioner, therefore geometries that are considered to be "topologically close" end up to partitions with the same *ID*. The RDDs with the same *partition ID* are then merged. This way, each RDD contains the geometries that lie within the same partition, ensuring that the ones that are likely to satisfy a topological relation coexist in the same partition.

Before joining the RDDs of source and target geometries in each partition, the granularity of space tiling is estimated. We use the same tile dimensions as in serial GIA.nt, as well as in our serial implementation. As previously mentioned, to calculate the granularity of space

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

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

[3] Spark revolves around the concept of a Resilient Distributed Dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel.

tiling, the computation of $\Delta_x = mean_{s \in S} MBR(s).width$ and $\Delta_y = mean_{s \in S} MBR(s).height$ is required. This computation is performed by the Driver, who at first requires each Executor to sum the extends of its local source geometries. Then, the Driver aggregates the results and computes the tile dimensions. In the end, Δ_x and Δ_y are broadcast to the Executors. All in all, this is a single *MapReduce* job.

During the Map phase, every Executor receives as input a partition of both input datasets and applies Filtering to index the source geometries. Then, it applies Supervised Scheduling, processing the target geometries one by one to estimate their weights with the intersecting source geometries, based on their classification probability. Supervised Scheduling consists of the Pre-processing, Training and Scheduling steps.

During the Pre-processing step, the relevant information regarding our feature extraction is computed. Such information can be the total, distinct and real candidates for source and target geometries. During this step we also create a balanced set of related and non-related pairs and define it as our training set. Next, the Training step takes place, which consists of the training of a Linear Regression classifier. Note that, here, each partition creates a local set of sample instances and trains a local classifier, meaning that each Executor performs computations independently, promoting concurrency and making the most of massive parallelization. Regarding Scheduling, a feature vector for each geometry pair with intersecting tiles is generated and passed through the classifier, which outputs their classification probability. The pairs are then scheduled based on their probability to be related. We set this probability to be the pair's weight w. Each pair is added to a local min-max priority queue according to w.

Each partition stores its top-k weighted pairs in the aforementioned min-max priority queue, where k is the local budget (local-BU). Local budget is is derived by dividing the global budget (BU) among the data partitions in proportion to the source geometries they contain. The target geometries are not taken into account, as they are not bulk loaded beforehand, but are read on-the-fly, one by one, similar to the serial implementation of rest of GIA.nt Progressive methods. After all target geometries have been processed, the Verification phase computes the intersection matrix for the top-k weighted candidate pairs in the local priority queue. The qualifying pairs of each Executor are aggregated by the Reduce phase.

Last but not least, the aforementioned spatial partitioning, results in uneven partitions that are skewed with respect to the volume of data and the corresponding computational cost. As a result, each partition ends up with a different workload, and therefore some of them require significant time, while others complete their jobs instantaneously, leaving the corresponding nodes idle. To tackle this issue, DS-JedAI distinguishes the partitions into overloaded and well-balanced ones. The former are defined as those with a number of source geometries significantly higher than the average one across all partitions. To detect them, the Z-score[4] is utilized. Z-score measures how many standard deviations a value is away from the mean value. In this context, a partition is considered to be overloaded if its Z-score exceeds a predefined threshold th_a, set to $th_a = 2.5$. The rest of the partitions are marked as well-balanced and are processed as described above. After completing their processing, the entities of the overloaded partitions are indexed and re-partitioned using a *HashPartitioner* that is based on tiles *ID*. In this way, geometries indexed in the same tiles will be placed in same partitions, without missing any candidate pairs. Redundant pairs are again discarded with the reference point technique. Overall, this is an effective and efficient load balancing strategy as long as it applies to a small portion of the input data, and not to the entire dataset, given that it requires the replication of each entity as many

[4] https://en.wikipedia.org/wiki/Standard_score

times as the numbers its tiles.

Algorithm 1: Supervised Progressive GIA.nt.

input : the source dataset S, the target dataset T, the feature set F, the maximum sample size m, the class size N, the probabilisic classification algorithm A
output: the links $L_{\mathcal{R}} = \{(s, r, t) \subseteq S \times T \times \mathcal{R} : r(s, t)\}$

```
1  I ← {};   (Δx, Δy) ← defineIndexGranularity(S);
2  foreach geometry s ∈ S do                                   // filtering indexes the source geometries
3      (x1(s), y1(s), x2(s), y2(s)) ← getDiagCorners(s);
4      for i ← ⌊x1(s) · Δx⌋ to ⌈x2(s) · Δx⌉ do
5          for j ← ⌊y1(s) · Δy⌋ to ⌈y2(s) · Δy⌉ do
6              I.addToIndex(i, j, s);
7              j ← j + 1;
8          end
9          i ← i + 1;
10     end
11 end
12 sourceStats ← {};  id ← 0;  sample ← {};  sampleIds ← randomGenerator(m,D);
13 foreach geometry t ∈ T do                                   // first pass over the target geometries
14     C_S ← {} ;                                              // the set of source candidates
15     (x1(t), y1(t), x2(t), y2(t)) ← getDiagCorners(t);
16     for i ← ⌊x1(t) · Δx⌋ to ⌈x2(t) · Δx⌉ do
17         for j ← ⌊y1(t) · Δy⌋ to ⌈y2(t) · Δy⌉ do
18             C_S.add(I.getTileContents(i, j));
19             j ← j + 1;
20         end
21         i ← i + 1;
22     end
23     foreach geometry s ∈ C_S do
24         sourceStats ← updateTotalDistinctPairs(s);
25         if intersectingMBRs(s, t) then
26             sourceStats ← updateRealPairs(s);
27             if sampleIds.contains(id) then
28                 sample.add({s, t});
29             end
30             id ← id + 1;
31         end
32     end
33 end
34 negPairs ← {};  posPairs ← {};  shuffle(sample) ;          // instance labelling
35 foreach pair {s, t} ∈ sample do
36     isRelated ← verifyPair({s, t});
37     if isRelated then
38         posPairs.add({s, t});
39     else
40         negPairs.add({s, t});
41     end
42     if N ≤ |posPairs|  &  N ≤ |negPairs| then
43         break;
44     end
45 end
46 L ← getFeatures(posPairs ∪ negPairs, F, sourceStats, I);   M ← trainModel(L);
47 L_R ← {};  min_w = 0.5;  T_C ← {} ;                        // priority queue
48 foreach geometry t ∈ T do                                  // second pass over the target geometries
49     ... ;                                                   /* Same as Lines 14-22 */
50     foreach geometry s ∈ C_S do
51         if intersectingMBRs(s, t) then
52             v ← getFeatureVector(s, t, F);
53             w_{s,t} ← M.getClassificationProbability(v);
54             if min_w < w_{s,t} then
55                 T_C.add({s, t}, w_{s,t});
56                 if BU < T_C.size() then
57                     head = T_C.pop();
58                     min_w = head.getWeight();
59                 end
60             end
61         end
62     end
63 end
64 while T_C ≠ {} do                                          // verification
65     tail = T_C.popLast();
66     IM ← verify(tail.s, tail.t);
67     L_R ← L_R ∪ IM.getRelations();
68 end
69 return L_R;
```

4. EXPERIMENTAL ANALYSIS

We now present the experiments that investigate the effectiveness and efficiency of our approach in contrast to the other progressive approaches. More precisely, in Section 4.2 we present the experiments conducted on the serial implementation regarding the size of the training set, the selection of the most suitable classification algorithm and the performance of our approach. The massive parallelization of our approach is examined in Section 4.3, where we illustrate the performance of the approach as well as discuss our main findings.

4.1 Experimental Setup

The experiments related to the serial implementation were carried out on a server with Intel Xeon Gold 6238R CPU @ 2.2 GHz with 28 cores and 256GB RAM. In all cases, a single CPU was used. All experiments were implemented and performed in Java 15, using Weka 3.8 [9] for the classifiers.

Regarding the parallel experiments, they were performed in a standalone machine that contains 32 virtual cores[1] at 2.20GHz and 128GB of memory. Unless specified otherwise, for the experiments we used 16 Executors with 2 cores each and 7GB of memory. The implementation is in Scala 2.12 using Spark 2.4.

4.1.1 Datasets

To assess the performance of our approach, we used large-scale, real-world datasets that are popular in the literature [8, 14, 18]. They are publicly available [1] and comprise data imported from the US Census Bureau TIGER files, namely USA's Area Hydrography (AREAWATER), Linear Hydrography (LINEARWATER), roads (ROADS) and edges (EDGES), as well as data extracted from OpenStreeMap, representing lakes (Lakes), parks (Parks) and roads (Roads) as well as of all buildings (Buildings) around the world. The datasets are combined into six pairs, presented in Table 4.1, D_1-D_6. They cover all possible combinations of geometry types: In D_1, D_2 and D_4 the source geometries are Polygons and the target ones LineStrings. Vice versa for D_6. D_3 and D_5 are homogeneous, involving only Polygons and only LineStrings, respectively.

In the following, Section 4.2 elaborates on the serial experiments that were carried out over the five smallest pairs of datasets, D_1 to D_5. The largest dataset pair, D_6, does not fit into the main memory of our stand-alone server. It is used only in the parallel experiments that are presented in Section 4.3.

4.1.2 Evaluation measures

We assess *effectiveness* through precision, recall and PGR (cf. Section 2). Precision and recall are computed using the following formulas:

$$Precision = P_Q^{D,BU} / P_Q^{BU} \tag{4.1}$$

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

Table 4.1: The dataset pairs used in our experiments.

	D_1	D_2	D_3	D_4	D_5	D_6
Source Dataset	AREAWATER	AREAWATER	Lakes	Parks	ROADS	Roads
Target Dataset	LNWATER	ROADS	Parks	Roads	EDGES	Buildings
#Source Geom.	2,292,766	2,292,766	8,326,942	9,831,432	19,592,688	72,339,926
#Target Geom.s	5,838,339	19,592,688	9,831,432	72,339,926	70,380,191	114,796,567
Cartesian Product	$1.34 \cdot 10^{13}$	$4.49 \cdot 10^{13}$	$8.19 \cdot 10^{13}$	$7.11 \cdot 10^{14}$	$1.38 \cdot 10^{15}$	$8.30 \cdot 10^{15}$
Candidate Pairs	6,310,640	15,729,319	19,595,036	67,336,808	430,597,631	257,075,645
`#Contains`	806,158	3,792	267,457	5,147,704	53,758,453	274,953
`#CoveredBy`	0	0	1,944,207	47,253	12,218,868	82,828
`#Covers`	832,843	4,692	267,713	5,284,672	53,758,453	274,966
`#Crosses`	40,489	106,823	217,198	5,700,257	6,769	313,566
`#Equals`	0	0	61,712	2,047	12,218,868	18,909
`#Intersects`	2,401,396	199,122	3,841,922	12,145,630	163,982,138	1,037,153
`#Overlaps`	0	0	488,814	42,331	73	54,810
`#Touches`	1,554,749	88,507	986,522	1,210,230	110,216,843	331,166
`#Within`	0	0	1,943,643	47,155	12,218,868	81,567
Total Topological Relations	5,635,635	402,936	10,019,188	29,627,279	418,379,333	2,481,027

$$Recall = P_Q^{D,BU}/BU \tag{4.2}$$

$$PGR = \sum_{i=1}^{|P|} P_Q^i/|P_Q^{BU}| \tag{4.3}$$

where $P_Q^{D,BU}$ stands for the number of detected qualifying pairs and P_Q^BU for the maximum possible number of qualifying pairs within BU. The latter essentially denotes the number of possible verifications. All measures are defined in $[0, 1]$, with higher values indicating higher effectiveness.

For the *time efficiency*, we consider the overall run-time RT, which is the time that intervenes between receiving the input data and producing the detected topological relations as output. For Supervised Scheduling, RT includes the *training time* (t_r), which captures the time required to learn the classification model, and the *prediction time* (t_p), which denotes the time required to apply the learned model to the set of candidates C.

4.2 Serial Processing

4.2.1 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 Scheduling, we perform analytical experiments about the performance of every category and type of features with respect to effectiveness and time efficiency. In these experiments, we assume that all candidate pairs have been labelled.

For each of the three smallest datasets, D_1-D_3, 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 established probabilistic classification algorithms [10]: Naive Bayes, Random Forest, Logistic Regression and Bayesian Networks. We repeated every experiment 5 times and took the average for every evaluation measure. The resulting performance is reported in Figure 4.1.

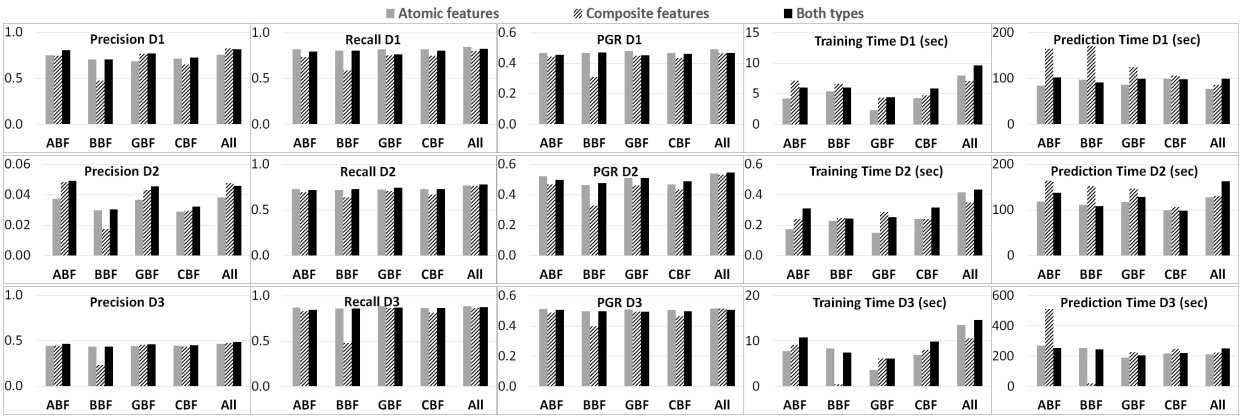

Figure 4.1: Average performance of area-based (ABF), boundary-based (BBF), grid-based (GBF), candidate-based (CBF) and all (All) features over D_1-D_3. In each case, we consider atomic and composite features as well as their combination.

Regarding the type of features, we observe the atomic consistently outperform the composite ones in terms of recall and PGR by 9.6% and 10%, on average, across all datasets and feature categories. This situation is reversed in half the cases for precision, but still the composite features underperform by 2.3%, on average. Combining both feature types yields an intermediate recall and PGR, but achieves the highest precision in practically all cases. Regarding time efficiency, the atomic features are faster than the composite ones by 19.3% and 25.3%, on average, with respect to training and prediction time, respectively. This indicates that more complex patterns are learned from the composite features, probably due to their lower discriminativeness. The combination of both feature types is slower than the atomic features by 36.2% and 8.4%, respectively. This is expected, given that the higher number of features typically yields more complex and time-consuming classification models.

These patterns advocate the superiority of atomic features. To select the best category among them, we observe that there are minor differences in terms of recall: the minimum is lower than the maximum one by just 5.1% (D_1), 6.7% (D_2) and 2.8% (D_3). For precision and PGR, we observe that the use of all atomic features consistently achieves the best performance. The boundary-, grid- and candidate-based features underperform by more than 5% in practically all cases. The area-based is the second best feature category, reducing the maximum precision and PGR by just 2.2% and 3.1%, on average, respectively. Regarding the time efficiency, using all atomic features almost doubles the training time of the area-based ones. However, this situation is reversed in the case of the prediction time, the bottleneck of the training phase, which is at least an order of magnitude higher than the training time: all atomic features are faster than the area-based ones by 7.3%, on average.

For these reasons, *we exclusively couple Supervised Scheduling with all atomic features in the following*.

4.2.2 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 Scheduling 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 time

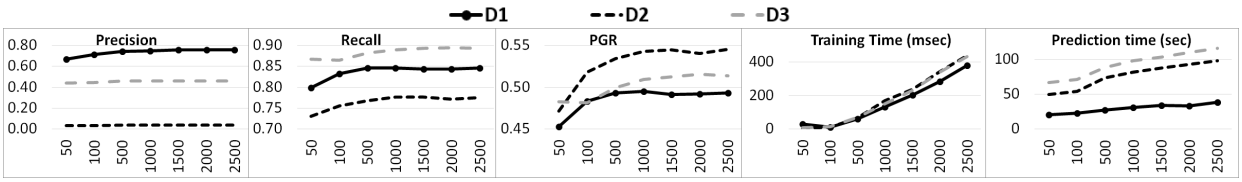

Figure 4.2: Evolution of average precision, recall, PGR, training and prediction time over D_1-D_3 when combining all atomic features with Naive Bayes, Random Forest, Logistic Regression and Bayesian Networks w.r.t. class size (on the horizontal axis).

required for building the training set is reduced.

To assess the impact of these two parameters, we performed a series of experiments over D_1-D_3, assuming that the labels of all candidates pairs are available. For the training set size, we consider six values: 50, 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 of five repetitions in Figure 4.2.

We observe that for each dataset, the effectiveness improves substantially for up to 500 labelled instances per class, but remains practically stable for larger training sets. In particular, precision, recall and PGR raise by 9.3%, 4.0% and 7.7%, respectively, on average, across all datasets, when increasing the training set from 50 to 500 instances per class. From 500 to $2,500$ labelled instances, these measures raise by at most 1.1%, 0.8% and 1.9%, respectively. Given that the training and the prediction time increase linearly with the size of the training set (due to the higher complexity of the learned models), we can conclude that 500 *labelled instances per class offer the best trade-off between effectiveness and time efficiency, minimizing the run-time for high and robust performance*.

4.2.3 Algorithm Selection

Table 4.2 reports the performance of the individual classification algorithms over the entire datasets D_1-D_3, when combined with all atomic features and 500 labelled instances per class. For every evaluation measure, we consider the average and the standard deviation per algorithm after 5 iterations.

Table 4.2: Performance per classification algorithm.

		Precision	Recall	PGR	t_r (ms)	t_p (sec)
D_1	NB	0.744±0.014	0.840±0.008	0.488±0.010	5±3	24±0.2
	RF	**0.818**±0.009	0.841±0.006	0.476±0.006	181±3	67±1.1
	LR	0.668±0.007	**0.863**±0.005	**0.520**±0.003	59±4	3±0.1
	BN	0.745±0.014	0.840±0.009	0.489±0.009	4±1	16±0.2
D_2	NB	0.034±0.003	0.740±0.020	0.507±0.023	4±0	59±1.5
	RF	**0.047**±0.001	0.783±0.009	0.548±0.006	221±5	188±4.3
	LR	0.035±0.001	**0.808**±0.010	**0.573**±0.010	39±7	6±0.1
	BN	0.034±0.003	0.742±0.020	0.509±0.025	4±0	40±0.5
D_3	NB	**0.474**±0.003	0.851±0.011	0.462±0.007	3±0	76±1.7
	RF	**0.474**±0.007	0.875±0.013	0.520±0.008	201±9	217±4.0
	LR	0.413±0.005	**0.950**±0.015	**0.553**±0.015	74±5	8±0.3
	BN	**0.474**±0.003	0.851±0.011	0.462±0.007	3±0	51±1.7

Regarding effectiveness, we observe that Logistic Regression (LR) consistently exhibits very low precision, but achieves the highest recall and PGR in all cases. The opposite

is true for Naive Bayes (NB), Random Forest (RF) and Bayesian Networks (BN), i.e., they emphasize precision at the cost of significantly lower recall and PGR – their relative performance depends on the dataset, except that NB and BN exhibit an almost identical effectiveness in all cases.

Regarding time efficiency, RF is by far the slowest algorithm with respect to training time (t_r); NB and BN are the fastest approach, with LR lying in the middle of these two extremes. For the prediction time (t_p), RF remains the most time-consuming approach, with LR being the most efficient one ~ 27 times faster, on average, than RF.

Overall, LR underperforms with respect to precision and training time, but excels in all other performance measures. Given that Problem 2 emphasizes PGR, *Logistic Regression constitutes the best choice among the four probabilistic classification algorithms*.

4.2.4 Comparison to state-of-the-art

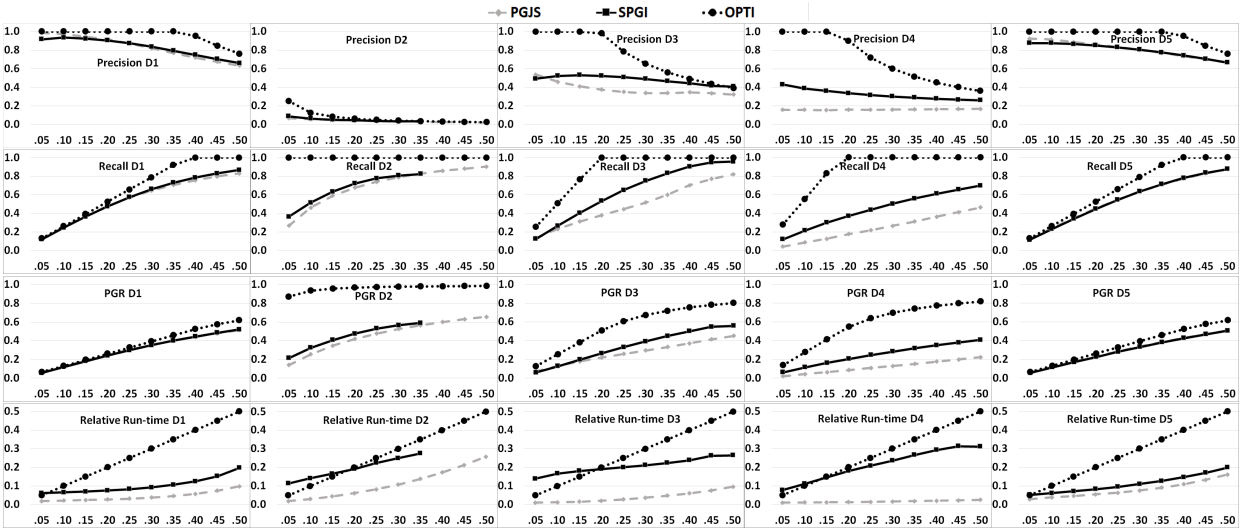

Figure 4.3: Performance of Supervised Progressive GIA.nt (SPGI), Progressive GIA.nt with JS (PGJS) and the optimal approach (OPTI) over all datasets in Table 4.1 using as budgets all portions of candidate pairs in $[0.05, 0.50]$ with a step of 0.05.

Using all atomic features, 500 labelled instances per class and Logistic Regression for learning the probabilistic classification model, we compare Supervised Progressive GIA.nt (SPGI) with the best learning-free progressive algorithm from [14]: Progressive GIA.nt in combination with the Jaccard similarity weighting scheme (PGJS). In essence, this algorithm considers exclusively feature F18, verifying the top-BU weighted pairs in decreasing order. We additionally report the performance of the optimal progressive algorithm (OPTI), which verifies all topologically related pairs before the non-related ones.

For every dataset in Table 4.1, we report the performance with respect to all evaluation measures (see the homonymous paragraph above) for all budgets in the interval $[0.05 \cdot |C|, 0.50 \cdot |C|]$ with a step of 0.05 (recall that $|C|$ denotes the set of candidate pairs). The results appear in Figure 4.3.

Looking into effectiveness, we observe the following patterns:

- For D_1 and D_5, our supervised approach achieves practically equivalent performance with the learning-free baseline method – their average difference is less than

2% for all three measures. Both methods are very close to the optimal approach, with their average difference being $\sim 15\%$ for all measures. Despite the heterogeneous types of geometries in D_1 (Polygons and LineStrings) and D_5 (LineStrings), this should be attributed to the relatively large portion of qualifying pairs: in both cases, 38.1% of all candidate pairs are topologically related.

- For D_3 and D_4, SPGI outperforms PGJS to a significant extent. Its precision and recall is higher by $\sim 20\%$ ($\sim 48\%$), on average, across all budgets in D_3 (D_4). For these measures, our approach actually lies in the middle of PGJS and OPTI, as its average distance from the optimal precision and recall is $\sim 28\%$ and $\sim 50\%$, resp. The same applies to PGR over D_4, where the average difference is $\sim 50\%$ between SPGI and PGJS and $\sim 57\%$ between SPGI and OPTI. In D_3, the PGR of our approach is higher than PGJS by 14.5%, on average, due to their minor differences in the smaller budgets.

  This is true in all datasets: the first verifications of Supervised Progressive GIA.nt target both positive and negative instances in order to build the training set for its probabilistic classifier. This results in lower PGR for small budgets, which is compensated by the high performance of the learned model in subsequent verifications.

  Note that despite the heterogeneity of D_3 and D_4 (Polygons and LineStrings-Polygons, respectively), their similar patterns should be attributed to the similar portions of qualifying pairs: 19.6% and 18.4% of all candidate pairs are topologically related, respectively.

- In D_2, SPGI achieves higher precision, recall and PGR than PGJS by 8.7%, 8.1% and 14.7%, respectively. However, it does not apply to the last three budgets, because it classifies $\sim 62\%$ of the geometry pairs as unlikely related. This should be expected, due to the heavy class imbalance [10], given that just 1.3% of the candidate pairs is qualifying. As a result, the random sampling (Lines 12 and 27-29 in Algorithm 1) cannot label a sufficient number of positive instances: for $D = 15,000$, only 180 topologically related pairs were detected – in all other datasets, $\sim 1,500$ verifications suffice for yielding a balanced training set. This means that *in cases with imbalanced training sets, Supervised Progressive GIA.nt should be combined only with small budgets.*

Finally, regarding time efficiency, we report the relative run-time of SPGI and PGJS with respect to GIA.nt [14], i.e., a batch algorithm that uses the same indexing and verifies all candidate pairs. Given the varying sizes of the datsets and the increasing size of the budgets we consider, the corresponding diagrams in Figure 4.3 are equivalent to a scalability analysis. The optimal baseline method corresponds to a linear increase in the run-time as the budget increases.

We observe that both SPGI and PGJS outperform this baseline, as *their run-time scales sublinearly with the increase of the verified pairs*. The reason is that both algorithms favor pairs with small and simple geometries, whose verification is rather efficient. This is especially true for PGJS, which is consistently the fastest approach. The more effective is SPGI in comparison to PGJS, the higher is their difference in run-time, which indicates that the simple geometries selected by PGJS are not sufficient for achieving high PGR. Note that the overhead of Supervised Scheduling (i.e., random sampling, the creation of features per candidate and their classification) accounts for less than 10% of the SPGI's overall run-time in all cases.

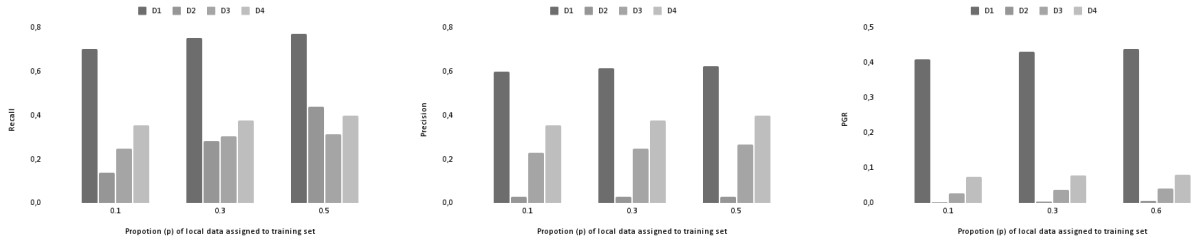

Figure 4.4: Performance of Parallel *SPGI* utilizing dynamic class size selection for the training set over D_1-D_4 datasets, using $BU = 5M$.

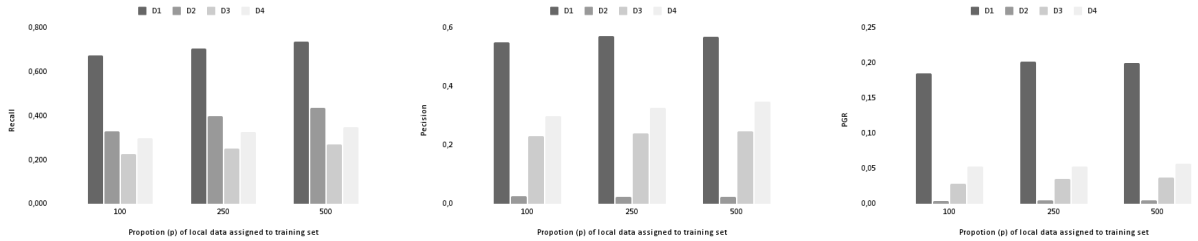

Figure 4.5: Performance of Parallel *SPGI* utilizing static class size selection for the training set over D_1-D_4 datasets, using $BU = 5M$.

4.3 Parallel Processing

4.3.1 Class Size Selection

As per section 4.2.2, we examine the sensitivity of our feature set in the parallel implementation of our approach as well. Here, we consider two cases of class size selection: (i) *dynamic* and (ii) *static*.

In *dynamic* class size selection, the size of the training set is given by a subset of the total source and target geometries lying in each partition. We refer to the total source and target geometries of each partition as *local geometries*. For our experiments we define the proportion of local geometries $p \in \{0.1, 0.3, 0.5\}$. Note that setting a *dynamic* class size selection means that each partition ends up with a different size of a training set, resulting in different workloads for each partition during preprocessing.

In *static* class size selection, the training set size is the same for each partition, regardless of the amount of local geometries. For this case we consider three values: 100, 250 and 500 instances per class.

We perform our experiments over D_1 - D_4. Figures 4.4 and 4.5 illustrate the performance of parallel *SPGI* in respect to *recall*, *precision* and *PGR*, while Figure 4.6 presents the growth of *scheduling time* (t_s) as the test size increases, following the aforementioned two approaches. In every case the training set is balanced, since undersampling is applied.

From our experiments we can deduct the following:

- In *dynamic* class size selection, as p increases, so does the performance of parallel *SPGI*. More precisely, recall, precision and PGR are increased by 12%, 2.6% and $1,3\%$ respectively, as the train set size increases from 10% of local geometries to 50%. It is worth mentioning, that the highest increase in performance occurred when

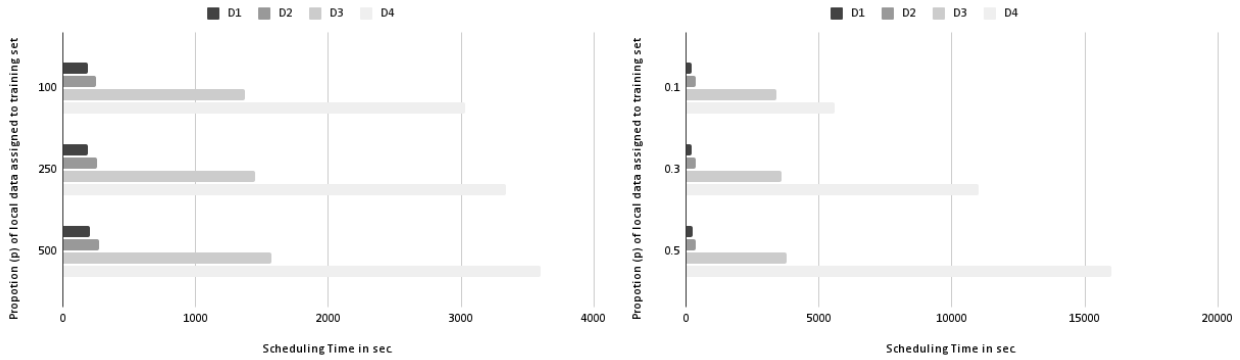

Figure 4.6: Evolution of t_s in Parallel *SPGI* utilizing dynamic class size selection (left) and static class size selection (right) for the training set over D_1-D_4 datasets, using $BU = 5M$.

D_2 was utilized. Since D_2 is a highly imbalanced dataset, it is expected that a larger amount of data assigned to the train set, will have such an impact in performance. Moreover, alongside with performance, the scheduling time also increases. This is dues to the fact that, the Scheduling phase includes the construction of a balanced train set as well as the training procedure itself.

- The same applies to *static* class size selection approach. In more detail, we detect 6%, 2% and $0,7\%$ increase in recall, precision and PGR as the number of instances formulating the train set grows from 100 to 500. Similarly, again, the scheduling time increases as the size of train set increases gradually.
- Comparing the two approaches we observed that *dynamic* class size selection hands in the best results in respect to our metrics for all datasets, however it appears to result in a higher scheduling time. This behavior is expected since each Executor processes a different amount of training data, as the number of geometries varies in each partition. The differences in scheduling time become more significant when a large dataset as $D3$ or $D4$ is processed. Nevertheless, for all of the datasets, even though *dynamic* class size selection has better performance, the difference between the two approaches is not significant.

Therefore, *we conclude that the size of the training set should be the same in each partition. In the following experiments we set the size of each local train set to* 500.

4.3.2 Scalability Analysis

We perform scalability analysis of our approach in terms of monitoring the ability of our system to handle the growth of (i) *computing resources* and (ii) *workload*.

Regarding the first one, we examine how the overall time of interlinking, using Supervised Scheduling, scales as we increase the number of available processing units. Figure 4.7, presents our findings over dataset D_3 with a budget BU = 5M. We observe that *SPGI* scales almost linearly and close to the ideal speedup. A small downshift only happens as we increase the available cores from 8 to 16. As the experimental analysis of Progressive methods reveals in [13, 14], this is mostly due to the redistribution of geometries and the computation of the tiles' granularity, as they both invoke data shuffling. On the other hand, during Supervised Scheduling no data shuffling is needed all necessary information

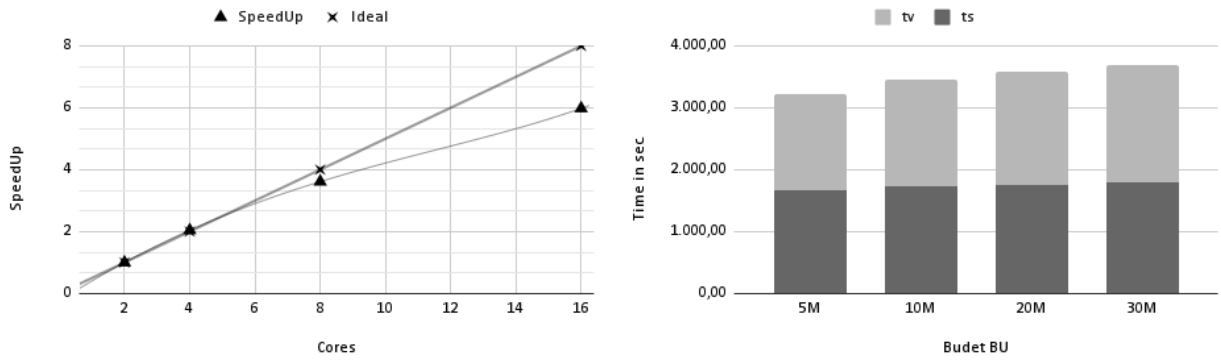

Figure 4.7: Scalability of *Parallel SPGI* in respect to the available preprocessing units (left figure) and the workload (right figure) over D_3 and D_1 respectively.

is locally available to each Executor. Overall, the total interlinking time of *SPGI* from 261.95 minutes using 2 cores, to 43.80 minutes using 16 cores.

To inspect the scalability *SPGI* in respect to the workload, we kept a constant number of the processing units set to 16 cores, while increasing the budget $BU \in \{5M, 10M, 20M, 30M\}$, and therefore increasing the size of the job. This experiment is performed on D_1 dataset. Here, we report the total scheduling t_s and verification time t_v of the overall interlinking process. We observe that t_s slightly increases as the BU increases. This is due to the fact that more pairs are inserted to the PQ as the BU grows. Regarding t_v, it increases more considerably. This is expected as the size of the BU is decisive for the duration of the Verification step.

4.3.3 Comparison with the state-of-the-art

We now compare the performance of Parallel *SPGI* with the different weighting schemes utilized by learning-free progressive approaches. More precisely, we compare *Supervised Scheduling (SS)* with:

- *Co-occurrence Frequency (CF)*, which counts the tiles shared by source s and target t geometries, i.e $CF = |B_s \cap B_t|$, where B_k stands for the set of tiles/blocks containing geometry k.
- *Jaccard Similarity (JS)* that normalizes the overlap similarity defined by *CF*, i.e. $JS(s,t) = \frac{|B_s \cap B_t|}{B_s + B_t - |B_s \cap B_t|}$, capturing the idea that the portion of tiles shared by two geometries is proportional to the likelihood that they satisfy a positive topological relation.
- *Pearson's χ^2 test (χ^2)*, extends *CF* by assessing whether two geometries s and t appear independently in the set of tiles. To infer their dependency, it estimates whether the distribution of tiles intersecting s in B is the same as the distribution if we exclude the tiles that intersect t.

For every dataset in Table 4.1, we report the performance with respect to all evaluation measures presented in above experiments and a budget $BU = 5M$ verifications. We also report the total interlinking time in seconds for every approach. The results appear in Table 4.3.

Table 4.3: Performance of Parallel Supervised Scheduling in comparison to other Parallel Progressive methods, across all datasets with BU = 5M.

		CF	JS	χ^2	SS
D1	Recall	0.776	0.950	0.968	0.790
	Precision	0.373	0.456	0.455	0.619
	PGR	0.339	0.650	0.647	0.449
	time (sec)	498	498	498	328
D2	Recall	0.764	0.838	0.808	0.582
	Precision	0.023	0.025	0.024	0.026
	PGR	0.578	0.545	0.505	0.008
	time (sec)	408	408	408	495
D3	Recall	0.244	0.458	0.460	0.367
	Precision	0.186	0.349	0.350	0.282
	PGR	0.122	0.267	0.268	0.137
	time (sec)	693	693	693	1,959
D4	Recall	0.362	0.159	0.155	0.404
	Precision	0.362	0.159	0.155	0.404
	PGR	0.192	0.083	0.082	0.082
	time (sec)	1,517	1,517	1,517	3,648
D5	Recall	0.195	0.904	0.918	0.632
	Precision	0.195	0.904	0.918	0.632
	PGR	0.094	0.452	0.459	0.167
	time (sec)	-	-	-	> 10,000
D6	Recall	0.049	0.144	0.141	0.288
	Precision	0.010	0.030	0.029	0.050
	PGR	0.025	0.065	0.064	0.030
	time (sec)	-	-	-	> 10,000

From our experimental analysis we can conclude that *SS* achieves a close performance in comparison to the learning-free baseline methods. More precisely, for D_1, D_3 and D_5, *SS* lies in the middle of *CF* and the other two weighting schemes, with the exception of precision in the case of D_1. In the latter, *SS* achieves higher precision, meaning that it manages to detect most of the qualifying pairs within BU. For the cases of D_4 and D_6 datasets, *SS* outperforms the rest of the methods to a significant extent. The only case that *SS* cannot manage to achieve as high results, is when utilizing D_2 dataset. This should be attributed to the relatively small amount of related pairs in dataset, given that only 1.3% of the candidate pairs are qualifying. In more detail, *SS* manages to detect only 89711 qualifying pairs out of the 199122 total.

Regarding execution time, *SS* achieves higher run-times compared to the the other progressive methods. That is mainly imputed to the Scheduling step, which *(i)* performs two passes over the target geometries in order to successfully perform feature extraction, *(ii)* constructs a balanced train set and *(iii)* performs the training procedure. This becomes more evident as the size of the dataset grows. However, the reported run-times of parallel *SPGI* are orders of magnitude lower compared to its serial implementation, though an exact comparison cannot be made since the two versions were run on different machines.

Overall, we observe that *SS* is relatively robust to the variety of datasets in contrast to the other progressive methods, which highly depend to the type of relations existing in each dataset. The only case in which *SS* underperforms is when the dataset is unbalanced. However, when combined with small budgets, it can still manage to detect a good amount of related pairs.

Altogether, Supervised Scheduling offers a good trade-off between effectiveness and time efficiency.

5. CONCLUSIONS AND FUTURE WORK

We proposed Supervised Scheduling as a new means of maximizing the throughput (PGR) of Geospatial Interlinking within a specific budget of verifications. This algorithm is incorporated into Supervised Progressive GIA.nt, an end-to-end approach that automatically learns a generic, effective and fast probabilistic classifier. Using five pairs of large, real-world datasets, we experimentally demonstrated that combining Logistic Regression with 16 atomic features and 500 labelled instances per class, randomly selected across all candidate pairs, suffice for achieving high performance under versatile settings.

We also integrated Supervised Progressive GIA.nt on tha system DS-JedAI that runs on top of Apache Spark and is able to perform Geospatial Interlinking at scale. We show that the parallel version of Supervised Progressive GIA.nt achieves much higher scalability and lower run-times compared to the serial version. We compare this version to other parallel progressive approaches and demonstrate its competitiveness.

In the future, we plan to extend our approach to proximity relations, that indicate that two entities are close enough without having physical contact with each other. Such a relation is the nearby relation. Finally, we intend to integrate our approaches with Entity Resolution and Link Discovery techniques that rely exclusively on text.

ABBREVIATIONS - ACRONYMS

GIS	Geographic Information System
OSM	OpenStreetMap
LOD	Linked Open Data
AWS	Amazon Web Services
DE9IM	Dimensionally Extended Nine-Intersection Model
IM	Intersection Matrix
RDD	Resilient Distributed Dataset
HDFS	Minimum Bounding Rectangle
MBR	Hadoop Distributed File System
GIA.nt	Geospatial Interlinking At large
DS-JedAI	Distributed-Spatial Java gEneric DAta Integration
PQ	Priority Queue
BU	Budget
CF	Co-occurrence Frequency
JS	Jaccard Similarity
MBRO	Minimum Bounding Rectangle Overlap
ISP	Inverse Sum of Points
LR	Logistic Regression
NB	Naive Bayes
RF	Random Forest
BM	Bayesian Networks
SS	Supervised Scheduling
SPGI	Supervised Progressive Geospatial Interlinking
PGJS	Progressive GIA.nt in combination with the JS weighting scheme
OPTI	Optimal Progressive Algorithm
PGR	Progressive Geometry Recall

BIBLIOGRAPHY

[1] Spatialhadoop real-world datasets. `http://spatialhadoop.cs.umn.edu/datasets.html`.

[2] 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.

[3] Edward P. F. Chan and Jimmy N. H. Ng. A general and efficient implementation of geometric operators and predicates. In *SSD*, volume 1262, pages 69–93, 1997.

[4] 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.

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

[6] Alishiba Dsouza, Nicolas Tempelmeier, Ran Yu, Simon Gottschalk, and Elena Demidova. Worldkg: A world-scale geographic knowledge graph. In *CIKM*, pages 4475–4484, 2021.

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

[8] Ahmed Eldawy and Mohamed F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *ICDE*, pages 1352–1363. IEEE Computer Society, 2015.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

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

[11] Rushi Longadge and Snehalata Dongre. Class imbalance problem in data mining review. *CoRR*, abs/1305.1707, 2013.

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

[13] George Papadakis, George Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. Static and dynamic progressive geospatial interlinking. *ACM Trans. Spatial Algorithms Syst.*, 8(2), Apr. 2022.

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

[15] 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.

[16] Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. Radon - rapid discovery of topological relations. In *AAAI*, pages 175–181, 2017.

[17] Panayiotis Smeros and Manolis Koubarakis. Discovering spatial and temporal links among RDF data. In *Workshop on Linked Data on the Web, LDOW*, 2016.

[18] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. Parallel in-memory evaluation of spatial joins. In *SIGSPATIAL*, pages 516–519, 2019.

[19] Mario Villamizar, Oscar Garces, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, and Mery Lang. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS lambda architectures. *Serv. Oriented Comput. Appl.*, 11(2):233–247, 2017.