Supervised Scheduling for Geospatial Interlinking

Maria Despoina Siampou
siampou@usc.edu
Dept. of Computer Science
University of Southern California, USA

George Papadakis
gpapadis@di.uoa.gr
Dept. of Informatics & Telecommunications
National and Kapodistrian University of Athens, Greece

Nikos Mamoulis
nikos@cs.uoi.gr
Dept. of Computer Science
University of Ioannina, Greece

Manolis Koubarakis
koubarak@di.uoa.gr
Dept. of Informatics & Telecommunications
National and Kapodistrian University of Athens, Greece

ABSTRACT

Geospatial Interlinking constitutes a crucial data integration task that associates pairs of geometries with topological relations. Its high computational cost, though, scales poorly to voluminous datasets. Progressive methods were recently proposed to reduce this cost by sacrificing recall to an affordable extent. They 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 extend them with Supervised Scheduling, 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. We integrate Supervised Scheduling into a progressive end-to-end algorithm that automatically labels the required training instances at a low computational cost. 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. Our experiments also verify the superior performance of our approach in comparison to existing learning-free ones over five real, large datasets.

KEYWORDS

Geospatial Interlinking, DE-9IM Relations, Supervised Scheduling

ACM Reference Format:
Maria Despoina Siampou, George Papadakis, Nikos Mamoulis, and Manolis Koubarakis. 2023. Supervised Scheduling for Geospatial Interlinking. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Geospatial data constitutes the cornerstone in numerous applications, especially on the Web. For example, OpenStreetMap alone contains data about the entire globe that amounts to 1.5 terabyte[1]. GeoNames describes more than 12 million locations[2]. There are also *geospatial knowledge graphs* such as YAGO2[3], YAGO2geo[4], WorldKG[5] and KnowWhereGraph[6]. YAGO2 has been built using data from GeoNames, thus focusing on point geometries (latitude and longitude) [10], but YAGO2geo extends it with millions of lines, polygons and multipolygons, drawing on OpenStreetMap and data sources with administrative divisions [12]. WorldKG is built using data from OpenStreetMap and contains around 113.4 million geographic entities [5]. KnowWhereGraph is the latest effort in this area, with pilot applications in disaster relief, agricultural land use and food-related supply chains [11]; in more than 12 billion RDF triples, it includes polygons and multipolygons. Finally, well-known, cross-domain knowledge graphs, such as DBpedia[7] and Wikidata[8], also contain their fair share of geospatial information.

Despite the prominence and volume of geospatial data on the Web as witnessed by the above examples, its sources are inadequately interlinked on the Linked Open Data (LOD) cloud[9]. Although 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 [16]. For example, only 0.52% of the OpenStreetMap geometries were linked to Wikidata as of April, 2021 [5].

To address this shortage, *Geospatial Interlinking* aims to automatically connect the geometric entities (*geometries* in the following) between the various data sources of the (Semantic) Web [19, 21]. 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 of such relations consider the geometries in Figure 1: LineString g_4 `intersects` LineString g_3, which `touches` Polygon g_1, which `contains` Polygon g_2. The topological relations between two geometries can be encoded by the DE-9IM model [3, 4, 6], which essentially builds an intersection matrix based on the relation between the interior, the boundary and the exterior of the two geometries (see Section 3 for more details). The time complexity of

Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]https://wiki.openstreetmap.org/wiki/Planet.osm
[2]https://www.geonames.org/about.html
[3]https://yago-knowledge.org/
[4]https://yago2geo.di.uoa.gr/
[5]https://www.worldkg.org
[6]https://knowwheregraph.org/
[7]https://www.dbpedia.org/
[8]https://www.wikidata.org/
[9]https://lod-cloud.net

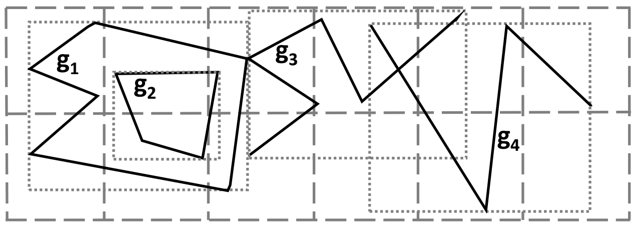

Figure 1: An example of topologically related geometries.

this operation is $O(n \log n)$, where n stands for the total number of boundary points in the two geometries [2].

Geospatial Interlinking is a demanding task that involves two main challenges [19, 21]: (i) its inherently quadratic time complexity, given that the brute-force approach has to examine all pairs of geometries, and (ii) the time-consuming identification of the topological relationship for each pair of geometries.

The first challenge is addressed through the Filtering-Verification framework that lies at the core of all relevant techniques [1, 17, 19, 21]. The source and target datasets, S and T respectively, are fed to the *Filtering* step, which efficiently computes the 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. The Filtering step can efficiently be computed by a *spatial intersection join* algorithm for MBRs (e.g., [24]); C is then forwarded to the *Verification* step, which examines every geometry pair to identify whether the pair has non-trivial topological relations and, if yes, which ones.

If the Verification step is applied to all pairs of geometries in C, Geospatial Interlinking may take too long to complete; for this reason, *progressive approaches* [17, 18] were recently proposed to address the second challenge of Geospatial Interlinking. Such approaches, turn Geospatial Interlinking into an *approximate* procedure, where not all geometry pairs in C are processed, but a subset of C, which is anticipated to include most pairs having non-trivial topological relationships. Hence, progressive approaches sacrifice recall to a small extent to reduce the run-time by orders of magnitude for applications with limited resources. Such applications are typically cloud-based, having a limited budget for services like the AWS Lambda functions [20], which charge whenever used.

To make the most of the applications' budget, progressive methods aim to determine the processing order of candidate pairs such that the topologically related ones take precedence. After Filtering, Scheduling associates every candidate pair with a score that is proportional to the likelihood that its geometries are topologically related. During Verification, only the top-k weighted pairs are examined, with k configured according to the available resources.

On the downside, the existing progressive methods weigh the candidate pairs according to learning-free heuristics that are typically based on the tiles that intersect the MBR of each geometry; e.g., the number of common tiles or their Jaccard similarity [17]. Yet, they prune the search space in a way that might be either too aggressive, missing many related pairs, or too conservative, verifying many non-related pairs. Their performance depends greatly on the weighting scheme and the characteristics of the geometries at hand, when the best scheme for a specific dataset is a-priori unknown.

In this work, we argue that supervised learning provides a principled framework that distinguishes between related and non-related geometry pairs by leveraging multiple sources of evidence, instead of a single weighting scheme. We formalize *Supervised Scheduling* as a probabilistic binary classification task that orders in decreasing classification probability the candidate pairs that are labeled as "`likely related`". To solve this task, we propose 31 features that represent every pair of geometries. These features are generic and able to be applied on both LineStrings and Polygons, while their extraction cost is very low (few milliseconds). Our objective is to identify the smallest set of features that achieves high accuracy. We experimentally verify its robustness, despite using a tiny training set, and examine its sensitivity to the classification algorithm.

We also propose *Supervised Progressive GIA.nt*, an end-to-end algorithm that builds the required training set on the fly, without any human intervention, learns the classification model for Supervised Scheduling, and uses it to feed Verification with the best candidates. Extensive experiments show that this approach significantly outperforms the existing progressive methods.

Overall, this paper makes the following contributions:

• We define Supervised Scheduling, a probabilistic binary classification task trading slightly lower recall for much higher precision.

• We define four categories of generic, efficient, and effective classification features, with each one further divided into two subcategories, and populate them with 31 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 candidate pairs and the run-time.

• We perform feature selection and demonstrate the robustness of the resulting features when trained over a few instances.

• We experimentally compare our approach to the existing learning-free progressive methods, demonstrating its superiority in terms of effectiveness and robustness.

2 RELATED WORK

Geospatial Interlinking is a core task in the process of populating the Semantic Web with links between its geospatial entities [1, 17, 21].

The first relevant technique is Silk-spatial [22]. Its Filtering divides the surface of the Earth into a user-defined number of tiles. Its Verification examines the candidate pairs inside every tile in parallel, leveraging Apache Hadoop (http://hadoop.apache.org).

To go beyond Silk-spatial, RADON [21] enhanced Filtering with a fine-grained Equigrid, whose dimensions depend on data characteristics. Every geometry is placed in all tiles that intersect its MBR and, thus, some candidate pairs co-occur in different tiles. The Verification step stores in a main-memory hash-map all pairs examined so far for deduplication purposes, i.e., in order to avoid verifying the same pair more than once.

stLD [19] enhances RADON in four ways: (i) its Filtering supports a series of indices, such as R-Tree, Equigrid and a hierarchical grid; (ii) Apache Flink (https://flink.apache.org) is used for massive parallelization; (iii) MaskLink [19] estimates the overlap of every geometry with every tile to check whether it is contained in the space left empty by the other geometries in the tile. If this is true, the entire tile is skipped, without generating any candidate pairs; (iv)

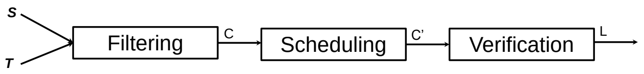

Figure 2: Learning-free Progressive Geospatial Interlinking.

only one of the two input datasets is indexed, reducing the memory requirements and treating the second input dataset as a stream of geometries. This means that only one dataset is loaded in the main memory, while the second can be read from the disk on-the-fly.

These works operate at the level of a single topological relation, repeating the entire processing for every topological relation of interest. This is addressed by RADON2 [1], which simultaneously extracts all relations from the intersection matrix of two geometries.

GIA.nt [17] combines the advantages of all the above works. During Filtering, it loads only the smallest dataset D_S in main memory and indexes it with an Equigrid, whose granularity depends on the characteristics of D_S. 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 (http://spark.apache.org) minimizes the run-time.

The above approaches operate in a batch manner that produces exact 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 [17], turning Geospatial Interlinking into an approximate process that promotes precision at the expense of recall. As shown in Figure 2, 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 performs a global sorting so as to verify the top-k weighted pairs. In contrast, Progressive RADON orders the tiles according to their size and performs a local sorting of geometry pairs inside every tile, until consuming the budget. None of them leverages machine learning to enhance the effectiveness and robustness of Geospatial Interlinking.

Both Progressive GIA.nt and Progressive RADON yield a static processing order, which is not affected by the results of Verification. This order can become dynamic through the algorithm presented in [18]: whenever two geometries s and t are detected as topologically related, the weight w of all non-verified candidate pairs that include s or t, is updated according to the following formula: $w' = w \times (1+q)$, where q is the number of times a geometry of this candidate pair has been qualified as topologically related. This is useful in specific cases, such as when interlinking a dataset with long LineString geometries like roads or rivers with another one that entails Polygon geometries like buildings or cities; the more buildings a road has touched, the higher the weight of the rest of the candidate buildings should be, denoting its likelihood to be a main road. This means that Dynamic Scheduling has a limited scope and, thus, we disregard it in this work.

Note also that (batch) Geospatial Interlinking seems similar to a spatial join [15], but in reality, the latter addresses only the Filtering step of the former since it aims to detect nearby points or pairs of geometries that intersect. Instead, the Verification step of Geospatial Interlinking aims to detect a series of topological relations, as explained in Section 3. There are no spatial join algorithms that perform Verification in a pay-as-you-go manner, i.e., by scheduling the processing order of geometry pairs. The progressive computation of spatial joins has only been examined in the context of stream processing [13, 23], which is irrelevant to this work.

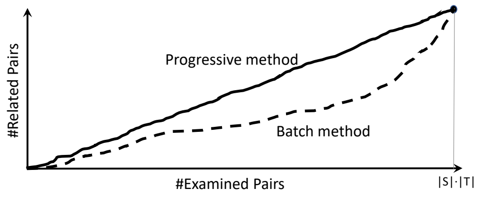

Figure 3: PGR for batch and progressive algorithms.

3 PRELIMINARIES

Our work focuses on two types of geometries: (i) *LineStrings* or *Polylines*, which are sequences of connected line segments, and (ii) *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, examples of LineStrings are geometries g_3 and g_4, while Polygons are represented by g_1 and g_2. Both types of geometries consist of three parts: (i) the interior, (ii) the boundary, and (iii) the exterior (i.e., the rest of the points).

For these two types of geometries, the *Dimensionally Extended Nine-Intersection Model* (DE-9IM) [3, 4, 6] defines the following topological relations between two geometries A and B:

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

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 [1, 17].[10] We disregard, though, the relation `Disjoint`, because it is not informative, as it typically applies to the vast majority of geometry pairs [17]. 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`.

Overall, Geospatial Interlinking is an exact task that misses no non-trivial topological relations between the given geometries [17]:

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

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

3.1 Progressive Geospatial Interlinking

An approximate solution to Problem 1 is provided by progressive algorithms, which operate in a pay-as-you-go manner [17] to maximize the throughput of applications with limited resources. Compared with the exact (batch) solutions to Problem 1, the progressive ones have two goals [17]: (i) to yield the same outcome when processing all input data, and (ii) to yield substantially more topologically related pairs, when terminating prematurely.

These requirements are reflected in the diagram of Figure 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 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 methods tackle this task [17]:

Problem 2 (Progressive Geospatial Interlinking). *Given a source and a target dataset, S and T respectively, 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 PGR 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 verified ones, and (iii) *recall*, the ratio between the detected and the existing related pairs.

3.2 (Supervised) Scheduling

The experiments in [17, 18] demonstrate that the global sorting of Progressive GIA.nt consistently outperforms the local one of Progressive RADON. For this reason, our approach to Supervised Scheduling employs a global sorting, too.

Also crucial for the performance of progressive methods is the weighting scheme that is used in Scheduling. Every scheme produces positive values that are proportional to the likelihood that two geometries s and t are topologically related. More specifically:

W1) Co-occurrence frequency (CF) measures the number of tiles shared by s and t [17].

W2) Jaccard similarity (JS) normalizes CF by the number of distinct tiles containing s and t [17].

W3) Pearson's χ^2 test (χ^2) checks whether the distribution of tiles intersecting s remains the same if we exclude the tiles intersecting t, and vice versa. That is, it assesses whether the two geometries appear independently in the tiles of the grid index based on the number of tiles intersecting s, t and both [17].

W4) MBR overlap (MBRO) is the portion of the area shared by the MBRs of s and t: $MBRO(s,t){=}|MBR(s{\cap}t)|/MBR(s{\cup}t)|$ [18].

W5) Inverse sum of points (ISP) assigns higher scores to pairs with a lower sum of boundary points, i.e., $ISP(s, t) = 1/(b(s){+}b(t))$, where $b(x)$ is the function returning the number of boundary points for geometry x [18].

The best performing weighting scheme depends on the dataset at hand and the characteristics of its geometries (e.g., the type of topological relations they satisfy) [17, 18]. Our goal is to use these schemes along with additional evidence as features that represent the geometry pairs fed to a probabilistic classifier. We argue that sorting the candidate pairs in decreasing matching probability consistently outperforms the individual schemes. We also demonstrate that this can be achieved without requiring any human intervention for the training or the configuration of the probabilistic classifier.

Scheduling in Figure 2 operates in a learning-free manner that associates every pair of geometries with a single heuristic score [17]. We argue that this is not sufficient for addressing Problem 2. Instead, we introduce Supervised Scheduling as the task of associating every pair with a probability that its constituent geometries are topologically related through a principled approach. More formally:

Problem 3 (Supervised Progressive Geospatial Interlinking). *Given a source and a target dataset, S and T respectively, along with a budget BU on the maximum calculations (or running time), learn a probabilistic binary classifier that associates every geometry pair (s, t) with the likelihood that s and t are topologically related so that ordering all pairs in decreasing weight maximizes PGR within BU.*

Our goal is to address Problem 3 without any human intervention while improving the effectiveness of progressive algorithms.

4 APPROACH

4.1 Features for Supervised Scheduling

Supervised Scheduling essentially associates every geometry pair with a feature vector, where every dimension is a separate numerical score. The desiderata of these features are: (i) They should be *generic*, applying seamlessly to LineStrings and Polygons and ideally, to any indexing scheme used by the Filtering step in Figure 2. (ii) They should be *effective* with high discriminatory power. (iii) They should be *efficient*, involving a low extraction cost so that the classification of a pair is much faster than its verification.

In this context, we propose 31 features for Supervised Filtering. To facilitate their description and understanding, we organize them into four complementary categories: (i) The *area-based features* consider the space occupied by the MBR of each geometry. (ii) The *boundary-based features* stem from the characteristics of each geometry's boundary. (iii) The *grid-based features* emanate from Filtering's index. (iv) 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 [17], which builds an Equigrid, whose dimensions correspond to the average width and height of the source geometries.

Every category includes two types that allow for exploring the impact of feature complexity on Supervised Filtering: (i) *atomic features*, which pertain to core characteristics of a single geometry, and (ii) the *composite features*, which encompass combinations of atomic features that typically normalize their values in $[0, 1]$, with higher values implying a stronger likelihood for topological relatedness. Next, we delve into the features per category and type.

Area-based features. To be generic, this category considers the area occupied by the MBR of a geometry – the area occupied by the geometry itself does not apply to LineStrings, where the interior coincides with the boundary. The atomic features are:
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 its candidates. 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), i.e., $W4$ in Section 3.2.

We assume that these features produce scores proportional to the likelihood that two geometries are topologically related.

Boundary-based features. This category includes the two atomic features per geometry that characterize the border of LineStrings and Polygons, i.e., their points, and their length:
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 are normalized measures of complexity, expressing the average 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.

Grid-based features. Using GIA.nt's Filtering, a uniform grid (Equigrid) 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, i.e., $W1$ in Section 3.2.

We assume that all these features are proportional to the likelihood that a geometry (pair) is topologically related. The same applies to the composite features, which 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), i.e., $W2$ in Sec. 3.2.
F19) Pearson's χ^2 test [17], i.e., $W3$ in Section 3.2.

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 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. This applies to the composite features, too:
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

4.2 Supervised Progressive GIA.nt

We now describe the algorithm that implements the pipeline in Figure 2, by replacing Scheduling with Supervised Scheduling. Following GIA.nt [17], Algorithm 1 first indexes the smallest input dataset, i.e., the source dataset (Lines 2-4). The dimensions of the grid cells are specified in Line 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 together with Δ_x and Δ_y determine the tiles that intersect $MBR(s)$ and should contain s in Line 3. Assuming $\Delta_x = 4$, $\Delta_y = 3$ and 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 $(20, 93)$; this geometry participates in the tiles with $\lfloor 16/\Delta_x \rfloor = 4 \leq i \leq 5 = \lceil 20/\Delta_x \rceil$ and $\lfloor 90/\Delta_y \rfloor = 30 \leq j \leq 31 = \lceil 93/\Delta_y \rceil$. Index I is ready after Line 4.

The training of the probabilistic binary classification model is carried out in Lines 5-32. The first step is the sampling of the geometry pairs that will form the training set. No ground-truth is required because Geospatial Interlinking involves an exact verification function that always decides with perfect accuracy whether two geometries are topologically related or not. Therefore, the goal of sampling is to pick the geometry pairs that will be verified and, thus, labeled during this stage. Two requirements should be met by this process: (i) The considered pairs should involve geometries with intersecting MBRs, because pairs with disjoint MBRs are ignored during Supervised Scheduling. (ii) The selected pairs should be representative of all geometry pairs with intersecting MBRs.

Geospatial Interlinking is a (heavily) imbalanced classification task, because the disjoint geometry pairs usually outnumber the topologically related ones to a significant extent (see Table 1). The latter requirement is thus addressed in one of the following ways [9, 14]: (i) *oversampling*, which randomly resamples the minority class until both classes have the same size, (ii) *undersampling*, which randomly samples a subset of equal size from both classes, (iii) *cost-sensitive learning*, which trains a classifier with a high misclassification cost for the minority class, or (iv) *ensemble learning*,

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  $I \leftarrow \{\}$; $(\Delta_x, \Delta_y) \leftarrow$ defineIndexGranularity($S$);
2  foreach geometry $s \in S$ do                                    // filtering
3      $I$.addToIndex($s$);
4  end
5  $D \leftarrow |S| \times |T|$; $sourceStats \leftarrow \{\}$; $id \leftarrow 0$; $sample \leftarrow \{\}$;
6  $sampleIds \leftarrow$ randomGenerator($m$,$D$);
7  foreach geometry $t \in T$ do                                    // first pass
8      $C_S \leftarrow \{\}$ ;                  // the set of source candidates
9      $(x_1(t), y_1(t), x_2(t), y_2(t)) \leftarrow$ getDiagCorners($t$);
10     for $i \leftarrow \lfloor x_1(t) \cdot \Delta_x \rfloor$ to $\lceil x_2(t) \cdot \Delta_x \rceil$ by 1 do
11         for $j \leftarrow \lfloor y_1(t) \cdot \Delta_y \rfloor$ to $\lceil y_2(t) \cdot \Delta_y \rceil$ by 1 do
12             $C_S$.add($I$.getTileContents($i$, $j$));
13         end
14     end
15     foreach geometry $s \in C_S$ do
16         $sourceStats \leftarrow$ updateTotalDistinctPairs($s$);
17         if intersectingMBRs($s$, $t$) then
18             $sourceStats \leftarrow$ updateRealPairs($s$);
19             if $sampleIds$.contains($id$) then $sample$.add($\{s, t\}$) ;
20             $id \leftarrow id + 1$;
21         end
22     end
23 end
24 $negPairs \leftarrow \{\}$; $posPairs \leftarrow \{\}$; $shuffle$(sample);
25 foreach pair $\{s, t\} \in sample$ do                            // labelling
26     $isRelated \leftarrow$ verifyPair($\{s, t\}$);
27     if $isRelated$ then $posPairs$.add($\{s, t\}$) ;
28     else $negPairs$.add($\{s, t\}$) ;
29     if $N \leq |posPairs|$ & $N \leq |negPairs|$ then break ;
30 end
31 $L \leftarrow$ getFeatures($posPairs \cup negPairs$,$F$,$sourceStats$,$I$);
32 $\mathcal{M} \leftarrow$ train($L$); $L_{\mathcal{R}} \leftarrow \{\}$; $min_w$=0; $T_C \leftarrow \{\}$;
33 foreach geometry $t \in T$ do                                   // second pass
34     ... ;                                     /* Same as Lines 9-14 */
35     foreach geometry $s \in C_S$ do
36         if intersectingMBRs($s$, $t$) then
37             $v \leftarrow$ getFeatureVector($s$, $t$, $F$);
38             $w_{s,t} \leftarrow \mathcal{M}$.getClassificationProbability($v$);
39             if $min_w < w_{s,t}$ then
40                 $T_C$.add($\{s, t\}$, $w_{s,t}$);
41                 if $BU < T_C$.size() then
42                     $min_w = T_C$.pop().getWeight();
43                 end
44             end
45         end
46     end
47 end
48 while $T_C \neq \{\}$ do                                        // verification
49     $tail = T_C$.popLast();
50     $IM \leftarrow$ verify($tail.s$, $tail.t$);
51     $L_{\mathcal{R}} \leftarrow L_{\mathcal{R}} \cup IM$.getRelations();
52 end
53 return $L_{\mathcal{R}}$;
```

which trains several classifiers such that they collectively label every instance. Oversampling may yield very large training sets that foster overfitting, due to the repetition of the minority class instances, while cost-sensitive and ensemble learning may produce complex, time-consuming classification models. Undersampling allows for minimizing the training and prediction time, leveraging small training sets that learn simple, fast, albeit effective classifiers. Hence, *Supervised Progressive GIA.nt relies on undersampling.*

Using undersampling, the representative training set is created by the *randomGenerator* function in Line 6, which requires two arguments: (i) the maximum number m of pairs to be verified/labelled during training, which ideally is $2 \cdot N$, where N is the input parameter that specifies the required number of labelled instances per class. In practice, though, m is two orders of magnitude larger than N, i.e., $m = 100 \cdot N$, to make up for class imbalance in favor of disjoint geometries and for the fact that some of the selected ids will not correspond to pairs with intersecting MBRs (see Exp. 2 in Section 5 for experimentally fine-tuning N). (ii) the total number of geometry pairs with intersecting MBRs, D, which requires checking the MBRs of all candidate pairs resulting after Filtering. Given that this is time-consuming, D is set to the maximum possible range of candidate pairs, i.e., the Cartesian product $|S| \times |T|$ (Line 5). Overall, Line 6 randomly selects m pair ids in the range $[0, |S| \times |T|]$.

Next, for every target geometry t, the tiles that intersect its MBR are inferred from its lower left and upper right MBR points (Lines 9-11). The source geometries participating in these tiles are aggregated into the *set* of candidates C_S (Line 12). Every source geometry appears in C_S just once, but a counter measures its actual frequency across the tiles intersecting $MBR(t)$ (we omit 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 15-16). If $MBR(s)$ intersects $MBR(t)$, feature F22 is updated, too (Lines 17-18). The two geometries are added to the random sample of pairs to be verified if their id is among the selected ones (Line 19).

Subsequently, the sampled pairs are shuffled, to randomize their order (Line 24), because not all of them will be verified. Verification extracts the class labels in Line 26 and terminates in Line 29 as soon as the necessary number of instances is gathered for both classes. Only the first N pairs from each class are taken into account, but we omit this for brevity. The topologically related pairs are added to the positive pairs and the rest to the negative ones (Lines 27-28).

Next, the feature vectors of the sampled pairs are generated in Line 31. Most features rely on characteristics of the geometries, but features F15-F19, F23-F25, and F29-F31 require that Lines 9-14 are repeated for every sampled *target* geometry (we omit this for brevity). The resulting training set L is then fed to the selected algorithm to learn the classification model $\mathcal{M}$ (Line 32).

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 9-14. 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 35-37). The vector is fed to $\mathcal{M}$, which predicts the classification probability for the pair $\{s, t\}$, $w_{s,t}$ (Line 38). 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 39-40) $-min_w$ is updated to the probability of the $(BU + 1)^{th}$ top-weighted pair, whenever the size of T_C exceeds the specified budget (Lines 41-43). Finally, the overall top-BU weighted pairs are verified in decreasing classification probability, adding their topological relations to the set of links $L_{\mathcal{R}}$ (Lines 48-51).

Table 1: Real dataset pairs for Geospatial Interlinking. CP stands for the Cartesian product, $|C|$ for the geometry pairs with intersecting MBRs and $|Q|$ for the topologically related (qualifying) pairs.

	$\mathbf{D_1}$	$\mathbf{D_2}$	$\mathbf{D_3}$	$\mathbf{D_4}$	$\mathbf{D_5}$
S	AREAWATER	AREAWATER	Lakes	Parks	ROADS
T	LINEARWATER	ROADS	Parks	Roads	EDGES
$\|S\|$	2,292,766	2,292,766	8,326,942	9,831,432	19,592,688
$\|T\|$	5,838,339	19,592,688	9,831,432	72,339,926	70,380,191
CP	$1.34 \cdot 10^{13}$	$4.49 \cdot 10^{13}$	$8.19 \cdot 10^{13}$	$7.11 \cdot 10^{14}$	$1.38 \cdot 10^{15}$
$\|C\|$	6,310,640	15,729,319	19,595,036	67,336,808	430,597,631
$\|Q\|$	2,401,396	199,122	3,841,922	12,145,630	163,982,138

The pairs verified in Line 26 are also included in the output $L_{\mathcal{R}}$. A hash map is checked between Lines 36 and 37 to avoid redundant verifications, but we omit these details for brevity.

Overall, Supervised Progressive GIA.nt has the same time complexity as Progressive GIA.nt, $O(|S| + |T| \cdot |\bar{C_S}| \cdot \log |BU| + |BU|)$, 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 (Lines 1-4), the second one to Supervised Scheduling (Lines 5-47), and the last one to Verification (Lines 48-52). The time required by Lines 24-32 to label the sample of candidate pairs and train the classification model, is negligible (cf. Exp. 4 in Section 5). Its space complexity is also equivalent to Progressive GIA.nt, $O(|S| + |BU|)$, given that the space occupied by the learned model and the candidate pairs that are automatically labeled is constant, due to the parameter configuration in Exp. 1-2 in Section 5.

5 EXPERIMENTAL ANALYSIS

Setup. All experiments 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 3.9.6 [8] as the library providing the classification algorithms.

Our experiments rely on publicly available[11], large-scale, real-world datasets that are popular in the literature [7, 17, 24] and involve LineStrings and Polygons. They 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 OpenStreetMap, representing lakes (Lakes), parks (Parks) and roads (Roads) from the whole world. They are combined into the five pairs in Table 1, D_1-D_5, that cover all possible combinations of geometry types: D_1, D_2 and D_4 interlink Polygons with LineStrings, while D_3 and D_5 involve only Polygons and only LineStrings, respectively.

We assess *effectiveness* through precision, recall, and PGR. For *time efficiency*, we consider the run-time per workflow step.

The first three experiments fine-tune the main parameters of Supervised Progressive GIA.nt, i.e., the feature set, the class size, and the classification algorithm. For brevity, they apply to the three smallest dataset pairs, i.e., D_1-D_3. Their conclusions, though, apply to the two largest datasets, too, as shown by the high performance of Supervised Progressive GIA.nt in the fourth experiment.

Exp. 1: Feature Selection. The more features that describe a labeled instance, the more complex and time-consuming is the resulting classifier. To minimize the features of Supervised Scheduling, we analytically experiment with every category and type of feature, considering their effectiveness and time efficiency. These experiments assume that all candidate pairs are labeled.

For each dataset, 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 the average performance across all established probabilistic classifiers offered by Weka: Naive Bayes, Random Forest, Logistic Regression, and Bayesian Networks. We repeated every experiment five times and took the average for every evaluation measure. The resulting performance appears in Figure 4, where the *training* and the *prediction time* capture the time required to learn the classification model and to apply it to the set of candidates with intersecting MBRs, C, resp.

Regarding the effectiveness of the feature types, we observe that the atomic features consistently outperform the composite ones with respect to recall and PGR. On average, across all datasets and feature categories, their recall is higher by 9.6% and their PGR by 10%. 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 mixed results, with the atomic features taking the lead in terms of recall and PGR in at least half the cases and by ~1%, on average, for both evaluation measures. However, the highest precision in practically all cases is achieved by the combination of both feature types, which outperform the atomic features by 8.8%, on average.

Regarding the time efficiency of the feature types, the atomic features are faster than the composite ones by 19.3% and 25.3%, on average, with respect to the training and the prediction time, respectively. This indicates that more complex patterns are learned from the composite features, probably due to their lower discriminativeness. The only exception corresponds to the boundary-based features over D_3, where the composite is an order of magnitude faster than the atomic ones, because of the rather simple classification model, which exhibits much lower effectiveness with respect to all evaluation measures. Finally, the combination of both feature types is slower than the atomic features with respect to training and prediction time by 36.2% and 8.4%, respectively. This is expected because 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 (which also corresponds to the maximum PGR across all feature types and categories in all three datasets). 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, with its precision and PGR lower than the best one by just 2.2% and 3.1%, on average, resp.

Regarding 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, which actually constitutes the bottleneck of Supervised Scheduling, being an order of

[11]https://zenodo.org/record/6384164#.ZFdQrupBxD8

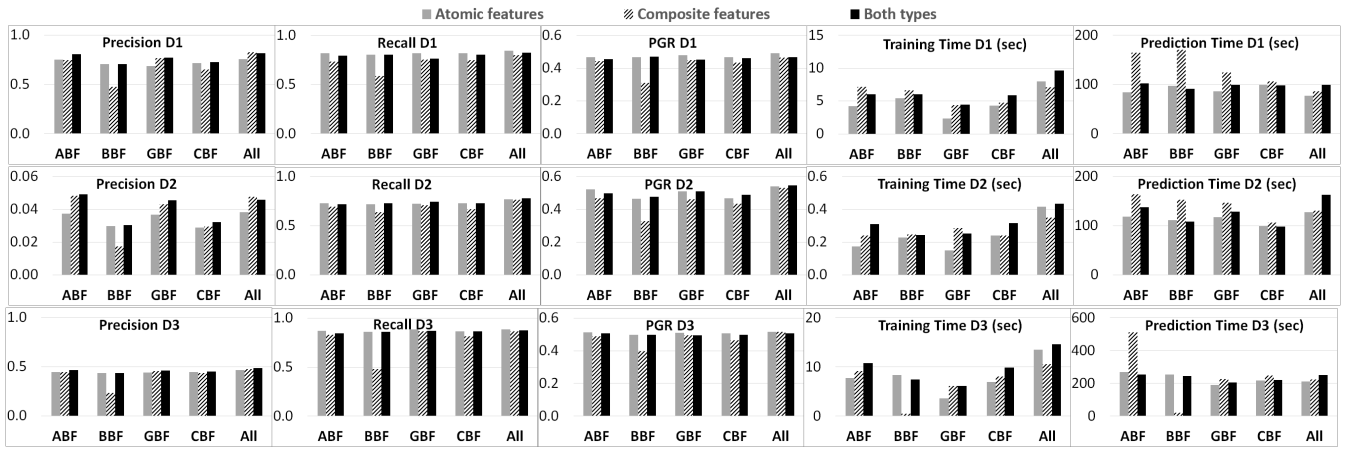

Figure 4: Average performance of area-based (ABF), boundary-based (BBF), grid-based (GBF), candidate-based (CBF) and all (All) features across all classifiers over D_1-D_3. In each case, we consider atomic and composite features as well as their combination.

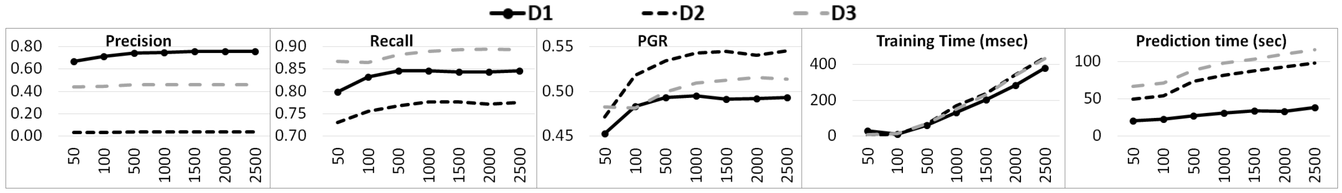

Figure 5: Average precision, recall, PGR, training and prediction time over D_1-D_3 when combining all atomic features with Naive Bayes, Random Forest, Logistic Regression and Bayesian Networks w.r.t. class size (i.e., the input parameter N of Algorithm 1).

magnitude higher than the training time: all atomic features are faster than the area-based ones by 7.3%, on average. This means that combining the atomic features from all categories yields slightly simpler and faster classification models (e.g., shorter decision trees in Random Forest) than relying exclusively on the area-based ones.

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

Exp. 2: Class Size Selection. We now examine how sensitive our feature set with respect to the size of the training set is. The labeled instances are generated automatically, but restricting their number lowers the cost of Supervised Scheduling because: (i) the training time decreases, (ii) the resulting classifier is simpler and, thus, the prediction time is lower, and (iii) the time required for building the training set is reduced.

We experiment with D_1-D_3, assuming that the labels of all candidate pairs are available. We consider seven training set sizes: 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 per algorithm in Figure 5.

We observe that for each dataset, the effectiveness improves substantially for up to 500 labeled instances per class, but remains practically stable for larger training sets. In particular, precision, recall, and PGR rise 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 labeled instances, these measures rise 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 labeled instances per class offer the best trade-off between effectiveness and time efficiency, minimizing the run-time for high and robust performance.*

Table 2: Performance per classification algorithm. The best performance per evaluation measure is highlighted in bold.

		Precision	Recall	PGR	t_r (ms)	t_p (sec)
(a) **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
(b) **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
(c) **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

Exp. 3: Algorithm Selection. Table 2 reports the average performance after 5 iterations per classification algorithm over D_1-D_3, when using all atomic features and 500 labeled instances per class.

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 approaches, with LR lying in the middle of these two extremes. For the prediction

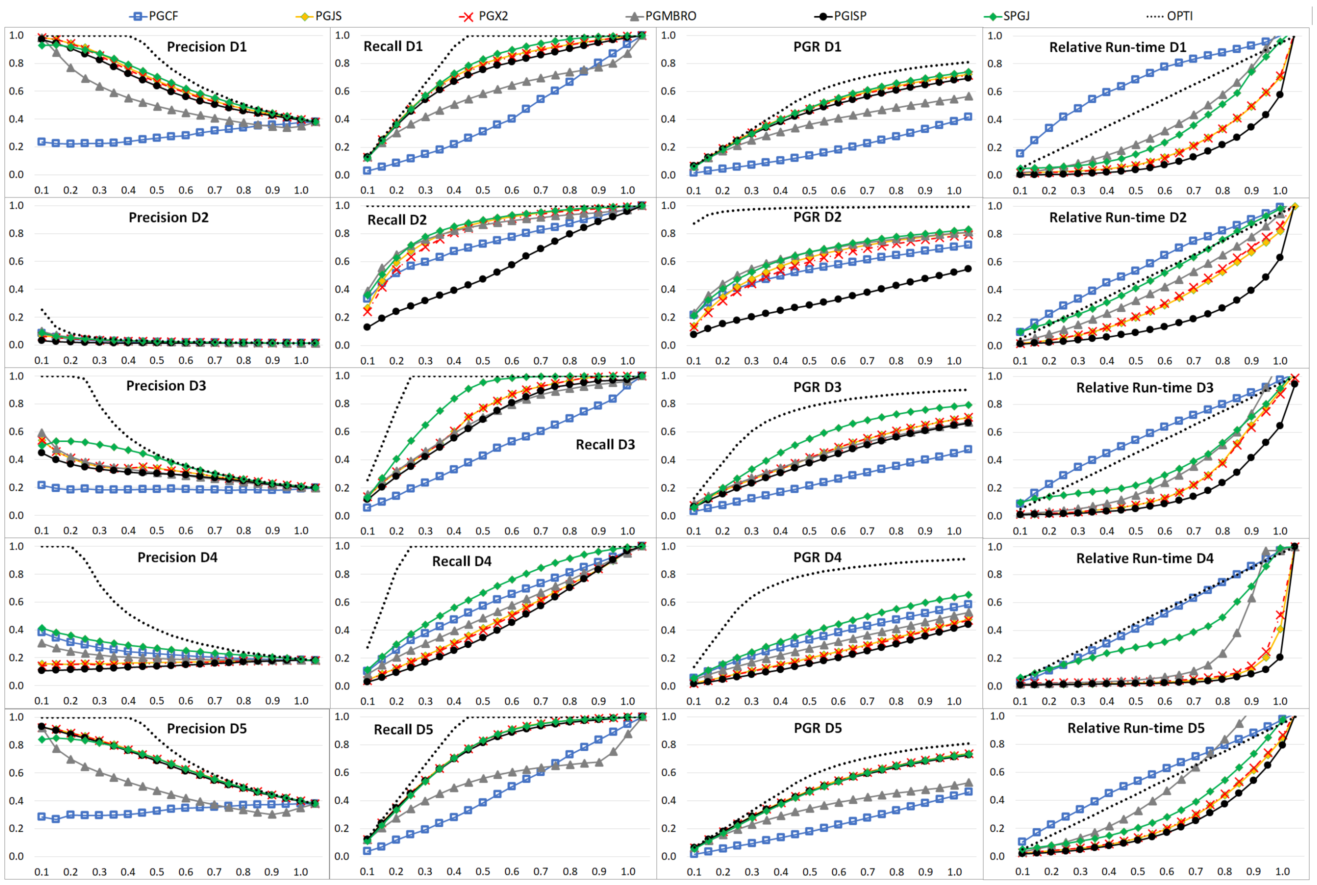

Figure 6: Performance of Supervised Progressive GIA.nt (SPGI), the learning-free Progressive GIA.nt (PGCF, PGJS, PGX2, PGMBRO and PGISP) and the optimal approach (OPTI) over all dataset pairs in Tb. 1 using as budgets all portions of candidate pairs in $[0.05, 1.00]$ with a step of 0.05.

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 w.r.t. precision and training time but excels in all other performance measures. Given that Problem 2 emphasizes PGR and prediction time, *Logistic Regression constitutes the best choice among the four probabilistic classification algorithms.*

Exp. 4: Comparison to state-of-the-art. We now compare the proposed end-to-end supervised approach (i.e., Algorithm 1) with the state-of-the-art learning-free algorithm, i.e., Progressive GIA.nt, in combination with all weighting schemes proposed in the literature (cf. Section 3.2): the co-occurrence frequency (PGCF), the Jaccard similarity (PGJS), the Pearson χ^2 test (PGX2), the MBR overlap (PGMBRO) and the inverse sum of points (PGISP). Note that the first four algorithms exclusively consider features F15, F18, F19, and F6, respectively, verifying the top-BU weighted pairs in decreasing order. Following the previous experiments, Supervised Progressive GIA.nt (SPGI) trains a Logistic Regression classifier on 500 labeled instances per class using all (16) atomic features, out of which only F15 is used by the learning-free baseline methods.

We also report the performance of the optimal progressive algorithm (OPTI), i.e., the ideal approach that verifies all topologically related pairs before the non-related ones, achieving the maximum possible value for PGR in each dataset.

Unlike [17, 18], which merely examine two budgets that are common to all datasets (5M and 10M verifications), we consider 20 different budgets per dataset, tailored to the characteristics of each dataset pair in Table 1. These are all budgets in the interval $[0.05 \cdot |C|, 1.00 \cdot |C|]$ with a step of 0.05, where $|C|$ denotes the set of geometry pairs with intersecting MBRs. Thus, the largest budget is equivalent to batch verification. The results with respect to all evaluation measures appear in Figure 6.

For effectiveness, we observe the following patterns:

• The first verifications of SPGI target both positive and negative instances in order to build the training set for its probabilistic classifier. This results in lower scores for all effectiveness measures for the smallest budget in all datasets. However, this is compensated by the high performance of the learned model in the larger budgets.

• For the two largest budgets, all methods converge to the same performance, approximating the batch algorithm, as they all process the same pairs of candidates. They only differ in the run-time for the maximum budget ($|C|$), as explained in the Appendix.

• The best weighting scheme for Progressive GIA.nt differs widely per dataset. In D_1 and D_5, the top performer is the Jaccard similarity, with the Pearson χ^2 test following in close distance. In D_2, the MBR overlap is by far the most effective scheme, in D_3, these three features exhibit very similar performance and in D_4, the co-occurrence frequency (CF) is the clear winner. This is

Table 3: The average distance per progressive algorithm from the best performance per budget across all dataset pairs. Lower distance indicates better performance, i.e., higher effectiveness. The best performance per evaluation measure and dataset is highlighted in bold. The last column shows the mean distance from the ideal solution.

	PGCF	PGJS	PGX2	PGMBRO	PGISP	SPGI	OPTI
Prec.	49.4%	2.2%	3.0%	21.9%	6.0%	**0.7%**	8.5%
Rec.	49.4%	2.2%	3.0%	21.9%	6.0%	**0.6%**	8.4%
PGR	65.3%	**1.4%**	2.3%	21.2%	5.7%	**1.4%**	10.2%

(a) $\mathbf{D_1}$

	PGCF	PGJS	PGX2	PGMBRO	PGISP	SPGI	OPTI
Prec.	14.2%	3.8%	6.4%	2.4%	36.9%	**1.4%**	15.1%
Rec.	14.5%	4.2%	6.7%	2.7%	37.1%	**1.0%**	14.7%
PGR	16.9%	9.3%	13.5%	**1.0%**	52.8%	2.0%	33.5%

(b) $\mathbf{D_2}$

	PGCF	PGJS	PGX2	PGMBRO	PGISP	SPGI	OPTI
Prec.	43.6%	13.0%	12.9%	15.7%	18.6%	**0.8%**	14.4%
Rec.	43.6%	13.0%	13.0%	15.7%	18.6%	**0.8%**	14.3%
PGR	56.3%	18.1%	19.8%	18.1%	25.8%	**1.9%**	27.5%

(c) $\mathbf{D_3}$

	PGCF	PGJS	PGX2	PGMBRO	PGISP	SPGI	OPTI
Prec.	11.2%	33.1%	34.2%	21.6%	40.1%	**0.0%**	29.5%
Rec.	11.2%	33.1%	34.2%	21.6%	40.1%	**0.0%**	29.5%
PGR	12.2%	45.1%	47.2%	26.7%	55.1%	**0.0%**	47.3%

(d) $\mathbf{D_4}$

	PGCF	PGJS	PGX2	PGMBRO	PGISP	SPGI	OPTI
Prec.	42.5%	**0.0%**	0.2%	25.6%	1.5%	1.5%	10.4%
Rec.	42.5%	**0.0%**	0.2%	25.6%	1.5%	1.5%	10.4%
PGR	57.4%	**0.3%**	0.5%	24.3%	1.6%	3.3%	12.4%

(e) $\mathbf{D_5}$

highlighted in Table 3, which reports the average distance of each algorithm from the top performance per evaluation measure and dataset across all 20 budgets. Note that these patterns verify the experimental results in [17, 18], as they depend on the topological relations that are detected in every dataset [18]. Overall, we can conclude that *the learning-free Progressive GIA.nt is hard to fine-tune, requiring expert knowledge about both the weighting schemes and the most frequent topological relations in the data at hand.*

• In contrast, SPGI's configuration exhibits quite stable effectiveness, which matches or achieves the best performance in practically all cases. In D_1, it outperforms all baseline methods for all budgets greater or equal to $0.4 \cdot |C|$ with respect to PGR and from $0.3 \cdot |C|$ with respect to precision and recall. The same applies to D_2. Hence, SPGI achieves the lowest average distance from the top for precision and recall, while for PGR, its difference with the best weighting schemes (JS in D_1 and MBRO in D_2) is statistically insignificant ($p > 0.05$). In D_3 and D_4, SPGI achieves the highest performance with respect to all evaluation measures across all budgets – except for the two smallest ones in D_3, due to the creation of the training set, as explained above. Finally, in D_5, SPGI matches the best performance only from $0.7 \cdot |C|$ on, but is consistently within reach from the three best weighting schemes with the differences being statistically insignificant. Therefore, *SPGI is a parameter-free, high-performing progressive method requiring no human/user intervention.*

Comparison to Optimal Approach. We now examine how well the considered progressive methods perform with respect to the ideal solution, i.e., OPTI. This can be deduced from the rightmost column in Figure 3, which reports the average distance of the best performing progressive method per budget and dataset from OPTI. We observe that for D_1 and D_5, the existing progressive methods lie within reach of the optimal solution. This should be attributed to the large portion (>1/3) of qualifying geometry pairs among the candidate ones, as shown in Table 1. In D_3 and D_4, the portion of qualifying pairs is much lower, <1/5, thus increasing the distance from the optimal solution. In D_2, only 1 out of 100 candidate pairs is qualifying, yet the distance from OPTI is lower than that in D_4. The reason is that most disjoint geometry pairs with intersecting MBRs share just one tile, thus receiving low scores by all progressive methods.

Time Efficiency Experiments. This section continues the comparison with the state-of-the-art in Exp. 4 in Section 5, discussing the relative run-time of the considered techniques.

The time efficiency of progressive methods is determined by the run-time of their three constituent workflow steps, i.e., Filtering, (Supervised) Scheduling and Verification. The first step is shared with the state-of-the-art batch algorithm, GIA.nt [17] (cf. Section 2), because all progressive methods apply the same dynamic approach for indexing the source dataset through an Equigrid. The last step is also shared with GIA.nt when the budget includes the entire set of candidate pairs C, i.e., all geometries with intersecting MBRs. The reason is that all geospatial interlinking algorithms: (i) apply the same algorithm for computing the intersection matrix of two geometries, and (ii) verify the same pairs of geometries, when the available budget is equal to $|C|$.

The run-times of these two steps are reported in Table 4. As expected, Filtering accounts for a tiny portion of the overall run-time – up to 0.5% (D_1). Verification *is slower by at least two orders of magnitude, thus justifying the need for progressive methods.*

The (Supervised) Scheduling time, t_s, per progressive method is reported in Table 5. Note that it is independent of the available budget, as it is determined by the number of candidate pairs (with intersecting MBRs) in each dataset. We observe that there are minor differences between the learning-free baseline methods, as they essentially perform the same operation, i.e., they assign a score to every candidate pair and sort all of them in decreasing weight. In absolute numbers, t_s is comparable to the filtering time in the smallest dataset(s), but increases substantially, even by a whole order of magnitude, for the largest ones. In any case, though, it accounts for a small portion of the total run-time even for the smallest considered budget (i.e., $0.05 \cdot |C|$). *The larger the budget is, the less significant is the cost of Scheduling.*

Compared to its learning-free counterparts, Supervised Scheduling is from 2.4 to 5.5 times slower (the former applies to D_5, when compared with χ^2, and the latter to D_3, when compared with *CF*). This means that despite its complex operations (i.e., feature generation, sampling, and labeling), the cost of Supervised Scheduling remains negligible when compared with the verification time, especially for larger budgets. Therefore, *the higher effectiveness of Supervised Progressive GIA.nt comes at an acceptable cost in run-time.*

The main variation in the total run-time of the progressive methods stems from their verification time per budget, which depends on the different complexity of the pairs verified by each algorithm. This run-time is reported in the rightmost column of Figure 6 in the form of the relative run-time with respect to batch GIA.nt, i.e., the run-time of each algorithm per budget is divided by the time required for verifying all candidate pairs in the dataset. This means that the batch algorithm corresponds to the point (1.0, 1.0) in each diagram. In this case, OPTI represents an ideal approach that evenly distributes the batch run-time among the available budgets.

Table 4: The filtering (t_f), verification (t_v) and total (t_t) run-time of batch GIA.nt per dataset.

Dataset	$\mathbf{t_i}$ (sec)	$\mathbf{t_v}$ (sec)	$\mathbf{t_t}$
D_1	13	2,389	40.0 min
D_2	13	5,159	86.2 min
D_3	48	16,397	4.6 hrs
D_4	34	55,905	15.5 hrs
D_5	38	37,490	10.4 hrs

Table 5: The run-time in seconds of the (Supervised) Scheduling step per progressive algorithm and dataset.

	PGCF	PGJS	PGX2	PGMBRO	PGISP	SPGI
D_1	23	25	21	23	22	64
D_2	40	37	35	37	36	118
D_3	65	67	72	70	66	353
D_4	437	456	489	405	428	1,596
D_5	354	368	394	391	373	953

We observe the following patterns:

• The most significant differences in the relative run-time of the progressive algorithms pertain to low budgets because, for larger ones, all methods basically verify the same geometry pairs. In other words, *progressive methods make little sense when combined with large budgets that include almost all candidate pairs.* Instead, they are more useful in applications with low budgets, i.e., limited computational and/or temporal resources.

• The fastest progressive method is PGISP in practically all cases (except for the largest budget). The reason is that it promotes simpler candidate pairs, with few boundary points, whose verification is rather efficient.

• PGJS and PGX2 follow PGISP in close distance because both promote candidate pairs that participate in a few tiles. These pairs typically involve small geometries, whose verification is also quite fast.

• The opposite is true for PGCF, which promotes candidate pairs that participate in many tiles. These typically involve large and complex geometries, with a time-consuming verification. As a result, PGCF is consistently the slowest progressive method, exhibiting much higher relative run-times than OPTI in all datasets, but D_4, where they coincide.

• Similar to PGMBRO, our supervised approach fluctuates between the two extremes of PGCF and PGJS/PGX2, being closer to the latter in most cases. The more effective is SPGI in comparison to PGJS/PGX2, the higher is their difference in run-time, which indicates that the simple and small geometries selected by the learning-free baselines are not sufficient for achieving high PGR. For example in D_4, PGJS and PGX2 are much faster than SPGI, but their effectiveness is rather poor. With the exception D_2, SPGI is also consistently faster than OPTI to a significant extent. This indicates that *its run-time scales sublinearly with the increase of the verified pairs* This also means that *the cost of Supervised Scheduling is insignificant in comparison to its benefit.*

• For the largest budget, all progressive methods are slower than batch GIA.nt to a minor extent (~5%). This additional cost is attributed to their (supervised) scheduling step, which is superfluous when processing (almost) all candidate pairs.

6 CONCLUSIONS

We proposed Supervised Scheduling as a new means of maximizing the throughput of Geospatial Interlinking (PGR) within a specific budget of verifications. We incorporated it into Supervised Progressive GIA.nt, an end-to-end approach that automatically learns a generic, effective, and fast probabilistic binary classifier. Using five pairs of large, real-world datasets, we experimentally demonstrated that combining Logistic Regression with 16 generic atomic features and 500 labeled instances per class, randomly selected across all candidate pairs with intersecting MBRs, suffices for achieving higher performance than the existing learning-free progressive methods at a small cost in run-time. In the future, we will parallelize Supervised Progressive GIA.nt on Apache Spark.

Ackowledgements. This work was supported by the first and second call for H.F.R.I. Research Projects to support faculty members and researchers and the procurement of high-cost research equipment grant (HFRI-FM17-2351 and Project No. 2757, respectively). It was also partially supported by the ESA project DA4DTE (subcontract 202320239), the Horizon 2020 projects AI4Copernicus (GA No. 101016798) and DeepCube (GA No. 101004188), and Horizon Europe project STELAR (GA No. 101070122).

REFERENCES

[1] Abdullah Fathi Ahmed, Mohamed Ahmed Sherif, and Axel-Cyrille Ngonga Ngomo. 2018. RADON2 - a buffered-intersection matrix computing approach to accelerate link discovery over geo-spatial RDF knowledge bases. In *OAEI.*

[2] Edward P. F. Chan and Jimmy N. H. Ng. 1997. A General and Efficient Implementation of Geometric Operators and Predicates. In *SSD*, Vol. 1262. 69–93.

[3] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. 1993. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In *SSD.*

[4] Eliseo Clementini, Jayant Sharma, and Max J. Egenhofer. 1994. Modelling topological spatial relations: Strategies for query processing. *Comput. Graph.* (1994).

[5] Alishiba Dsouza et al. 2021. WorldKG: A World-Scale Geographic Knowledge Graph. In *CIKM.* 4475–4484.

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

[7] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE.* 1352–1363.

[8] Eibe Frank, Mark Hall, and Ian Witten. 2016. The WEKA Workbench. Online Appendix for" Data Mining: Practical Machine Learning Tools and Techniques. https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf.

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

[10] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Intell.* 194 (2013), 28–61.

[11] Krzysztof Janowicz et al. 2022. Know, Know Where, Knowwheregraph: A Densely Connected, Cross-Domain Knowledge Graph and Geo-Enrichment Service Stack for Applications in Environmental Intelligence. *AI Mag.* 43, 1 (2022), 30–39.

[12] Nikolaos Karalis, Georgios Mandilaras, and Manolis Koubarakis. 2019. Extending the YAGO2 Knowledge Graph with Precise Geospatial Knowledge. In *ISWC.*

[13] Oje Kwon and Ki-Joune Li. 2011. Progressive spatial join for polygon data stream. In *SIGSPATIAL.* 389–392.

[14] Rushi Longadge and Snehalata Dongre. 2013. Class Imbalance Problem in Data Mining Review. *CoRR* abs/1305.1707 (2013).

[15] Nikos Mamoulis. 2011. Spatial data management. *Synthesis Lectures on Data Management* 3, 6 (2011), 1–149.

[16] Axel-Cyrille Ngonga Ngomo. 2013. ORCHID - Reduction-Ratio-Optimal Computation of Geo-spatial Distances for Link Discovery. In *ISWC.* 395–410.

[17] George Papadakis, Georgios Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. 2021. Progressive, Holistic Geospatial Interlinking. In *WWW.*

[18] George Papadakis, George Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. 2022. Static and dynamic progressive geospatial interlinking. *ACM Transactions on Spatial Algorithms and Systems (TSAS)* 8, 2 (2022), 1–41.

[19] Georgios M. Santipantakis, Apostolos Glenis, Christos Doulkeridis, Akrivi Vlachou, and George A. Vouros. 2019. stLD: towards a spatio-temporal link discovery framework. In *SBD@SIGMOD*. 4:1–4:6.
[20] Peter Sbarski and Sam Kroonenburg. 2017. *Serverless architectures on AWS: with examples using Aws Lambda*. Simon and Schuster.
[21] Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. 2017. Radon - Rapid Discovery of Topological Relations. In *AAAI*. 175–181.
[22] Panayiotis Smeros and Manolis Koubarakis. 2016. Discovering Spatial and Temporal Links among RDF Data. In *Workshop on Linked Data on the Web, LDOW*.
[23] Wee Hyong Tok, Stéphane Bressan, and Mong-Li Lee. 2006. Progressive Spatial Join. In *SSDBM*. 353–358.
[24] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. 2019. Parallel In-Memory Evaluation of Spatial Joins. In *SIGSPATIAL*. 516–519.