Raster Intervals: An Approximation Technique for Polygon Intersection Joins

Thanasis Georgiadis and Nikos Mamoulis
Department of Computer Science and Engineering
University of Ioannina, Greece
{ageorgiadis,nikos}@cs.uoi.gr

ABSTRACT

Many data science applications, most notably Geographic Information Systems, require the computation of spatial joins between large object collections. The objective is to find pairs of objects that intersect, i.e., share at least one common point. The intersection test is very expensive especially for polygonal objects. Therefore, the objects are typically approximated by their minimum bounding rectangles (MBRs) and the join is performed in two steps. In the filter step, all pairs of objects whose MBRs intersect are identified as candidates; in the refinement step, each of the candidate pairs is verified for intersection. The refinement step has been shown notoriously expensive, especially for polygon-polygon joins, constituting the bottleneck of the entire process. We propose a novel approximation technique for polygons, which (i) rasterizes them using a fine grid, (ii) models groups of nearby cells that intersect a polygon as an interval, and (iii) encodes each interval by a bitstring that captures the overlap of each cell in it with the polygon. We also propose an efficient intermediate filter, which is applied on the object approximations before the refinement step, to avoid it for numerous object pairs. Via experimentation with real data, we show that the end-to-end spatial join cost can be reduced by up to one order of magnitude with the help of our filter and by at least three times compared to using alternative intermediate filters.

ACM Reference Format:
Thanasis Georgiadis and Nikos Mamoulis. 2023. Raster Intervals: An Approximation Technique for Polygon Intersection Joins. In *Proceedings of ACM Conference (ACM SIGMOD 2023).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Spatial data are ubiquitous in scientific and commercial applications, such as Geographic Information Systems (GIS), which manage huge volumes of geographic data. For instance, Natural Earth (naturalearthdata.com) includes public-domain global geographic data at various scales in both vector and raster format. Numerous free sources of GIS data can be found at freegisdata.rtwilson.com. With the growing spatial data availability, there is an increasing need for efficient spatial data analysis tools.

An important query operation in spatial data management is the spatial intersection join [13]. Given two collections R and S of data objects, the objective is to find pairs (r, s) of objects, $r \in R$, $s \in S$, such that r and s *intersect*, i.e. they share at least one common point. This operation (a.k.a. *map overlay*) is commonly used in GIS to find the intersection of two data layers (e.g., lakes and parks). We focus on the most complex (and most expensive) case of spatial intersection joins, where the join inputs R and S contain polygons. Besides their popularity in GIS, intersection joins between polygon sets find application in many other domains. In computer graphics (e.g., solid modeling, molecular modeling) they are used for detecting the interference between geometric models [11]. Neuroscientists have used intersection joins to determine where to put synapses between neurons in synthetically constructed (polygonal or polyhedral) brain models [19]. Spatial intersection joins have also been used to identify topological relations between polygons corresponding to sources of the Semantic Web and interlink them in the Linked Open Data cloud [22].

The spatial join is an expensive operation for several reasons. First of all, the result could be as large as the Cartesian product of R and S, meaning that up to $O(|R| \times |S|)$ object pairs may have to be examined. More importantly, two polygons $r \in R$ and $s \in S$ that pass the filter step could be very complex, which means that the cost for verifying their intersection could be very high. Specifically, the time complexity of such a verification is $O(n \log n)$, where n is the sum of edges in the two polygons [7]. Third, the volume of the data in their raw format could be too large to accommodate in the memory of a commodity machine.

For the above reasons, spatial objects are typically approximated by their *minimum bounding rectangles* (MBRs) and the spatial join is processed in two steps [8, 23]: the *filter* step finds fast the pairs (r, s), where $MBR(r)$ and $MBR(s)$ intersect, with the help of spatial indexes or spatial partitioning. For each such pair, a *refinement* step accesses the geometries of r and s and verifies whether they intersect. Although the filter step eliminates from consideration the great majority of pairs, the number of pairs that pass the filter is still significant (typically, in the order of $\min(|R|, |S|)$, as we expect each object from the smallest dataset to intersect a few objects from the largest one). As we observed in our experiments (and also coined in previous work [24]), the refinement step of polygon-polygon joins may take more than 99% of the overall join time.

Given this, there are several attempts in the literature to avoid as much as possible the refinement step. Brinkhoff et al. [7] propose a number of additional object approximations (e.g., the convex hull) to be used as subsequent filters after MBR-intersection. Zimbrao and de Souza [37] proposed more effective *raster object approximations*, where each object MBR is partitioned using a grid and

ACM SIGMOD 2023, June 2023, Seatle, WA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

approximated by the percentages of cell areas that the object overlaps. This approach has several limitations. First, the raster object representations may occupy a lot of space. Second, the approximations of two candidate objects may be based on grids of different scales; their re-scaling and subsequent comparison can be quite expensive. Third, the cost of comparing two rasters in order to filter a candidate pair is linear to the number of cells in the rasters.

We propose *Raster Intervals* (RI); a raster approximation technique for polygonal objects, which does not share the drawbacks of [37] and reduces the end-to-end spatial join cost up to 10 times, when we use it as a pre-refinement, intermediate filter. Our technique uses a *global fine grid* to approximate all objects, hence, no re-scaling issues arise. In addition, RI encodes each cell by a 3-bit sequence; whether two objects overlap in a cell can be determined by bit-wise ANDing the corresponding sequences. Finally, RI models the set of cells that approximate an object o by a sorted list of *raster intervals*, determined by the Hilbert curve order of continuous cells in o's representation. For each such interval, we unify in a bitstring all 3-bit sequences of the included cells. Object pair filtering is then implemented as a merge join between the corresponding raster interval lists. For each pair of intersecting intervals, the sub-bitstrings corresponding to the common cells are ANDed to find whether there is at least one cell wherein the polygons overlap.

RI is space-economic and very efficient to use as a post-MBR filter for spatial joins. Our experiments on 7 pairs of real geographic datasets show that not only does it filter consistently more pairs compared to the state-of-the-art approaches, but also it is much more efficient compared to the rasterization approach of [37]. Another advantage of our RI approximations is that they occupy considerably less space compared to the sizes of the exact data, rendering their storage in main memory feasible.

The contributions of this paper can be summarized as follows.

- We propose a novel representation of raster object approximations as sets of intervals paired with binary codes which model the level of overlap of each object with each cell.
- We propose an efficient algorithm for joining the raster intervals of two objects that pass the filter step of the spatial join. The algorithm is an easy-to-implement merge-join paired with bitshifting and bitwise XOR and AND operations.
- We evaluate our approach on a wide range of real datasets of varying sizes and complexities and demonstrate that our approach is significantly more effective and space/time-efficient compared to alternative filters and reduces the overall join cost by up to one order of magnitude.

The rest of the paper is organized as follows. Section 2 provides the necessary background. Section 3 presents our approach, which is evaluated in Section 4. Related work on spatial intersection joins is reviewed in Section 5. Finally, we conclude in Section 6.

2 BACKGROUND

Figure 1 illustrates the spatial intersection join pipeline. An MBR-join algorithm takes as input the MBR approximations of objects to identify all pairs of objects that intersect (*filter step*) [13, 30]. Before accessing and comparing the exact object geometries for each such candidate pair, in an *intermediate step*, more detailed object approximations (than the MBR) are used to verify (fast) whether the pair is a sure result (true hit) or a sure non-result (false hit), or we cannot decide based on the approximations [7, 37]. Finally, if the pair is still a candidate, it is passed to the *refinement step* where the exact geometries are accessed and an (expensive) algorithm from computational geometry [25] is run to determine whether the pair is a result. Most previous work focused on the filter step [8, 13, 19, 30]. However, the refinement step dominates the overall cost, as discussed in the Introduction. The intermediate step using additional object approximations has been proved valuable toward reducing the overal join cost [7].

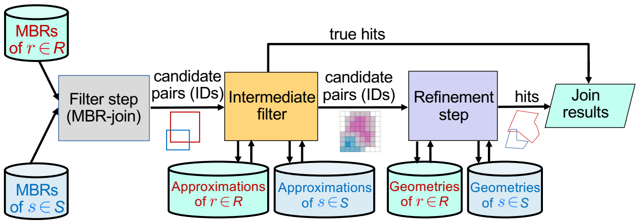

Figure 1: Spatial intersection join pipeline [7]

Zimbrao and de Souza [37] introduced an effective intermediate filter, by imposing a grid over each object's MBR. The cells of the grid comprise the *raster approximation* of the object. Each cell belongs to one of the following four types: *full* (the object completely covers the cell), *strong* (the object covers more than 50% of the cell), *weak* (the object covers at most 50% of the cell), or *empty* (the object is disjoint with the cell). Figure 2 shows an example.

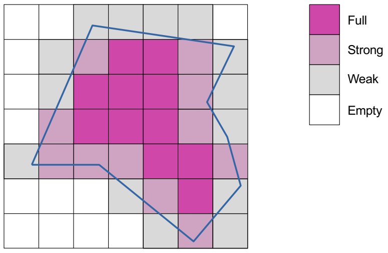

Figure 2: Four types of cells in a raster approximation [37]

To create the raster approximation (RA) of a polygon, a grid of at most K square cells is defined. The side of each cell should be $\omega 2^k$, for some $k \geq 0$, where ω is a minimum cell side (unit). In addition, the coordinates of each cell should be multiples of $\omega 2^k$.

For a pair (r, s) of candidate objects, the cells in their approximations $RA(r)$ and $RA(s)$ that overlap with their common MBR are identified and the remaining ones are ignored. If the cells of $RA(r)$ are smaller than the cells of $RA(s)$, groups of neighboring cells in $RA(r)$ are combined to infer the type of a larger cell that is perfectly aligned with a cell of $RA(s)$. Re-scaling is expensive, results in accuracy loss and reduces the effectiveness of RAs.

After re-scaling, the common cells in the two raster approximations are examined and, for each such cell, we use the cell's types in the two approximations to conclude whether the objects intersect in the cell, according to Table 1. Specifically, if at least one of the two types is *empty* the objects definitely do not intersect in the cell. If at least one of the two types is *full* and the other is not *empty* or both types are *strong*, then the objects definitely intersect in the cell. In all other cases, we cannot conclude whether the objects intersect in the cell. If we find at least one cell, where the objects intersect, the pair is directly reported as a spatial join result (true hit). If at all common cells, the objects do not intersect, then the pair is pruned (false hit). If we cannot conclude about the object pair, the refinement step should be applied.

Table 1: Do two objects intersect in a cell, based on the cell's types in the two raster approximations? [37]

	empty	**weak**	**strong**	**full**
empty	no	no	no	no
weak	no	*inconclusive*	*inconclusive*	yes
strong	no	*inconclusive*	yes	yes
full	no	yes	yes	yes

3 RASTER INTERVALS

We propose a new framework for the intermediate step of spatial joins, which builds upon, but is significantly more effective than the raster approximation technique of previous work [37]. Our approach has three important differences: (i) we use the same global (and fine-grained) grid to rasterize all objects; (ii) we use bitstring representations for the cell types of object approximations; and (iii) we represent the set of all non-empty cells of each object as a sorted list of intervals paired with binary codes. In this section, we present in detail the steps that we follow in order to generate the raster intervals approximation for each object.

3.1 Object rasterization and raster encoding

We superimpose over the entire data space (e.g., the map) a $2^N \times 2^N$ grid. For each data object o, we identify set of the cells C_o that the object intersects and use this set to approximate o. Each cell in C_o may belong to three types: *full*, *strong*, or *weak*; as opposed to [37], we do not include empty cells in C_o. In order to compute C_o for each object, and the type of each cell, we apply the algorithm of [37]. In a nutshell, the algorithm first identifies the grid columns (stripes) which overlap with o. It clips the object in each stripe, and then runs a plane-sweep algorithm along the stripe to identify the cells and the type of each cell.

Furthermore, we *encode* the three types of cells that we are using, as shown in Table 2. Note that we use a different encoding for the cell types depending on whether the object comes from join input R or S. This encoding has two important properties. First, if for two objects $r \in R$ and $s \in S$ and for a cell c, the bitwise AND of the codes of r and s in cell c is non-zero, then we are sure that r and s intersect in cell c. Indeed, this corresponds to the case where at least one type is *full* or both are *strong*. If the logical AND is 0, we cannot be sure whether r intersects s in c.

The second property of the encoding is that it allows us to swap the roles of R and S in the join, if necessary. Specifically, the code for a cell c of an object in one join input (e.g., R) can be converted to the code for c if the object belonged to the other join input (e.g., S) by XORing the code with the mask $m = 110$. For example, 011, the R-encoding of *full* cells, after bitwise XORing with m, becomes 101, i.e., the S-encoding of *full* cells. This is important for the case where the rasterization of a dataset has been *precomputed* before the join, according to the R-encoding and we want to use the dataset as the right join input S. XORing can be done on-the-fly when we apply our filter, as we explain in Section 3.3, with insignificant cost.

Table 2: 3-bit type codes for each input dataset

	input R	**input S**
full	011	101
strong	101	011
weak	100	010

3.2 Intervalization

We use the Hilbert curve [12] to order the cells in the $2^N \times 2^N$ grid. Hilbert curve is a well-known space filling curve that preserves spatial proximity. Hence, each cell is mapped to a value in $[0, 2^{2N} - 1]$. By this, the set of cells C_o that intersect an object o can be represented as a list of intervals L_o formed by consecutive cells in C_o according to the Hilbert order. Figure 3 exemplifies the *intervalization* for a polygonal object o in a $2^3 \times 2^3$ space. The cells are marked according to their Hilbert order and shaded based on their type. There are in total 36 cells in C_o, which are represented by 7 intervals. To intervalize C_o, we sort the cells there in Hilbert order and scan the sorted array, merging cells of consecutive cells into the current interval. The cost for this is $O(|C_o| \log |C_o|)$.

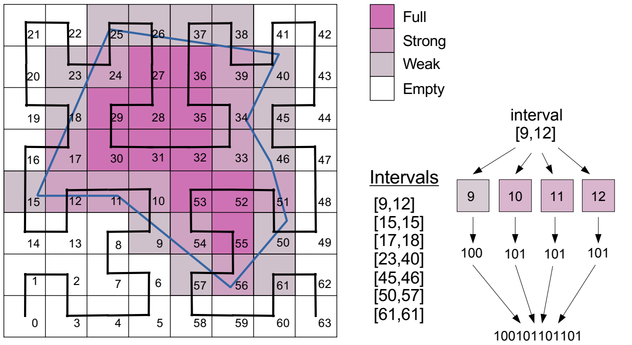

Figure 3: The Hilbert curve cell enumeration and interval generation for a polygon in a 8×8 space.

For each interval in L_o, during the interval construction, we *concatenate* the bitwise representations of the cells in their Hilbert order, to form a *single* code for the entire interval. This allows us to replace the set C_o of cells that intersect an object o by L_o. For example, assume that the polygon of Figure 3 belongs to the left join input R. We replace cells 9, 10, 11, and 12 in C_o with codes 100, 101, 101, and 101, respectively, by interval $[9, 12]$ with binary code 100101101101, as shown in the figure. This helps us to greatly reduce the space requirements for the rasterized objects. In addition, as we will show next, we save many computations while verifying a pair of objects, because we can apply the bitwise AND for multiple cells simultaneously. The resulting *raster intervals* (RI) approximation of each object is a sequence of $\langle st, end, code\rangle$ triples (ordered by st), where $[st, end]$ is an interval in the Hilbert curve space and *code* is bitstring encoding the cell types in the interval.

Practical considerations A larger value for N results in a finer-grained grid and thus more accurate approximations. Moreover, a polygon rasterized with higher granularity has an increased probability to have completely covered cells (i.e., type *full*), which increases the chances of the intermediate spatial join filter to identify a true hit. At the same time, a large N requires more space for storing the endpoints of the intervals in L_o. We choose $N = 16$, which results in a grid with a fine granularity; in addition, the Hilbert order of cells (i.e., the interval endpoints) can be stored as 32-bit unsigned integers. As each cell in an interval contributes three bits to the interval's concatenated binary code, for a $[st, end]$ interval, we need $\lceil (end - st + 1) * 3/8 \rceil$ bytes to encode its cells. We may opt to compress binary codes consisting of many bytes and the RI approximation of an object, overall.

3.3 Intermediate filter

For a join candidate pair (r, s), $r \in R, s \in S$ which is produced by the MBR-join algorithm, our objective is to use the raster intervals approximations $RI(r)$ and $RI(s)$ of r and s to verify fast whether r and s definitely intersect, (ii) r and s definitely do not intersect, or (iii) we cannot conclude about the intersection of r and s, based on their RIs. This is done via our RI-join procedure (Algorithm 1).

Algorithm 1 RI-join algorithm

```
Require: $RI(r)$ as $X$, $RI(s)$ as $Y$
ovl ← False;                                   ▷ no overlapping interval pair found yet
$i \leftarrow 0$; $j \leftarrow 0$
while $i < |X|$ and $j < |Y|$ do
    if $X_i$ overlaps with $Y_j$ then
        if AlignedAND($X_i.code$, $Y_j.code$) then
            return true hit                    ▷ bitwise AND is non-zero
        end if
        ovl ← True;                            ▷ found an overlapping interval pair
    end if
    if $X_i.end \leq Y_j.end$ then $i \leftarrow i + 1$ else $j \leftarrow j + 1$
end while
if ovl then                                    ▷ at least one overlapping interval pair
    return indecisive
else
    return false hit                           ▷ no common cells in $X$ and $Y$
end if
```

RI-join merge-joins the sorted interval lists $RI(r)$ and $RI(s)$, denoted by X and Y in the pseudocode, respectively, and identifies pairs (X_i, Y_j) of intervals that overlap; i.e., X_i and Y_j include at least one common cell. For each such pair, there is a possibility to find out that (r, s) is a true hit (i.e., a spatial join result) and avoid sending the pair to the refinement step. Specifically, if in at least one of the common cells of X_i and Y_j the logical AND of the cell codes is non-zero, we have a sure true hit and we do not need to continue the RI-join. Having the codes of the cells in X_i and Y_j concatenated in two single bitstrings $X_i.code$ and $Y_j.code$ allows us to perform this check (abstracted by Function AlignedAND) efficiently. We first select from each bitstring the fragment that includes the codes of all cells in the common subinterval $[\max\{X_i.st, Y_j.st\}, \min\{X_i.end, Y_j.end\}]$. Then, we bitwise AND the fragments. If the fragments have been encoded by the same encoding (i.e., both have R or S encoding as shown in Table 2), ANDing is preceded by XORing one of the two codes. If there is at least one pair (X_i, Y_j) of overlapping intervals (variable *ovl* of Algorithm 1 is True at the end of the while-loop), but the object pair is not found to be a true hit, then the object pair is *indecisive*, meaning that we will have to apply the refinement step for it. On the other hand, if there are no overlapping intervals in the two RIs (*ovl* remains False), there are no common cells in the raster representations of the objects, and we can conclude that the two objects definitely do not intersect (false hit). As an example, Figure 4 shows two rasterized polygons and the pairs (X_i, Y_j) of intervals from the two raster intervals that overlap.

In general, the codes (bitstings) of two intersecting intervals may occupy multiple bytes and the common subinterval may be of arbitrary length. Before bit-shifting, Function AlignedAND truncates all unmatched bytes from the two bitstrings. In addition, bit-shifting is done at the bytes of one interval only (the one that starts earlier), making sure to carry over the required bits from the next byte to avoid any loss of information. This continuous shifting and matching (binary AND between aligned bitstrings) is performed byte-by-byte, hence, once two ANDed bytes give a non-zero, we immediately report the true hit. XORing, (if both join inputs have the same encoding), is done on-demand on the shifted byte, after any potential bit carryover. A byte-wide XOR mask m_{byte} is used, created by concatenating our mask $m = 110$ a few times to fill a byte; m_{byte} is shifted, if necessary. The whole process can easily be parallelized (shifting and bitwise operations are independent for each byte).

For each pair of intervals, the last bytes to be matched is a special case and has to be treated cautiously, since the remaining bits that need checking may be less than 8 and the rest of the bits in that byte should not be included in the bitwise operations. In other words, the XOR and AND operations applied on the last bytes should consider bits only in the positions relevant to the compared intervals, otherwise we may mistake a false positive as a true hit. Hence, we apply one last bit mask with 1s at the positions of the bits that need to partake in the operation, setting the rest to zero.

Figure 5 shows how the codes for first pair (X_0, Y_1) of intersecting intervals from the example of Figure 4 are matched, where $X_0 = \langle [9, 12], 100101101101\rangle$ and $Y_1 = \langle [11, 14], 100100101100\rangle$ (i.e., assume that both datasets are R-coded). Each code occupies 2 bytes. Since the interval of Y_1 starts 2 cells after the interval of X_0, the code of X_0 is shifted by $2 \times 3 = 6$ bits in the first step. This aligns the common cells (11 and 12) in the two codes. The common fragment (6 bits) occupies 1 byte, so there will be one byte-by-byte match. As both intervals are R-coded, we first XOR the X_0-byte with the (shifted) byte-wise XOR mask m_{byte}. Before ANDing the two bytes, we AND the shifted byte with a mask that clears the bits that are outside the common fragment of the intervals, as we are at the last byte. Finally, the bytes are ANDed with a 0 result, so the intersection of the two objects remains indecisive with respect to (X_0, Y_1). As a result, Algorithm 1 continues to find next pair of overlapping intervals (X_5, Y_2) and performs the corresponding code matching.

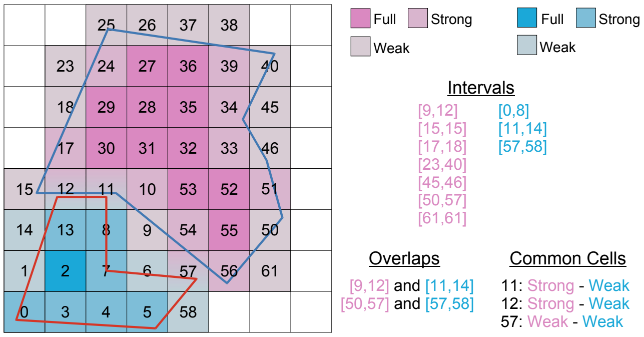

Figure 4: Two rasterized polygons, the overlaps between their raster intervals, and their common cells

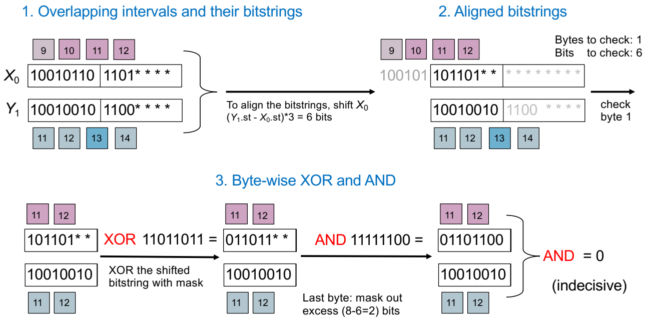

Figure 5: Intervals $[9, 12]$ and $[11, 14]$ of our two example polygons overlap but are not aligned. Byte truncation and bit shifting (if necessary) align their bitstrings before performing the bitwise operation(s)

RI-join requires a single scan of interval lists X and Y, since no two intervals in the same list (i.e., in the same polygon) overlap. Assuming that bitstrings are relatively short so that their matching (a call to Function AlignedAND) takes constant time, the time complexity of Algorithm 1 is $O(|X| + |Y|)$ since the number of overlapping interval pairs is at most $|X| + |Y|$.

3.4 "Within" spatial joins

Although we focus on polygon-polygon intersection joins, RI can also be used as an intermediate filter for *within joins*. The objective of a spatial within join is to find pairs (r, s) of objects, $r \in R$, $s \in S$, such that r is *within* s, i.e. the space occupied by r is a subset of the space occupied by s. For each pair (r, s) of polygons that passes the filter step of the within join (i.e., the MBR of r is within the MBR of s), we can apply Algorithm 1 with the following changes in order to identify whether (r, s) is a true negative (false hit), a true positive (i.e., true hit), or an indecisive pair w.r.t. the within predicate: As soon as we find an interval $X_i \in RI(r)$ which is not a subset of any interval $Y_j \in RI(r)$, we can terminate with the assertion that r is not within s, since there is at least one non-empty cell of r which is empty in s. In addition, for an identified pair of (X_i, Y_j), such that $X_i \subseteq Y_j$, if there is a cell in X_i that is (i) *full* in X_i but not full in Y_j or (ii) *strong* in X_i and *weak* in Y_J, then (r, s) should a true negative and the algorithm terminates. For (x, y) to be characterized as a true hit without refinement, for all identified (X_i, Y_j) such that $X_i \subseteq Y_j$, all cells in the subinterval X_i where X_i and Y_j overlap should be *full* in Y_j; if at least one such cell is not full, then we cannot guarantee a true hit and the pair (x, y) must be passed to the refinement step, unless it is found to be a true negative.

4 EXPERIMENTAL ANALYSIS

We experimentally assess the effectiveness of our raster interval (RI) approximation approach, comparing it with previously proposed intermediate filters for spatial intersection joins. The competitors include the 5-corner approximations comparison followed by a comparison of convex hulls (5C-CH) (i.e., the approach of [7]) and the raster approximation (RA) approach of [37]. The RA of each polygon is represented in memory by the minimum coordinates of its MBR, the number of cells (partitions) in each dimension and the scale (k) of the rasterization. We also keep in an array the types of all cells in the RA. We do not use our bit encoding and we do not perform intervalization in RA. We also included a baseline approach that conducts the refinement step directly for each pair of MBRs that passes the filter step without using any intermediate filter. The filter step of the join was implemented using the algorithm of Tsitsigkos et al. [30]. The refinement step was implemented using the approach of [2], where point-in-polygon tests and line segment intersections are avoided as much as possible. All methods were implemented in C++ and compiled with the -O3 flag. The experiments were run on a machine with a 3.6GHz Intel i9-10850k and 32GB RAM.

4.1 Datasets

We used datasets from SpatialHadoop's [27] collection. The first two datasets (T1 and T2) contain landmark and water areas, respectively, from the United States (conterminous states only). We also used two OpenStreetMap (OSM) datasets (O5 and O6), containing lakes and parks, respectively, from all over the world. We grouped the polygons of each of the two OSM datasets by continent and created 6 pairs of OSM datasets: North America (O5NA and O6NA), South America (O5SA and O6SA), Oceania (O5OC and O6OC), Europe (O5EU and O6EU), Asia (O5AS and O6AS), and Africa (O5AF and O6AF). Spatial joins were conducted only between datasets that refer to the same geographic regions (i.e., T1 ⋈ T2, O5NA ⋈ O6NA, O5SA ⋈ O6SA, etc.). Table 3 summarizes the statistics of all 14 datasets used in the joins. Pairs of joined datasets vary in size, relative size between inputs R and S, and average MBR area ratio of objects to the MBR of all data in both inputs (third row of Table 3).

4.2 The effect of N in RI

Recall that our RI approach superimposes a $2^N \times 2^N$ grid over the data space and approximates each object o with the set C_o of cells that overlap with o. C_o is then modeled by a set of intervals and a bitstring for each interval, which encodes the types of the cells that it contains. As discussed in Section 3.2, we set the value of N to 16, in order to have a fine granularity and be able to store the interval endpoints in 4-byte unsigned integers. In the first experiment, we confirm the appropriateness of this choice, by evaluating the effectiveness of RI in spatial joins for various values of N.

Table 3: Statistics of the datasets and space requirements of the data and the approximations

	T1	T2	O5NA	O6NA	O5SA	O6SA	O5OC	O6OC	O5EU	O6EU	O5AS	O6AS	O5AF	O6AF
# of Polygons	125K	2.25M	4.02M	1M	124K	228K	195K	315K	1.97M	7.18M	448K	623K	72K	192K
Avg # of vertices	32.17	31.92	37.73	47.68	48.90	41.81	52.15	40.85	35.14	32.24	46.14	42.75	59.99	36.53
Avg obj MBR/ world MBR	2.5E-07	2.8E-08	3.4E-08	1.3E-07	3.9E-07	6.7E-07	5.4E-08	6.1E-08	6.2E-08	5.9E-08	1.6E-07	1.3E-07	4.1E-07	2.5E-07
Geometries size (MB)	65.67	1.17K	2.5K	771.50	163.90	208.30	97.80	154.50	1.1K	3.8K	334.30	431.30	70.20	113.60
MBR size (MB)	4.52	81.08	144.84	36.03	4.46	8.22	7.01	11.34	70.90	258.48	16.13	22.44	2.61	6.90
RI size (MB)	33.20	128.29	231.73	220.64	32.01	143.67	16.32	32.97	173.82	920.28	63.01	122.71	18.51	55.43
RA size (MB)	1.2K	19.9K	6.2K*	1.5K*	1.1K	2.1K	1.6K	2.9K	3.1K*	11.4K*	3.7K	5.6K	621.80	1.7K
5C-CH size (MB)	20.70	705.40	1.17K	257.70	50.40	80.50	30.40	52.90	515.40	1.7K	117.80	159.40	18.50	46.60

Table 4: Computational costs (in sec) of intermediate filter approximations

	T1	T2	O5NA	O6NA	O5SA	O6SA	O5OC	O6OC	O5EU	O6EU	O5AS	O6AS	O5AF	O6AF
RI preprocessing (s)	42.109	93.161	243.30	230.630	43.472	201.617	18.920	37.420	164.22	761.78	73.96	168.83	29.56	70.27
RA preprocessing (s)	38.211	623.256	269.25*	67.455*	38.552	68.257	55.840	95.966	116.66*	408.76*	127.21	195.99	21.71	55.55
5C-CH preprocessing (s)	1.228	30.780	52.69	11.799	1.619	2.226	2.652	3.507	22.08	65.12	5.54	7.25	1.04	1.89

Table 5 analyzes the performance of RI for different values of N in spatial join T1 ⋈ T2. The number of candidate join pairs (whose MBRs intersect) is 94813 and the number of join results is 35365 (i.e., about 38% of the pairs that pass the filter step indeed intersect). The first three columns of the table show the percentage of candidate pairs identified by RI as true hits, false hits, or inconclusive (i.e., should be sent to the refinement step). The last four columns show the cost of the filter step of the spatial join (MBR-join), the total cost of applying our RI-filter to all candidate pairs, the total cost of the refinement step, and the overall join cost. The MBR-join cost is N-invariant, as this operation is independent of the subsequent steps (intermediate filter, refinement). Observe that the number of inconclusive pairs shrinks as N increases; the refinement cost decreases proportionally. On the other hand, the cost of RI-filter increases with N as the intervals become more and longer. Eventually, for the largest value of N, the overall join cost converges to about 1 second.

Table 5: Effect of N on the performance of RI in T1⋈T2

	True hits	False hits	Indecisive	MBR-join (s)	RI-filter (s)	Refinement (s)	Total time(s)
$N = 10$	7.93%	23.71%	68.36%	0.044	0.040	3.649	3.733
$N = 13$	17.54%	45.07%	37.40%	0.044	0.046	1.526	1.616
$N = 14$	22.46%	49.65%	27.89%	0.044	0.147	1.142	1.333
$N = 15$	27.40%	53.23%	19.37%	0.044	0.263	0.784	1.091
$N = 16$	30.87%	55.87%	13.26%	0.044	0.501	0.531	1.076

Table 6: Effect of N on the cost and space of RI for T1 and T2

T1	RI constr. cost (s)	Intervals/Polygon	Cells/Interval	Size (MB)
$N = 10$	2.75	1.11	1.24	2.30
$N = 13$	3.08	2.55	4.25	3.50
$N = 14$	5.33	2.90	13.56	5.70
$N = 15$	13.09	4.92	29.36	12.10
$N = 16$	41.42	9.04	61.20	33.20
T2	RI constr. cost (s)	Intervals/Polygon	Cells/Interval	Size (MB)
$N = 10$	44.21	1.04	1.05	39.19
$N = 13$	35.98	1.39	1.56	46.19
$N = 14$	39.56	1.74	2.39	55.19
$N = 15$	51.12	2.52	4.17	76.09
$N = 16$	93.88	4.13	7.88	128.29

In Table 6, we show the total time required to compute the RI object approximations of all objects in T1 and T2 and the corresponding storage requirements for them, as a function of N. For small values of N, where RI is not very effective, the computation cost and the space requirements are low because, for each object, only a small number of intervals, each approximating a small number of cells are constructed. On the other hand, for large values of N, where our RI-filter is effective, the approximations are very fine and require more time for computation and more space. We performed the same analysis for all other pairs of joined datasets (results are not shown, due to space constraints) and drew the same conclusions. Overall, due to the high effectiveness of RI for $N = 16$, which brings the best possible performance to the overall spatial join, we choose this value of N in the rest of the experiments. Although we use a fixed grid for all objects (independently of their sizes), the intervalization and compression of the raster representations does not incur an unbearable space overhead and at the same time we achieve a very good filtering performance even for small objects, while avoiding re-scaling at runtime (as opposed to [37]).

4.3 Data preprocessing

The approximations used by intermediate filters (i.e., RI, RA, 5C-CH) need not be computed on-the-fly, but can be generated in a data preprocessing phase. Like object MBRs, these approximations can be useful in other operations besides joins (e.g., range queries [8]), so it is reasonable to pre-compute them and store them in dedicated data structures (or together with the object geometries). This way, we can generate the approximations once per dataset and then use them indefinitely.

Table 4 shows the costs for precomputing the three intermediate filter approximations for all real datasets. The corresponding space requirements are shown in Table 3 together with the space requirements of all geometries and their MBRs. As discussed, for our RI approach, we set $N = 16$. For the RA approach, we set $K = 750$, as suggested in [37], in all cases, except for the OSM North America and Europe datasets, where we set $K = 100$ (for $K = 750$ the RAs occupy too much space and they cannot be loaded in memory and used for the join). Experimental results for RA with $K = 100$ are marked with an asterisk (*). As expected, the 5C-CH approximations have the lowest computation cost, because they do not involve a rasterization process. RI has similar precomputation cost compared to RA. The difference between RI and 5C-CH pays off as we will see later. When comparing the space requirements of all methods in Table 3, we observe that RI approximations are space-economic, being of similar scale as 5C-CH approximations, sometimes much cheaper, especially in cases of complex polygons with small areas (e.g., T2 and O5NA), where the space requirements of RI are close to the (minimal) space requirements of the object MBRs. On the other hand, RA is very space-consuming, typically occupying 1-2 orders of magnitude more space than RI and 5C-CH.

For the RI approximations, Table 7 shows the average number of intervals per polygon and the average number of cells per interval for each OSM dataset. Observe that the number of intervals per object is quite small, but the number of cells per interval is much larger, signifying that RI can achieve a quite good compression of the object approximations and explaining the low space requirements of RI compared to RA, as shown in Table 3.

Table 7: Statistics of RI approximations for OSM data

	Intervals/Polygon	Cells/Interval
O5NA	4.37	9.53
O6NA	10.28	41.40
O5SA	14.69	31.84
O6SA	12.48	159.81
O5OC	6.23	12.33
O6OC	5.19	37.07
O5EU	6.30	15.60
O5EU	8.81	18.16
O5AS	9.38	18.74
O6AS	7.15	66.99
O5AF	13.81	31.34
O6AF	8.46	101.32

To justify the use of Hilbert curves for intervalization, we measured, for $N = 16$, the average number of intervals per intervalized polygon, the average number of cells per interval, and the space requirements if we replace Hilbert order by the popular Z-order [18]. Indicatively, Table 9 shows these statistics for some of our datasets (see Tables 6, 7, and 3, for the corresponding numbers for RI). Intervalization based on Z-order curves results in more and shorter intervals and, as a result in larger space requirements for RI. We also observed that replacing Hibert-order by Z-order increases the runtime cost of using the RI filter by up to 50%, as more intervals need to be accessed and joined.

4.4 Performance in end-to-end spatial joins

We evaluate RI, RA, and 5C-CH on all join pairs, in terms of filtering effectiveness and spatial join cost that we can achieve using them. Table 8 compares the performances of all three methods in an end-to-end spatial join pipeline on the 7 join pairs of real datasets. As a point of reference, we also included None, the baseline method that does not apply an intermediate filter, but directly passes the candidate pairs to the refinement step. The first three columns of the table show the percentage of MBR-join results (i.e., candidate pairs) which are identified by each intermediate filter as true hits, as false hits, or as indecisive (these are passed to the refinement step). The next two columns show the average number of vertices in the candidates from each input dataset (R or S) which are indecisive. The last four columns show the total cost of the filter step (MBR-join [30]), intermediate filter step (RI, RA, or 5C-CH), and refinement step [2] in the end-to-end join process. For RI, we preprocessed all datasets using $N = 16$ and R-encoding (hence, the RI-join involves XORing besides ANDing). For RA, we used $K = 750$ for all datasets, except for OSM North America and Europe datasets, as explained in Section 4.3.

Filter effectiveness Observe that RI has consistently the best filtering performance among the three intermediate filters, since it results in the smallest percentage of indecisive pairs. The only exception is in O5OC ⋈ O6OC, where RI comes second to RA, by just 1%. In all joins, the true hit and false hit ratio of RI is higher compared to the corresponding ratios of the other filters (except for the true hits in O5OC ⋈ O6OC). This shows that the global grid used by RI is more effective in finding true and false hits compared to the local grid used by RA for each object. 5C-CH can only identify false hits; still, it finds fewer false hits than RI (in all cases) and RA (in most cases).

Intermediate filter cost 5C-CH operates on simpler approximations to rasters and applies fast computational geometry techniques for convex polygons; hence, it is faster than RI and RA in all cases. RI is consistently better than RA because it uses our fast RI-join algorithm and bitwise operations to conduct its checks. In addition, RI avoids the re-scaling cost that RA bears for the spatial alignment of the raster approximations of the two objects. Especially when the candidate pair includes objects of very different sizes, the re-scaling cost is high and at the same time the approximation quality of RA decreases a lot. The cost of re-scaling in RA is between 25.2% (in T1 ⋈ T2) to 59.4% (in O5OC ⋈ O6OC) of the total RA filter cost.

Refinement cost The refinement cost of the indecisive pairs that pass the RI-filter is much lower compared to the corresponding refinement costs for the pairs that pass the other filters. The relative cost difference is much higher compared to the corresponding difference in the percentages of indecisive pairs. For example, in T1 ⋈ T2, the refinement cost of RI is about 7 times lower compared to that of RA, although the difference in the number of indecisive pairs is less than 3. In order to understand the reason behind this difference, we measured the average number of vertices in the polygon pairs that pass the intermediate filters from both inputs (shown in the 4th and 5th columns of Table 8). A first observation is that the number of vertices of the polygons that pass the MBR-join is very large compared to the average number of vertices of the polygons in the corresponding datasets (see Table 3). By looking into the results, we found out that the join pairs mainly include large polygons with multiple edges, whereas small polygons rarely participate in join results. The second and most important observation is that the polygons that survive our RI-filter have much lower complexity compared to the ones that survive the other filters. This happens because our global fine grid, which is the same for all objects, is more appropriate for finding intersections between large polygons compared to the object-size parametric grid of RA. These measurements unveil an additional and not that obvious advantage of our approach.

Overall cost The total cost of RI-based end-to-end spatial join is consistently lower compared to end-to-end joins that use alternative intermediate filters. The relative speedup compared to the runner up (5C-CH) is 2.73x-5.77x. The improvement over the baseline approach (None) that does not apply any intermediate filter is between 6 and 10.7 times. The fact that the space requirements of RI approximations are much smaller than the space required for the object geometries (see Table 3) makes them a very attractive approximation approach.

Spatial within joins Table 10 shows the performance of all intermediate filters for spatial *within* joins. Section 3.4 explains how the

Table 8: Performance of intermediate filters in spatial intersection joins

	True hits	**False hits**	**Indecisive**	**Avg # vertices** R	**Avg # vertices** S	**MBR-join (s)**	**Int. Filter (s)**	**Refinement (s)**	**Total time (s)**
	T1 $\bowtie$ T2 (Tiger landmark and water areas)								
RI	**30.87%**	**55.87%**	**13.26%**	**630.70**	300.34	0.04	0.50	**0.53**	**1.07**
RA (K = 750)	22.72%	46.13%	31.15%	1568.91	367.30	0.04	1.70	3.58	5.32
5C-CH	0.00%	39.86%	60.14%	1450.08	**279.29**	0.04	**0.07**	4.34	4.45
None	0.00%	0.00%	100.00%	1337.64	375.68	0.04	-	7.15	7.19
	O5NA $\bowtie$ O6NA (OSM lakes and parks in North America)								
RI	**47.15%**	**34.30%**	**18.55%**	**533.64**	**358.49**	0.69	9.57	**20.14**	**30.40**
RA (K = 100)	24.51%	20.27%	55.22%	1993.36	831.70	0.69	11.29	254.72	266.70
5C-CH	0.00%	27.28%	72.72%	1180.19	814.88	0.69	**2.16**	172.52	175.38
None	0.00%	0.00%	100.00%	395.01	789.53	0.69	-	325.34	326.03
	O5SA $\bowtie$ O6SA (OSM lakes and parks in South America)								
RI	**13.75%**	**62.29%**	**23.97%**	**827.93**	**160.34**	0.05	0.58	**1.26**	**1.88**
RA (K = 750)	11.89%	51.56%	36.55%	3691.81	384.24	0.05	2.37	9.63	12.04
5C-CH	0.00%	46.97%	53.03%	3112.23	611.08	0.05	**0.09**	9.77	9.91
None	0.00%	0.00%	100.00%	2534.90	480.83	0.05	-	15.27	15.32
	O5OC $\bowtie$ O6OC (OSM lakes and parks in Oceania)								
RI	14.87%	**51.01%**	34.12%	**515.50**	**602.59**	0.06	0.34	**2.55**	**2.95**
RA (K = 750)	**17.71%**	49.21%	**33.08%**	1245.15	2438.91	0.06	2.80	10.11	12.97
5C-CH	0.00%	42.18%	57.82%	1028.98	2187.97	0.06	**0.14**	11.60	11.80
None	0.00%	0.00%	100.00%	1335.92	1686.58	0.06	-	19.09	19.15
	O5EU $\bowtie$ O6EU (OSM lakes and parks in Europe)								
RI	**12.64%**	**54.99%**	**32.37%**	**174.48**	**107.20**	1.50	14.71	**27.08**	**43.29**
RA (K = 100)	6.81%	36.85%	56.33%	712.82	491.03	1.50	26.97	216.62	245.09
5C-CH	0.00%	50.13%	49.87%	688.37	552.29	1.50	**3.89**	146.02	151.42
None	0.00%	0.00%	100.00%	838.13	440.23	1.50	-	303.97	305.48
	O5AS $\bowtie$ O6AS (OSM lakes and parks in Asia)								
RI	**8.73%**	**62.95%**	**28.32%**	**381.42**	**2013.75**	0.12	1.25	**10.37**	**11.74**
RA (K = 750)	8.59%	55.18%	36.23%	2009.54	5621.89	0.12	7.92	58.07	66.12
5C-CH	0.00%	53.84%	46.16%	1293.49	4354.51	0.12	**0.30**	37.52	37.95
None	0.00%	0.00%	100.00%	1618.71	3620.09	0.12	-	76.61	76.73
	O5AF $\bowtie$ O6AF (OSM lakes and parks in Africa)								
RI	**16.13%**	**58.41%**	**25.46%**	**439.73**	**214.42**	0.02	0.19	**0.44**	**0.65**
RA (K = 750)	14.48%	49.06%	36.47%	1298.53	300.31	0.02	1.07	1.76	2.85
5C-CH	0.00%	44.90%	55.10%	941.46	329.29	0.02	**0.05**	1.70	1.77
None	0.00%	0.00%	100.00%	1310.28	273.77	0.02	-	3.86	3.88

Table 9: Statistics using Z-order curve

	Intervals/Polygon	Cells/Interval	Size (MB)
T1	16.60	36.06	43.2
T2	6.18	5.23	172.6
O5NA	6.06	6.67	297.5
O6NA	15.91	26.52	268.8
O5AF	19.74	21.69	22.4
O6AF	12.19	68.59	62.1

RI-join is adapted for within joins. Similar changes are applied to RA, where the types of common cells of $RA(x)$ and $RA(y)$ are used to identify true positives and true negatives. Regarding 5C-CH, a pair is a false hit, if the 5C approximation of x does not intersect the 5C of y or if the CH of x is not within the CH or y. In all pairs of datasets, we used water areas as the left join input, since land areas are rarely contained in water areas. Due to space limitation, we only show the results of O5NA $\bowtie_{\subseteq}$ O6NA from the OSM datasets; the results for other pairs are similar. As the table shows, RI is significantly better compared to RA and 5C-CH also for within joins. Note that the refinement step for x within y is more expensive than the refinement step for x intersects y, as the former performs a point-in-polygon test for every vertex of x; all x-vertices should be included in y if x is within y. Hence, although the number of candidate pairs for within are fewer compared to the corresponding candidates for intersection, the join cost may be increased in some cases (e.g., T2 $\bowtie_{\subseteq}$ T1 is more expensive than T2 $\bowtie$ T1).

Table 10: Filter performance in spatial within joins

	True hits	**False hits**	**Indecisive**	**Total time (s)**
	T2 $\bowtie_{\subseteq}$ T1 (Tiger water and landmark areas)			
RI	**27.72%**	**59.74%**	**12.54%**	**1.67**
RA (K = 750)	20.88%	46.67%	32.45%	7.09
5C-CH	0.00%	42.10%	57.90%	9.62
None	0.00%	0.00%	100.00%	11.43
	O5NA $\bowtie_{\subseteq}$ O6NA (OSM lakes and parks in North America)			
RI	**50.52%**	**30.23%**	**19.24%**	**21.19**
RA (K = 100)	25.00%	13.25%	61.75%	91.23
5C-CH	0.00%	20.48%	79.52%	124.41
None	0.00%	0.00%	100.00%	134.07

5 RELATED WORK

Filter step of spatial joins The majority of previous work on spatial joins focuses on the filter step [13]. Divide-and-conquer join evaluation techniques partition the data space explicitly [19, 23, 30] or implicitly [8, 17] with the help of pre-existing spatial indexes [6], and assign the object MBRs to the partitions. For each pair of (explicit or index) partitions that spatially overlap, the intersecting MBR-pairs in the partitions are found using plane-sweep [4].

Intermediate filters To further reduce the candidate pairs that reach the refinement step, conservative and/or progressive object approximations can be used for identifying false hits and/or true hits, respectively. Brinkhoff et al. [7] suggested the use of the convex hull and the minimum bounding 5-corner convex polygon (5C) as conservative approximations and the maximum enclosing rectangle (MER) as a progressive approximation. MER is hard to compute and of questionable effectiveness [37], hence, we did not include it in our comparison. In follow-up work [37], the object geometries are rasterized and modeled as grids, where each cell is colored based on its percentage of its coverage by the object. By re-scaling and aligning the grids of two candidate join objects, we can infer, in most cases, whether the objects are a join pair or a false hit. Indecisive pairs are forwarded to the refinement step. Hierarchical (quad-tree based) raster approximations based on a hierarchical grid have been used in the past [10] for window and distance queries. In addition, Teng et al. [29] propose a hybrid vector-raster polygonal approximation, targeting point-in-polygon queries and point-to-polygon distance queries. This approach has significant storage overhead as it keeps both the raster representations and the intersections of each polygon with its raster cells. Neither [10] or [29] use the *full-strong-weak* cell classification [37], which is very effective for polygon-polygon tests, or our intervalization combined with bitstring compression that replaces expensive geometric intersection tests by cheap bitwise operations.

Speeding up the refinement step Identifying whether two polygons overlap requires point-in-polygon tests and finding an intersection in the union of line segments that form both polygons [7]. A point-in-polygon test bears a $O(n)$ cost, while the second problem can be solved in $O(n \log n)$ time [25], where n is the total number of edges in both polygons. Given a pair of candidate objects, Aghajarian et al. [2] prune all line segments from the object geometries that do not intersect their common MBR (CMBR) (i.e., the intersection area of their MBRs), before applying the refinement step. This reduces the complexity of refinement, as a smaller number of segments need to be checked for intersection. In addition, if one object MBR is contained in the other, then the point-in-polygon test is applied before the segment intersection test. Polysketch [16] decomposes each object to a set of tiles, i.e., small MBRs which include consecutive line segments of the object's geometry. Given two candidate objects, the refinement step is then applied only for the tile-pairs that overlap. A similar idea (trapezoidal decomposition) was suggested by Brinkhoff et al. [7] and alternative polygon decomposition approaches where suggested in [5]. PSCMBR [15] combines Polysketch with the CMBR approach. Specifically, for the two candidate objects, the overlapping pairs of Polysketch tiles are found; for each such pair, the segments in the two tiles that do not overlap with the CMBR of the tiles are pruned before refining the contents of the tiles. Polysketch and PSCMBR focus on finding the intersection points of two objects, hence, unlike our approach, they do not identify true hits. The CMBR approach [2] is fully integrated in our implementation; still the refinement cost remains high. Finally, the Clipped Bounding Box (CBB) [26] is an enriched representation of the MBR that captures the dead (unused) space at MBR corners with a few auxiliary points, providing the opportunity of refinement step avoidance in the case where object CBBs intersect only at their common dead-space areas. CBBs can also be used by R-tree nodes to avoid their traversal if the query range overlaps only with their dead space.

Approximate spatial joins Fast evaluation of spatial joins and other operations based on raster (and other) approximations has been explored recently as an alternative to exact, but expensive spatial query evaluation [14, 35]. Still, there is no previous work that applies our idea of cell encoding and intervalization even for approximate query evaluation. The approximation of spatial objects using space-filling curves (and approximate evaluation of spatial queries) was first suggested by Orenstein [20], however, we are the first to suggest the binary encoding of cells and merging the codes to bitstrings for identifying true hits in spatial joins.

Scalability With the advent of cloud computing, there have been many efforts in scaling out spatial data management. SJMP [36] is an adaptation of the PBSM spatial join algorithm [23], which evaluates the join using the mapreduce framework. Using a virtual grid, each object is mapped to one or more partitions, based on the cells it intersects; for each partition, a reducer evaluates the join. Hadoop-GIS [3] employs a similar idea. Spatialhadoop [9] investigates the optimal repartitioning of already partitioned datasets in order for the join partitions to be perfectly aligned. Spatial data management systems that extend Apache Spark and handle (among other operations) spatial joins include Magellan [28], SpatialSpark [32], Simba [31], and Apache Sedona (formerly, GeoSpark) [33]. An experimental comparison between big spatial data systems was conducted by Pandey et al. [21]. Regarding spatial joins, all the aforementioned systems focus on the filter step only. Recently, there is a trend in implementing spatial joins for GPUs [1, 2, 15, 16], with a focus on the refinement step. Polygon decomposition and rasterization techniques for point-polygon joins using GPUs and CPUs have been explored in [14, 34].

6 CONCLUSIONS

In this paper, we proposed raster intervals (RI), a technique that encodes and compresses the raster approximations of polygons as sets of intervals, offering a fast and effective intermediate step between the filter and the refinement steps of polygon intersection joins. RI achieves a speedup of at least 3x compared to previous intermediate filters (raster approximations [37], 5C-CH [7]). At the same time, the space complexity of RI is relatively low and the approximations can easily be accommodated in main memory.

In the future, we will investigate more efficient rasterization techniques (e.g., using GPUs for both rasterization and parallel evaluation), compression schemes for the lists of intervals (e.g., run-length encoding), and the application of hierarchical raster approximations based on Hilbert code prefixes, for polygons of varying sizes. We will also study joins that involve other data types besides polygons (e.g., linestrings) as well as joins between object rasterizations of different granularity. In addition, we will explore the effectiveness of approximate join evaluation according with the recent trend [35]. Finally, we will investigate the application of RI filters to other spatial queries (e.g., non-rectangular range queries).

ACKNOWLEDGMENTS

T. Georgiadis was supported by project "Dioni: Computing Infrastructure for Big-Data Processing and Analysis." (MIS No. 5047222) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and EU. N. Mamoulis was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the "2nd Call for HFRI Research Projects to support Faculty Members & Researchers" (Project No. 2757).

REFERENCES

[1] Danial Aghajarian and Sushil K. Prasad. 2017. A Spatial Join Algorithm Based on a Non-uniform Grid Technique over GPGPU. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017*, Erik G. Hoel, Shawn D. Newsam, Siva Ravada, Roberto Tamassia, and Goce Trajcevski (Eds.). ACM, 56:1–56:4.

[2] Danial Aghajarian, Satish Puri, and Sushil K. Prasad. 2016. GCMF: an efficient end-to-end spatial join system over large polygonal datasets on GPGPU platform. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016, Burlingame, California, USA, October 31 - November 3, 2016.* ACM, 18:1–18:10.

[3] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel H. Saltz. 2013. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proc. VLDB Endow.* 6, 11 (2013), 1009–1020.

[4] Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, and Jeffrey Scott Vitter. 1998. Scalable Sweeping-Based Spatial Join. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Ashish Gupta, Oded Shmueli, and Jennifer Widom (Eds.). Morgan Kaufmann, 570–581.

[5] Wael M. Badawy and Walid G. Aref. 1999. On Local Heuristics to Speed Up Polygon-Polygon Intersection Tests. In *ACM-GIS '99, Proceedings of the 7th International Symposium on Advances in Geographic Information Systems, November 2-6, 1999, Kansas City, USA.* ACM, 97–102.

[6] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, May 23-25, 1990*, Hector Garcia-Molina and H. V. Jagadish (Eds.). ACM Press, 322–331.

[7] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1994. Multi-Step Processing of Spatial Joins. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, May 24-27, 1994*, Richard T. Snodgrass and Marianne Winslett (Eds.). ACM Press, 197–208.

[8] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. 1993. Efficient Processing of Spatial Joins Using R-Trees. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993.* ACM Press, 237–246.

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

[10] Yi Fang, Marc T. Friedman, Giri Nair, Michael Rys, and Ana-Elisa Schmid. 2008. Spatial indexing in microsoft SQL server 2008. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008.* 1207–1216.

[11] Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. 1996. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996.* 171–180.

[12] David Hilbert. 1891. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen* 38, 1 (1891), 459–460.

[13] Edwin H. Jacox and Hanan Samet. 2007. Spatial join techniques. *ACM Trans. Database Syst.* 32, 1 (2007), 7.

[14] Andreas Kipf, Harald Lang, Varun Pandey, Raul Alexandru Persa, Christoph Anneser, Eleni Tzirita Zacharatou, Harish Doraiswamy, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2020. Adaptive Main-Memory Indexing for High-Performance Point-Polygon Joins. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020.* OpenProceedings.org, 347–358.

[15] Yiming Liu and Satish Puri. 2020. Efficient Filters for Geometric Intersection Computations using GPU. In *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 3-6, 2020*, Chang-Tien Lu, Fusheng Wang, Goce Trajcevski, Yan Huang, Shawn D. Newsam, and Li Xiong (Eds.). ACM, 487–496.

[16] Yiming Liu, Jie Yang, and Satish Puri. 2019. Hierarchical Filter and Refinement System Over Large Polygonal Datasets on CPU-GPU. In *26th IEEE International Conference on High Performance Computing, Data, and Analytics, HiPC 2019, Hyderabad, India, December 17-20, 2019.* IEEE, 141–151.

[17] Nikos Mamoulis and Dimitris Papadias. 2003. Slot Index Spatial Join. *IEEE Trans. Knowl. Data Eng.* 15, 1 (2003), 211–231.

[18] Guy Macdonald Morton. 1966. A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing, Technical Report, Ottawa, Canada: IBM Ltd.

[19] Sadegh Nobari, Farhan Tauheed, Thomas Heinis, Panagiotis Karras, Stéphane Bressan, and Anastasia Ailamaki. 2013. TOUCH: in-memory spatial join by hierarchical data-oriented partitioning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013.* ACM, 701–712.

[20] Jack A. Orenstein. 1989. Redundancy in Spatial Databases. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, USA, May 31 - June 2, 1989.* ACM Press, 295–305.

[21] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. 2018. How Good Are Modern Spatial Analytics Systems? *Proc. VLDB Endow.* 11, 11 (2018), 1661–1673.

[22] George Papadakis, Georgios M. Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. 2021. Progressive, Holistic Geospatial Interlinking. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021.* ACM / IW3C2, 833–844.

[23] Jignesh M. Patel and David J. DeWitt. 1996. Partition Based Spatial-Merge Join. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, H. V. Jagadish and Inderpal Singh Mumick (Eds.). ACM Press, 259–270.

[24] Suprio Ray, Bogdan Simion, Angela Demke Brown, and Ryan Johnson. 2014. Skew-resistant parallel in-memory spatial join. In *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014.* ACM, 6:1–6:12.

[25] Michael Ian Shamos and Dan Hoey. 1976. Geometric Intersection Problems. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976.* IEEE Computer Society, 208–215.

[26] Darius Sidlauskas, Sean Chester, Eleni Tzirita Zacharatou, and Anastasia Ailamaki. 2018. Improving Spatial Data Processing by Clipping Minimum Bounding Boxes. IEEE Computer Society, 425–436.

[27] SpatialHadoop. 2015. *TIGER datasets.* http://spatialhadoop.cs.umn.edu/datasets.html

[28] Ram Sriharsha. [n. d.]. Magellan: Geospatial Analytics Using Spark. https://github.com/harsha2010/magellan.

[29] Dejun Teng, Furqan Baig, Qiheng Sun, Jun Kong, and Fusheng Wang. 2021. IDEAL: a Vector-Raster Hybrid Model for Efficient Spatial Queries over Complex Polygons. In *22nd IEEE International Conference on Mobile Data Management, MDM 2021, Toronto, ON, Canada, June 15-18, 2021.* 99–108.

[30] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. 2019. Parallel In-Memory Evaluation of Spatial Joins. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2019, Chicago, IL, USA, November 5-8, 2019.* ACM, 516–519.

[31] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 1071–1085.

[32] Simin You, Jianting Zhang, and Le Gruenwald. 2015. Large-scale spatial join query processing in Cloud. In *CloudDB, ICDE Workshops.* 34–41.

[33] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. 2019. Spatial data management in apache spark: the GeoSpark perspective and beyond. *GeoInformatica* 23, 1 (2019), 37–78.

[34] Eleni Tzirita Zacharatou, Harish Doraiswamy, Anastasia Ailamaki, Cláudio T. Silva, and Juliana Freire. 2017. GPU Rasterization for Real-Time Spatial Aggregation over Arbitrary Polygons. *Proc. VLDB Endow.* 11, 3 (2017), 352–365.

[35] Eleni Tzirita Zacharatou, Andreas Kipf, Ibrahim Sabek, Varun Pandey, Harish Doraiswamy, and Volker Markl. 2021. The Case for Distance-Bounded Spatial Approximations. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings.* www.cidrdb.org.

[36] Shubin Zhang, Jizhong Han, Zhiyong Liu, Kai Wang, and Zhiyong Xu. 2009. SJMR: Parallelizing spatial join with MapReduce on clusters. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing, August 31 - September 4, 2009, New Orleans, Louisiana, USA.* IEEE Computer Society, 1–8.

[37] Geraldo Zimbrao and Jano Moreira de Souza. 1998. A Raster Approximation For Processing of Spatial Joins. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA.* Morgan Kaufmann, 558–569.