

Scalable Management of Complex Spatial Data Types

Thanasis Georgiadis

Ph.D. Dissertation



Ioannina, September 2025



ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Scalable Management of Complex Spatial Data Types

A Dissertation

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Thanasis Georgiadis

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

University of Ioannina

School of Engineering

Ioannina 2025

Advisory Committee:

- **Nikos Mamoulis**, Professor, Department of Computer Science and Engineering, University of Ioannina (advisor)
- **Evaggelia Pitoura**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Vassilios V. Dimakopoulos**, Professor, Department of Computer Science and Engineering, University of Ioannina

Examining Committee:

- **Nikos Mamoulis**, Professor, Department of Computer Science and Engineering, University of Ioannina (advisor)
- **Evaggelia Pitoura**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Vassilios V. Dimakopoulos**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Panos Vassiliadis**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Eleni Tzirita Zacharatou**, Assistant Professor, Hasso Plattner Institute for Digital Engineering
- **Georgia Koutrika**, Research Director, Information Management Systems Institute (IMSI) of the Research and Innovation Centre in Information, Communication and Knowledge Technologies "Athena"
- **Manolis Koubarakis**, Professor, Department of Informatics and Telecommunications, National and Kapodistrian University of Athens

DEDICATION

To my beloved Dimitra.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my mentor and advisor, Prof. Nikos Mamoulis, for his guidance, support, and kindness throughout this challenging yet rewarding journey. Our collaboration and discussions were always fruitful and have significantly contributed to both my academic growth and personal development. Additionally, I would like to thank Giannis Pavlopoulos and Eleni Tziritza Zacharatou for generously sharing their knowledge, for our excellent collaboration, and for their support in helping me advance my work.

I am deeply grateful to Dimitra for her unwavering support and belief in me. She has continually encouraged me through the most difficult times and shared in my happiness when things went well. In the same spirit, I wish to thank my best friends, Alex and Giannis, for the priceless moments we have shared, their encouragement and trust in me, and for standing by my side through everything over the past 25 years.

I also want to thank my parents, my brother, and my sister for their understanding and constant support since the beginning of this journey.

Last but not least, I would like to thank my lab colleagues, Dimitris Tsitsigkos, Achilleas Michalopoulos, and Dimitris Dimitropoulos, for the many discussions, idea exchanges, collaborations, and hangouts we enjoyed together. Their companionship has been both stress-relieving and invaluable in broadening my perspective.

Thanasis Georgiadis

September 2025, Ioannina

TABLE OF CONTENTS

List of Figures	vi
List of Tables	x
List of Algorithms	xiii
Abstract	xiv
Εκτεταμένη Περίληψη	xvi
1 Introduction	1
1.1 Efficient In-Memory Spatial Joins with Complex Geometries	4
1.1.1 Raster Intervals: An Approximation Technique for Polygon In- tersection Joins	5
1.1.2 Advancing Raster Interval Approximations for Spatial Joins . . .	6
1.1.3 Scalable Spatial Topology Joins	7
1.2 Hecatoncheir: Scaling Up and Out Spatial Data Management	9
1.3 Spatial Reasoning using Retrieval-Augmented Generation	11
1.4 Dissertation Outline	13
2 Background & Related Work	15
2.1 Complex Geometries & Approximations	15
2.2 Spatial Joins	18
2.2.1 Join Predicates	18
2.2.2 Multi-step Evaluation	21
2.2.2.1 Intermediate Filtering	22
2.2.2.2 Speeding up the Refinement Step	23
2.2.3 Raster-based approaches for other queries	24

2.3	Scalable Spatial Data Management	26
2.3.1	Distributed Spatial Analytics Frameworks	27
2.3.2	MPI	29
2.3.3	Spatial joins on GPUs	29
2.4	Spatial Reasoning in LLMs through Retrieval-Augmented Generation .	30
2.4.1	Retrieval-Augmented Generation	30
2.4.2	Spatial Reasoning and GeoAI	31
2.4.3	Motivation	33
3	Efficient In-Memory Spatial Joins with Complex Geometries	36
3.1	An Approximation Technique for Polygon Intersection Joins	37
3.1.1	Raster Intervals	37
3.1.1.1	Object rasterization and raster encoding	38
3.1.1.2	Intervalization	39
3.1.1.3	Intermediate filter	40
3.1.1.4	“Within” spatial joins	43
3.1.2	Experimental Analysis	44
3.1.2.1	Datasets	45
3.1.2.2	The effect of N in RI	45
3.1.2.3	Data preprocessing	48
3.1.2.4	Performance in end-to-end spatial joins	49
3.2	Advancing Raster Interval Approximations for Spatial Joins	53
3.2.1	APRIL	53
3.2.1.1	A - and F -Interval Lists	53
3.2.1.2	APRIL Intermediate Spatial Join Filter	55
3.2.1.3	Generality	57
3.2.1.4	Selection Queries	57
3.2.1.5	Spatial Within Joins	57
3.2.1.6	Linestring to Polygon Joins	58
3.2.2	Customization	58
3.2.2.1	Compression	58
3.2.2.2	Partitioning	60
3.2.2.3	Different Granularity	60
3.2.3	APRIL Construction	62

3.2.3.1	Efficient Graphics-Inspired Rasterization	62
3.2.3.2	One-Step Intervalization	65
3.2.4	Experimental Analysis	68
3.2.4.1	Datasets	68
3.2.4.2	Optimizations and Customizations	69
3.2.4.2.1	The effect of N in RI	69
3.2.4.2.2	Join Order	70
3.2.4.2.3	Partitioning	72
3.2.4.2.4	Different Granularity	72
3.2.4.3	APRIL Construction Cost	74
3.2.4.4	Comparative Study	77
3.2.4.4.1	Space Complexity	78
3.2.4.4.2	Comparison in Spatial Intersection Joins	79
3.2.4.4.3	Effect of variance in object sizes	81
3.2.4.4.4	Performance in other queries	83
3.3	Scalable Spatial Topology Joins	85
3.3.1	MBR Filter	85
3.3.2	Intermediate Filter	85
3.3.3	Relate	89
3.3.4	Experimental Analysis	90
3.3.4.1	Datasets & Setup	91
3.3.4.2	Performance	93
3.3.4.3	Scalability	94
3.3.4.4	<i>Relate</i> Performance	96
3.4	Conclusions	97
4	Hecatoncheir: Scaling Up and Out Spatial Data Management	99
4.1	Architecture	100
4.1.1	Data Loading and Distribution	100
4.1.2	MPI Layer	101
4.1.3	Internal Layers	102
4.1.3.1	Boost Geometry	102
4.1.3.2	Index Layer	103
4.1.3.3	Query Processing Layer	103

4.1.4	API & Interface	106
4.1.4.1	C++ API	106
4.1.4.2	GUI	110
4.1.5	In-Memory Management	111
4.2	Experimental Evaluation	112
4.2.1	Setup	112
4.2.2	Datasets	112
4.2.3	Scalability	113
4.2.4	Performance	115
4.2.4.1	Range Queries	115
4.2.4.2	Spatial Intersection Joins	116
4.2.4.3	k NN Queries	116
4.2.4.4	Distance Joins	117
4.2.5	Profiling	117
4.3	Conclusions	122
5	Spatial Reasoning using Retrieval-Augmented Generation	124
5.1	SpaRAGraph Overview	125
5.1.1	Pipeline	126
5.1.2	Usage	126
5.2	Pre-processing	129
5.2.1	<i>SpaTex</i> : Spatial Relation RDF Generator	130
5.2.2	Graph-based Topology Index	133
5.2.3	Spatial Relation Composition matrix	134
5.3	Spatial Reasoning with SpaRAGraph	136
5.3.1	Graph Traversal	136
5.3.2	Context Generation	138
5.4	SRB - The Spatial Reasoning Benchmark	140
5.4.1	Source Data	141
5.4.2	Construction	143
5.4.2.1	Yes/No - Binary Classification	143
5.4.2.2	Radio - Multiclass Classification	144
5.4.2.3	Checkbox - Multilabel Classification	145
5.5	Experimental Evaluation	146

5.5.1	SpaRAGraph Generation Evaluation	148
5.5.1.1	Effectiveness	149
5.5.1.2	Zero- vs Few-shot	150
5.5.2	Retrieval Accuracy	152
5.6	Conclusions	153
6	Conclusions and Future Work	155
6.1	Summary of Contributions	155
6.2	Directions for Future Work	156
	Bibliography	159

LIST OF FIGURES

1.1	Topological relations and proposed workflow	8
2.1	Examples of conservative polygon approximations. (Image from [1]) . .	17
2.2	Examples of progressive polygon approximations. (Image from [1]) . .	17
2.3	Raster image of a polygon.	18
2.4	An intersection example between 2 raster approximations.	19
2.5	Venn diagram of the 8 most common topological relations.	20
2.6	Example of MBR intersection tests. The first pair of polygons do not intersect, however their MBRs do.	21
2.7	Common intersection query pipeline with a Filter and a Refinement stage.	22
2.8	Query pipeline with an intermediate step.	23
2.9	Examples of IDEAL and Raptor object approximations.	26
3.1	The Hilbert curve cell enumeration and interval generation for a poly- gon in a 8×8 space.	39
3.2	Two rasterized polygons, the overlaps between their raster intervals, and their common cells.	43
3.3	Intervals $[9, 13)$ and $[11, 15)$ 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).	44
3.4	The interval generation for a polygon in a 8×8 space, without bit-coding and using interval lists.	54
3.5	The three steps of the intermediate filter for a candidate pair of polygons.	56
3.6	A linestring's APRIL approximation size in bytes, if stored as intervals versus cells.	59

3.7	Example of a partition P , a group of polygons in it and P 's raster area with granularity order $N = 8$	61
3.8	The Scanline rendering algorithm we implemented, filling the Full cells without performing any PiP tests.	64
3.9	The flood fill algorithm, performing three iterations with different seeds c to completely fill all unchecked cells.	64
3.10	Example of the intervals/gaps for a set of Partial cells. Whether a gap will be labeled as Full or Empty depends on the outcome of the PiP test.	66
3.11	Filter effectiveness and spatial join cost for T1 dataset joins.	76
3.12	Filter effectiveness and spatial join cost for O5O6 AF and AS dataset joins.	77
3.13	Filter effectiveness and spatial join cost for O5O6 EU and NA dataset joins.	78
3.14	Filter effectiveness and spatial join cost for O5O6 SA and OC dataset joins.	79
3.15	The candidate and definite (bold) topological relations of two polygons, based on how their MBRs intersect.	86
3.16	The intermediate filters for different cases of intersections between the MBRs of r and s (Part 1).	88
3.17	The intermediate filters for different cases of intersections between the MBRs of r and s (Part 2).	89
3.18	Intermediate filter flow diagrams for <i>relate_p</i> tests.	90
3.19	Throughput and Effectiveness for <i>find relation</i>	93
3.20	<i>Find relation</i> filter effectiveness and total cost for polygons pairs grouped by complexity level.	95
3.21	A level-10 complexity pair of a lake (blue) residing inside a park (green), from the OLE-OPE scenario.	96
4.1	Hecatoncheir architecture overview.	100
4.2	Spatial partitioning using a coarse grid.	101
4.3	Communication setup.	102
4.4	The Two-Layer index on a Worker W 's local fine-grained grid.	104
4.5	Snapshot of spatial query execution through Hecatoncheir's GUI.	111

4.6	Per Worker (rank) profiling for the $O5 \bowtie O6$ spatial intersection join, regarding count of messages sent/received (a), memory (b), and query execution time (c).	119
4.7	Per Worker (rank) profiling for the $T2P \bowtie O3P$ distance join with $\epsilon = 0.001$, regarding count of messages sent/received (a), memory (b), and query execution time (c).	120
4.8	Per Worker (rank) profiling for the Q1 range queries on the O3P dataset, regarding count of messages sent/received (a), memory (b), and query execution time (c).	121
4.9	Per Worker (rank) profiling for the NN k NN queries on the O3P dataset with $k = 5$, regarding count of messages sent/received (a), memory (b), and query execution time (c).	123
5.1	SpaRAGraph’s overview divided into two stages: pre-processing, performed once and consisting of <i>SpaTex</i> ’s spatial RDF generation and their indexing in a graph; and usage, which illustrates user interaction with SpaRAGraph, showing how context is generated in the back-end through named entity recognition and path search on the RDF graph, before being appended to the prompt and sent to the LLM.	127
5.2	Example interaction with SpaRAGraph when given a spatial question. The context is automatically generated and appended to the prompt, to guide the model’s response.	127
5.3	Map of all counties in Kansas, illustrating possible shortest paths (yellow) between Dickinson County (red) and Douglas County (blue). These paths enable inference of the relative spatial relation between the two entities through intermediate connections. The map was created on https://www.mapchart.net	128
5.4	Llama-3.1-8B-Instruct response to an example spatial question.	129
5.5	Llama-3.1-8B-Instruct response to an example spatial question when employed with SpaRAGraph.	130
5.6	The Spatial relation identification process by <i>SpaTex</i> that uses a global grid to group nearby entities and compute their spatial relations, outputting them in RDF format.	131

5.7	An example graph constructed from a set of spatial RDF triplets. The map illustrates the spatial arrangement of objects in the data space, with dotted lines indicating the grid cells defined by <i>SpaTex</i> and the resulting bi-directional index built over them.	134
5.8	Path traversal and context generation for an example spatial question based on the data in Figure 5.7. SpaRAGraph identifies a 2-hop path, performs one directional composition, and generates context that accurately captures the spatial relation between the entities based on the map.	139
5.9	Path traversal and context generation for an example spatial question based on the data in Figure 5.7. SpaRAGraph identifies a 3-hop path, performs two directional compositions, and generates context that is approximately correct for entities D and E. in reality, D lies southwest of E based on the map.	140
5.10	Illustrative example showing the State of California and its Counties (a), with a focus on the Zipcodes within Santa Clara County, CA (b), to highlight the relative scale of the geographic divisions in our source data. (Visualized on QGIS [2].)	142
5.11	Overview of our pipeline for generating spatial questions from raw geographic data. The process involves calculating spatial relationships and transforming them into RDF triplets using <i>SpaTex</i> . Then, structured yes/no, single-choice, and multiple-choice questions are randomly generated from the set of RDF triplets.	143
5.12	Example of Yes/No random question generation from a randomly sampled RDF triplet. The correct answer is indicated with a bolded boundary.	144
5.13	Radio (single-choice) random question generation from a randomly selected RDF triplet. The correct answer is indicated with a bolded boundary.	145
5.14	Checkbox (multiple-choice) random question generation from all triplets regarding a randomly selected entity (Zipcode 16650). The correct answers are shown with a boundary.	146
5.15	F1, macro-F1 and sample-F1 scores (Y axis) of each model for Yes/No, Radio and Checkbox questions, respectively, in k -shot settings ($k \in [0, 3]$).	151

LIST OF TABLES

1.1	Hecatoncheir and Apache Sedona features list.	11
2.1	Do two objects intersect in a cell, based on the cell's types in the two raster approximations?	18
2.2	DE-9IM masks of topological relations.	20
2.3	An overview of related work on Spatial Reasoning and Geospatial LLM approaches and their features.	35
3.1	3-bit type codes for each input dataset	38
3.2	Statistics of the datasets and space requirements of the data and the approximations.	45
3.3	Preprocessing costs (in sec) of intermediate filter approximations	45
3.4	Effect of N on the performance of RI in $T1 \bowtie T2$	47
3.5	Effect of N on the cost and space of RI for $T1$ and $T2$	47
3.6	Statistics of RI approximations for OSM data	49
3.7	Performance of intermediate filters in spatial intersection joins	50
3.8	Statistics using Z-order curve	50
3.9	Filter performance in spatial within joins	53
3.10	APRIL: Do two objects intersect in a common cell?	54
3.11	Statistics of the datasets and space requirements of the data and the approximations	68
3.12	Effect of N on the performance of RI and APRIL in $T1 \bowtie T2$	71
3.13	Effect of N on the cost and space of RI and APRIL for $T1$ and $T2$. . .	71
3.14	Join order effect on APRIL filter cost.	72
3.15	# partitions per dimension effect on join time.	73
3.16	# of partitions per dimension, effect on APRIL size (MB).	74
3.17	Join between $T1$ (order 16) and $T3$ (order N).	74

3.18	Total construction cost (sec) for all datasets.	75
3.19	Improvement of APRIL over RI on all join pairs	81
3.20	End-to-end join performance between T2 and datasets with varying average object area.	82
3.21	TIGER dataset statistics, sorted by ascending average object MBR area.	82
3.22	APRIL vs. RI (polygonal range queries).	83
3.23	Performance of filters (spatial within joins)	84
3.24	Polygon-linestring spatial intersection joins.	84
3.25	Description of datasets.	91
3.26	Semantically meaningful dataset combinations for the <i>find relation</i> and <i>relate_p</i> problems.	92
3.27	OLE-OPE post-MBR filter polygon pairs, grouped by complexity level (sum of their vertex count).	94
3.28	Throughput (pairs/sec) of <i>find relation</i> and <i>relate_p</i> methods using our P+C approach (OLE-OPE).	97
4.1	Summary of datasets that were used in our experiments.	113
4.2	Scaling of Hecatoncheir’s query evaluation time (seconds) for spatial join, distance join and <i>k</i> NN query scenarios.	114
4.3	Scaling of Hecatoncheir’s query evaluation time (seconds) for the QX batch range queries on the T2P and O3P datasets.	115
4.4	Hecatoncheir and Apache Sedona comparison on Range Queries eval- uation time (seconds), in a cluster with 10 nodes.	116
4.5	Hecatoncheir and Apache Sedona comparison on Spatial Intersection Join evaluation time (seconds), in a cluster with 10 nodes.	116
4.6	Hecatoncheir and Apache Sedona comparison on <i>k</i> NN query evaluation time (seconds), for increasing <i>k</i> , in a cluster with 10 nodes.	117
4.7	Hecatoncheir and Apache Sedona comparison on Distance Join evalu- ation time (seconds) for increasing distance value ϵ , in a cluster with 10 nodes.	118
5.1	Relation composition matrix with a right arrow ($\rightarrow$) denoting left-to- right composition.	135
5.2	Overview of the different question types in the dataset, each illustrated with an example.	142

5.3	Breakdown of question datasets by type of entities referenced in questions.	147
5.4	The models that we tested on SpaRAGraph.	148
5.5	Comparison of models' average execution time after three runs, in seconds $\pm$ the standard error of the mean rounded up, on the Spatial Reasoning Benchmark, with and without SpaRAGraph in zero-shot setting.	149
5.6	Performance comparison of models on the Spatial Reasoning Benchmark, with and without SpaRAGraph in zero-shot setting. The scores (%) are the averages across three different response generation cycles and the standard error of the mean across all scores does not exceed 1%.150	
5.7	SpaRAGraph's retrieval accuracy expressed through its MRR, FMR (percentage of prompts that had all their entities retrieved correctly) and Precision (percentage of all retrieved entities that are correct for their respective prompts). Metrics are shown per individual dataset, as Yes/No and Radio questions always regard two entities each, whilst SRB's Checkbox questions always pertain five entities.	153

LIST OF ALGORITHMS

3.1	RI-join algorithm	41
3.2	APRIL join algorithm.	56
3.3	The One-Step Intervalization algorithm.	67
3.4	Evaluation of <i>find relation</i>	87

ABSTRACT

Thanasis Georgiadis, Ph.D., Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2025.

Scalable Management of Complex Spatial Data Types.

Advisor: Nikos Mamoulis, Professor.

Scalable spatial data management is crucial in both scientific and commercial domains, particularly in Geographic Information Systems (GIS), which handle massive volumes of geographic data. As spatial data continues to grow rapidly, the demand for efficient spatial data analytics tools has become increasingly pressing. A core functionality of such tools is the computation of spatial and topological joins over large collections of objects. These operations aim to identify intersecting object pairs (i.e., objects that share at least one common point), a fundamental task with applications in geospatial interlinking, spatial databases, and beyond. However, intersection testing is computationally intensive, especially for polygonal objects, which often contain a large number of vertices and require costly geometric processing.

This dissertation investigates approximation techniques for handling high-complexity polygons, with the aim of making processing faster and more efficient. The central objective is to minimize reliance on original geometries, using them for computations only as a last resort. Our proposed solutions introduce efficient polygon approximation methods with a low memory footprint, along with filtering techniques that enable spatial joins to be evaluated without directly accessing the original geometries. The work addresses both scalability and accuracy challenges while striving to deliver solutions that are directly applicable to modern in-memory spatial databases.

Scalable spatial data management has two key aspects. First, query processing algorithms must be highly parallelizable and independent, enabling them to fully leverage distributed and parallel spatial databases for both vertical and horizontal scalability. Second, they must maintain efficiency as geometric complexity increases,

since complex shapes often become a major bottleneck in spatial query processing. In the second part of this dissertation, we design and implement a prototype distributed spatial data management framework that operates independently of underlying engines, focusing specifically on the performance and scalability of spatial query evaluation in tightly coupled clusters. The prototype integrates state-of-the-art indexing, approximation, and filtering techniques while carefully minimizing both communication overhead and memory usage.

With the rapid advancement of Large Language Models (LLMs) and their expanding use across diverse domains, questions arise regarding their ability to handle complex tasks, particularly spatial reasoning over text. While LLMs excel at inferring and extracting information from large text collections, spatial knowledge is often domain-specific and not inherently intuitive for them. The first limitation is commonly mitigated through Retrieval-Augmented Generation (RAG), where external databases provide context at inference time, enhancing factual accuracy in responses. However, without fine-tuning or re-training, which can be costly and counterproductive to the goal of broad generalization, LLMs have consistently demonstrated weak performance on spatial reasoning tasks.

In the final part of this dissertation, we investigate how topological relations can help LLMs generate correct responses to spatial reasoning questions expressed in text. We first employ our efficient spatial topology algorithms to scalably compute key inter-dataset spatial relations and represent them as RDF (text) triplets. These are then leveraged through RAG-based mechanisms and indexing techniques to enable fast and accurate spatial context retrieval at inference time, with the dual objective of (i) supplying the LLM with domain-specific spatial knowledge and (ii) supporting it in producing factually correct responses.

In summary, this dissertation presents a comprehensive study of scalable spatial data management. It introduces in-memory solutions that are both efficient and accurate, addressing key challenges across a broad spectrum of use cases. The proposed approaches are directly applicable to modern spatial databases and well-suited for data-intensive geospatial applications.

ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ

Θανάσης Γεωργιάδης, Δ.Δ., Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2025.

Τίτλος στα Ελληνικά.

Επιβλέπων: Νίκος Μαμουλής, Καθηγητής.

Η κλιμακώσιμη διαχείριση χωρικών δεδομένων είναι κρίσιμη τόσο στον επιστημονικό όσο και στον εμπορικό τομέα, ιδιαίτερα στα Γεωγραφικά Συστήματα Πληροφοριών (GIS), τα οποία διαχειρίζονται τεράστιους όγκους γεωγραφικών δεδομένων. Καθώς τα χωρικά δεδομένα συνεχίζουν να αυξάνονται ραγδαία, η ανάγκη για αποδοτικά εργαλεία χωρικής ανάλυσης γίνεται ολοένα και πιο επιτακτική. Βασική λειτουργικότητα τέτοιων εργαλείων αποτελεί ο υπολογισμός χωρικών και τοπολογικών συνενώσεων (joins) σε μεγάλες συλλογές οντωτήτων. Οι πράξεις αυτές αποσκοπούν στον εντοπισμό ζευγών που τέμνονται (δηλαδή αντικείμενα που μοιράζονται τουλάχιστον ένα κοινό σημείο), μια θεμελιώδης διαδικασία με εφαρμογές στη γεωχωρική διασύνδεση, στις χωρικές βάσεις δεδομένων και πέραν αυτών. Ωστόσο, ο έλεγχος τομής είναι υπολογιστικά δαπανηρός, ιδιαίτερα για πολυγωνικά αντικείμενα, τα οποία συχνά περιέχουν μεγάλο αριθμό κορυφών και απαιτούν κοστοβόρα γεωμετρική επεξεργασία.

Η παρούσα διατριβή διερευνά τεχνικές προσεγγιστικής αναπαράστασης για την αποδοτική διαχείριση πολύπλοκων πολυγώνων, με στόχο την ταχύτερη και πιο αποτελεσματική επεξεργασία. Ο κεντρικός στόχος είναι η ελαχιστοποίηση της εξάρτησης από τις αρχικές γεωμετρίες, χρησιμοποιώντας τις μόνο ως έσχατη λύση για υπολογισμούς. Οι προτεινόμενες λύσεις εισάγουν αποδοτικές μεθόδους προσεγγιστικής αναπαράστασης πολυγώνων με μικρό αποτύπωμα μνήμης, καθώς και τεχνικές φιλτραρίσματος που επιτρέπουν την εκτέλεση χωρικών συνενώσεων χωρίς άμεση πρόσβαση στις αρχικές γεωμετρίες. Η εργασία αντιμετωπίζει τόσο ζητήματα επεκτασιμότητας όσο και ακρίβειας, ενώ παράλληλα επιδιώκει την ανάπτυξη λύσεων

άμεσα εφαρμόσιμων σε σύγχρονες χωρικές βάσεις δεδομένων στη μνήμη.

Η κλιμακώσιμη διαχείριση χωρικών δεδομένων έχει δύο βασικές διαστάσεις. Πρώτον, οι αλγόριθμοι επεξεργασίας ερωτημάτων πρέπει να είναι υψηλά παραλληλοποιήσιμοι και ανεξάρτητοι, ώστε να αξιοποιούν πλήρως τις κατανεμημένες και παράλληλες χωρικές βάσεις δεδομένων τόσο για κάθετη όσο και για οριζόντια κλιμάκωση. Δεύτερον, πρέπει να διατηρούν την αποδοτικότητά τους καθώς αυξάνεται η γεωμετρική πολυπλοκότητα, δεδομένου ότι τα πολύπλοκα σχήματα αποτελούν συχνά σημαντικό σημείο συμφόρησης στην επεξεργασία χωρικών ερωτημάτων. Στο δεύτερο μέρος της διατριβής, σχεδιάζουμε και υλοποιούμε ένα πρωτότυπο κατανεμημένο πλαίσιο διαχείρισης χωρικών δεδομένων, το οποίο λειτουργεί χωρίς εξάρτηση σε υποκείμενα συστήματα, εστιάζοντας ειδικά στην απόδοση και την κλιμακωσιμότητα της εκτέλεσης χωρικών ερωτημάτων σε στενά συνδεδεμένα, κατανεμημένα υπολογιστικά συστήματα. Το πρωτότυπο ενσωματώνει σύγχρονες τεχνικές δεικτοδότησης, προσεγγιστικής αναπαράστασης και φιλτραρίσματος, ελαχιστοποιώντας παράλληλα την επικοινωνιακή επιβάρυνση και τη χρήση μνήμης.

Με τη ραγδαία πρόοδο των Μεγάλων Γλωσσικών Μοντέλων (LLMs) και την αυξανόμενη χρήση τους σε ποικίλους τομείς, προκύπτουν ερωτήματα σχετικά με την ικανότητά τους να χειρίζονται πολύπλοκες εργασίες, και ειδικότερα τη χωρική συλλογιστική μέσω κειμένου. Αν και τα LLMs διαπρέπουν στην εξαγωγή και επεξεργασία πληροφορίας από μεγάλες συλλογές κειμένων, η χωρική γνώση είναι συχνά εξειδικευμένη και όχι εγγενώς διαισθητική για αυτά. Ένας βασικός περιορισμός μετριάζεται μέσω της μεθόδου Retrieval-Augmented Generation (RAG), όπου εξωτερικές βάσεις δεδομένων παρέχουν συμπραζόμενα κατά το χρόνο εκτέλεσης, βελτιώνοντας την πραγματολογική ακρίβεια των απαντήσεων. Ωστόσο, χωρίς προσαρμοστική εκπαίδευση ή επανεκπαίδευση, διαδικασίες που είναι δαπανηρές και αντίθετες προς τον στόχο της γενικής χρήσης, τα LLMs εμφανίζουν σταθερά αδύναμες επιδόσεις σε χωρικές συλλογιστικές εργασίες.

Στο τελικό μέρος της διατριβής, διερευνούμε πώς οι τοπολογικές σχέσεις μπορούν να βοηθήσουν τα LLMs να παράγουν ορθές απαντήσεις σε ερωτήσεις χωρικής συλλογιστικής που εκφράζονται σε κείμενο. Αρχικά, εφαρμόζουμε τους αποδοτικούς αλγόριθμους χωρικής τοπολογίας που αναπτύξαμε για να υπολογίσουμε με κλιμακώσιμο τρόπο βασικές δια-συνόλου χωρικές σχέσεις και να τις αναπαραστήσουμε ως RDF τριάδες (κειμενικές). Στη συνέχεια, αυτές αξιοποιούνται μέσω μηχανισμών RAG και τεχνικών δεικτοδότησης, ώστε να καταστεί δυνατή η ταχεία και ακριβής

ανάκτηση χωρικών συμφραζομένων κατά την εκτέλεση, με διττό στόχο: (i) τον εφοδιασμό του LLM με εξειδικευμένη χωρική γνώση και (ii) την υποστήριξή του στην παραγωγή πραγματολογικά ορθών απαντήσεων.

Συνοψίζοντας, η διατριβή αυτή παρουσιάζει μία ολοκληρωμένη μελέτη για την κλιμακώσιμη διαχείριση χωρικών δεδομένων. Εισάγει λύσεις στη μνήμη που είναι ταυτόχρονα αποδοτικές και ακριβείς, αντιμετωπίζοντας κρίσιμες προκλήσεις σε ευρύ φάσμα περιπτώσεων χρήσης. Οι προτεινόμενες προσεγγίσεις είναι άμεσα εφαρμόσιμες σε σύγχρονες χωρικές βάσεις δεδομένων και κατάλληλες για εφαρμογές εντατικής γεωχωρικής επεξεργασίας δεδομένων.

CHAPTER 1

INTRODUCTION

1.1 Efficient In-Memory Spatial Joins with Complex Geometries

1.2 Hecatoncheir: Scaling Up and Out Spatial Data Management

1.3 Spatial Reasoning using Retrieval-Augmented Generation

1.4 Dissertation Outline

Database management systems (DBMSs) [3, 4] are the cornerstone of modern data organization. Across all domains and applications, efficient data management and analysis are essential for driving innovation and improving people’s lives. From social networks and healthcare to scientific research and industrial manufacturing, database systems empower users to interact with data in a reliable, robust, and efficient way.

Data can vary not only in volume but also in modality. Additionally, access patterns and usage requirements often differ across application domains. As a result, databases are categorized into different types based on how they organize and process data. The most common and widely used are relational databases, which structure data in tables composed of rows and columns. These tables represent entities or relationships within the data, offering a structured and efficient way to store and manage records, particularly suited for transaction processing, where entire records are typically inserted, retrieved, or deleted. Spatial databases are designed for managing geographic and spatial data, typically organizing information based on geographic coordinates or shapes. They process data using topological relationships, such as containment and intersection, alongside associated relational attributes, making them

a powerful tool for spatial analytics. Graph databases represent data as nodes and edges within a graph structure, while temporal databases specialize in storing time-associated data, enabling the tracking of history and changes over time, and supporting time-based queries. Another prominent type is column-store databases, which organize data by attributes (columns) rather than by records (rows). This layout enables faster retrieval and processing, making it ideal for analytical workloads where queries typically target specific attributes rather than entire records.

With the rise of cloud computing services, distributed database management systems (DDBMSs) have become increasingly available and popular, enabling broader and more scalable data organization approaches. Data warehouses [5] consolidate and store data from multiple sources into a unified repository, supporting analysis and insight extraction over both current and historical data. In contrast, data lakes store data in its original, raw format, serving as a flexible repository that integrates data across all modalities that is useful for visualization, machine learning, and other advanced analytics.

Spatial Databases are primarily used in applications such as geographic information systems (GIS), location-based services (e.g., Uber, GPS), and mapping applications (e.g., Google Maps). Spatial data is organized based on geographic coordinates or other location-based 2D or 3D positioning, often using geometric shapes to represent real-world regional entities. As geometries increase in both complexity and quantity, spatial databases face significant challenges in storing all data in memory and processing it efficiently. To address these issues, advanced indexing techniques are essential for fast data retrieval, along with specialized algorithms to ensure effective data processing.

In-Memory Management of data is closely tied to proper organization, which is essential for both efficient retrieval and processing. Indexing plays a critical role in structuring how data is stored, enabling fast access based on its attributes. These attributes vary depending on the data modality. For instance, relational data is stored in tables, where entities or relationships group related attributes together. Temporal data, on the other hand, is typically organized around timestamps, which may be associated with other non-temporal attributes. Consequently, such data is indexed based on its temporal characteristics to support time-based retrieval.

Spatial data, on the other hand, is almost always indexed based on its coordinate properties using specialized indexes [6, 7], enabling spatial pruning. This means that

when specific retrieval parameters are provided, the database can avoid scanning all objects in the data space. Instead, it can quickly and efficiently exclude entire regions that are guaranteed not to contain any candidate objects (i.e., potential retrieval results). However, complete spatial datasets containing large and complex geometries (e.g., polygons) may not fit entirely in memory. As a result, non-point data is often stored on disk, while approximations of the geometries, such as the *minimum bounding rectangle* (MBR), are kept in memory. This approach enables selective loading of geometries from disk during query evaluation, which is especially important when memory is limited. However, it comes at the cost of increased I/O overhead.

Parallel Processing is increasingly important in modern databases due to significant advancements in hardware, such as multi-core processors, faster intra-process communication and specialized processing units, as well as the growing accessibility of these resources. Additionally, cloud services enable the scale-out of workloads across multiple machines in disaggregated environments. However, databases do not scale automatically, as both the indexing methods and the data processing algorithms they support have to be parallelizable.

Depending on the database environment, parallelism can be achieved at different levels. In shared-memory architectures, multiple processors (or threads) have access to a common memory space. This setup enables fast data retrieval and low-latency communication between processors. However, it also requires careful concurrency control to ensure data consistency. While parallel reads are generally safe and efficient, simultaneous reads and writes (or concurrent writes) can lead to race conditions and data corruption. To prevent such conflicts, locking mechanisms or synchronization primitives (e.g., mutexes, semaphores) are employed. However, these mechanisms can introduce significant overhead and reduce the degree of parallelism, as processors may be forced to wait for access to shared memory regions.

In contrast, distributed-memory architectures allocate private memory to each processor, meaning a processor can access only its local memory directly. When a processor requires data stored in another processor's memory, communication must occur explicitly. This is typically handled through *remote memory access* (RMA), as seen in paradigms like the *Message Passing Interface* (MPI) [8, 9], or through *Remote Direct Memory Access* (RDMA) techniques, which allow one processor to read or write directly to another processor's memory without involving the remote processor's CPU.

The choice between shared-memory and distributed-memory models and the

mechanisms used for data access, depends on the system’s architecture, the capabilities of the interconnect network (e.g., bandwidth, latency), and the workload characteristics. Modern high-performance systems often adopt hybrid architectures, combining shared memory within nodes and message-passing between nodes, to balance speed, scalability, and resource isolation.

1.1 Efficient In-Memory Spatial Joins with Complex Geometries

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 (www.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 www.freegisdata.rtwilson.com. With the growing spatial data availability, there is an increasing need for efficient spatial data analysis tools.

We study the problem of computing the spatial intersection join [10] between two spatial object collections R and S , which identifies all pairs of objects $(r, s), r \in R, s \in S$ such that r shares at least one common point with s . Besides being a common operation in GIS, the spatial intersection join finds a wide range of applications in geo-spatial interlinking [11], GeoSPARQL queries on RDF data stores [12], interference detection between objects in computer graphics [13], and suggesting synapses between neurons in neuroscience models [14]. In computer graphics (e.g., solid modeling, molecular modeling) they are used for detecting the interference between geometric models [15]. Recently, there has been a growing interest in spatial query evaluation over complex object geometries, such as polygons [16, 17, 18, 19, 20, 21, 22, 23, 24, 25].

A naive way to evaluate the join is to run an intersection test algorithm from computational geometry for each pair (r, s) in $R \times S$. However, this method is extremely expensive, since (i) the number $|R \times S|$ of pairs to be tested can be huge and (ii) for each pair the test takes $O(n \log n)$ time [1]. To mitigate (i), the join is evaluated in two steps. Provided that the minimum bounding rectangles (MBRs) of the objects are available (and possibly indexed), in the *filter step*, an efficient MBR-join algorithm [26, 27] is used to find the pairs of objects $(r, s) \in R \times S$ such that $MBR(r)$ intersects with $MBR(s)$. In the *refinement step*, for each pair that passes the filter step, the expensive intersection test on the exact object geometries is applied. To further

reduce the number of pairs needing refinement, *intermediate filters* can be added to the pipeline [1, 28]. The main idea is to use object approximations, in addition to the MBR, that can help to quickly determine whether a candidate pair (r, s) that passes the MBR filter is (i) a sure result, (ii) a sure non-result, or (iii) an indecisive pair, for which we still have to apply the geometry intersection test. Brinkhoff et al. [1] investigated the use of different object approximations (e.g., the convex hull) to be used as subsequent filters after MBR-intersection. Zimbrao and de Souza [28] proposed a more effective *raster* object approximation, where each object MBR is partitioned using a grid and the object is approximated by the percentages of grid 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.

1.1.1 Raster Intervals: An Approximation Technique for Polygon Intersection Joins

We propose *Raster Intervals* (RI); a raster approximation technique for polygonal objects, which does not share the drawbacks of [28] 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 [28]. 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.

Contributions. 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. Additionally, 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.

1.1.2 Advancing Raster Interval Approximations for Spatial Joins

Despite its effectiveness and efficiency compared to previous filters, RI has a relatively high preprocessing cost and occupies significant space. In this extended version of [29] we propose APRIL (Approximating Polygons as Raster Interval Lists), a significant improvement over RI. Unlike [28, 29] that divide the raster cells intersecting a polygon into three classes, APRIL uses only two cell classes, which improves storage efficiency and accelerates the intermediate filter. Second, the main novelty of APRIL lies in the way it represents objects using *two lists of intervals*: the first (*A*-list) includes all cells, regardless of their class, and the second (*F*-list) includes only cells that are fully covered by the object. The intermediate filter is then implemented as a sequence of three simple merge joins between the sorted interval lists of a given object pair. The first join, performed between the two *A*-lists, effectively identifies all true negatives. The last two joins, performed between one object’s *A*-list and the other object’s *F*-list, identify true positives. Since APRIL does not explicitly store or encode cell-class information and does not perform cell-specific comparisons, it is significantly faster than previous raster approximations. Finally, APRIL applies a compression technique based on delta encoding to greatly reduce the space required to store the interval lists. As a result, APRIL approximations may require even less space than object MBRs, allowing them to be stored and processed in main memory. Moreover,

APRIL’s compression scheme allows partial, on-demand decompression of interval lists during interval join evaluation.

Contributions. In addition to improving RI, we show the generality of APRIL in supporting spatial selection queries, spatial within joins, and joins between polygons and linestrings. Furthermore, we present a space partitioning approach, which increases the resolution of the raster grid and achieves more refined object approximations as necessary, leading to fewer inconclusive cases and, therefore, faster query evaluation. We also investigate options for defining and joining APRIL approximations of different polygons at different granularities based on their geometries. Finally, a significant contribution of this work is a novel, one-step “intervalization” algorithm that computes the APRIL approximation of a polygon without having to rasterize it in full. We show that this method is orders of magnitude faster compared to other rasterization approaches on CPU [28, 23].

1.1.3 Scalable Spatial Topology Joins

Topological relations between geometrical objects, illustrated in Figure 1.1(a) capture semantics that are unaffected by transformations of the data space, such as translation, rotation, and scaling. In GIS [30], they can be used in urban and transportation planning [31]. In *environmental studies*, detecting topological relations on climate, sustainability, and energy data (such as data offered by the European Environment Agency (EEA) [32]), can help understanding factors contributing to urban and global heating, pollution, and biodiversity decline. In *spatial databases* [33], topological relations are often used as predicates in selection queries and spatial joins [1]; besides, they have also been used in spatial query optimization [34]. In *geo-spatial interlinking* [35, 11], they are used to enrich and integrate knowledge graph databases with links between spatial entities.

In *image and multimedia databases*, relations between detected objects are valuable features, e.g., in medical image analysis [36]. Besides, content-based image retrieval based on topological queries [37, 38] finds interesting arrangements of objects in images. Proteins in large biological databases are also topologically related [39].

The Dimensionally Extended 9-intersection model (DE-9IM) [40, 41] is the de-facto standard for capturing the possible spatial relations between shapes. DE-9IM has been widely used in GIS (ArcGIS [42], QGIS [2]) and spatial DBMS (PostGIS [43],

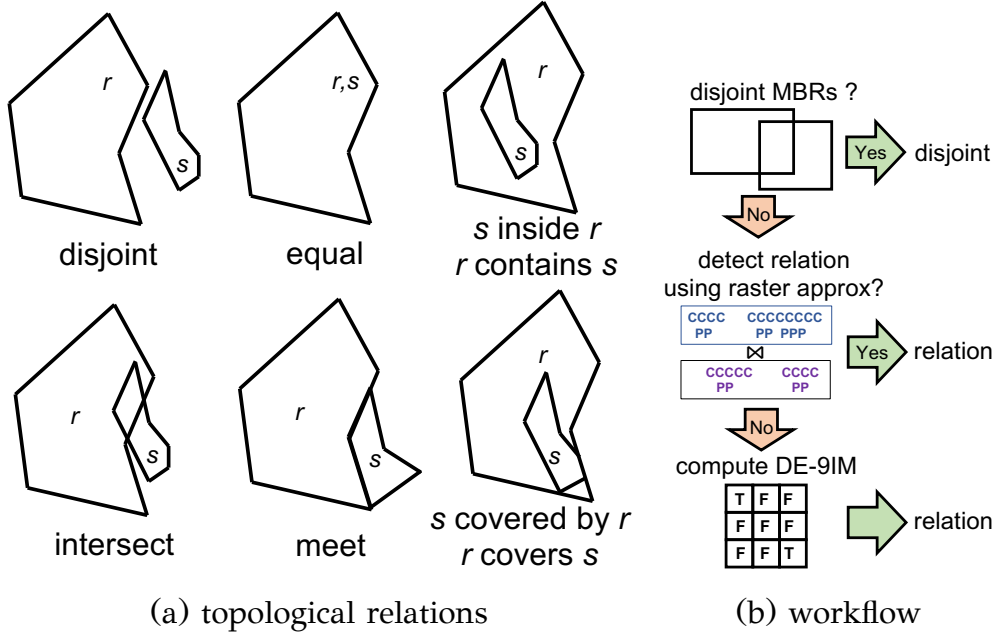


Figure 1.1: Topological relations and proposed workflow

Oracle Spatial [44]), while it has been implemented in popular geometry libraries (JTS [45], GEOS [46], boost [47]).

Our proposed APRIL approximation and filtering can detect intersecting pairs in the broad sense, without distinguishing more precise topological relations such as those illustrated in Figure 1.1(a). A naive workaround would be to forward the pairs identified as true hits by the APRIL filter to refinement, in order to compute their DE-9IM matrix and determine their exact topological relation. However, this process requires $O(n \log n)$ time (the same complexity as a polygon–polygon intersection test) where n is the number of vertices of the two shapes [48]. This approach is sub-optimal, since an effective intermediate filter should ideally detect both true hits and true negatives to minimize refinement costs. In this case, however, the filter would only detect true negatives, thereby undermining its purpose.

The workflow of our proposed approach is illustrated in Figure 1.1(b). For each pair of objects whose MBRs intersect (i.e., pairs produced by an algorithm handling the filter step [49, 27]), we perform merge-join operations on their APRIL approximations, modeled as interval lists, to potentially determine their topological relationship. If the raster approximations are insufficient, we compute the DE-9IM matrix as a fallback. A key advantage of our method is its scalability, i.e. its effectiveness improves with increasing object complexity, enabling the detection of hundreds of thousands of

topological relations per second for complex object pairs. Additionally, for most comparisons, our approach avoids loading full object geometries, significantly reducing data access costs.

Contributions. We propose an intermediate step in the pipeline of topology detection, after the MBR filter step, which takes advantage of precomputed spatial object approximations as lists of raster intervals [50]. We define a set of *relations* (e.g., overlap) between lists of raster intervals, which can be evaluated in linear time by merge-joining the lists. Then, depending on how the MBRs of two objects intersect, we propose specialized filters, which verify a sequence of list relations to potentially confirm the topological relation between the objects, without having to access and process their exact geometries. Hence, compared to previous work which uses raster object approximations [28, 29, 50] to detect spatial *intersection* only (including RI and APRIL), we exploit such approximations in full for the detection of the most specific topological relation between two objects. When applied on benchmarking datasets (Tiger, OSM [51]), our approach boosts the overall throughput of spatial topology joins up to one order of magnitude compared to state-of-the-art geo-spatial interlinking methods [35, 11] and up to several times compared to using raster approximations for intersection detection [50].

1.2 Hecatoncheir: Scaling Up and Out Spatial Data Management

Existing distributed spatial libraries such as Apache Sedona [52], SpatialHadoop [53], SIMBA [54], LocationSpark [55] and more, require Spark and Hadoop to be setup a priori in the cluster in order to be deployed upon them. Additionally, the architectures of these frameworks were specifically designed to integrate seamlessly with their underlying engines. As a result, they not only inherit the benefits of those engines but also their limitations. Moreover, spatial libraries like JTS [45], GEOS [46] and Google S2 are limited to providing APIs for geometry operations, but they do not constitute standalone systems for distributed indexing, data partitioning, and query evaluation. Several distributed spatial analytics frameworks have been compared against each other [21], with Apache Sedona being considered the most prominent and popular one. All these frameworks share the following drawbacks:

Resource cost The spatial indexes provided by existing frameworks are becoming outdated, with high construction and usage costs. Their memory usage for large

datasets often reaches tens of gigabytes, while query throughput hovers around a few hundred per minute [21]. In today’s era of cloud services and pay-as-you-go pricing models, these limitations can significantly inflate operational costs for both users and enterprises.

Setup Complexity They are based on an underlying engine (e.g., Spark) which needs to be installed beforehand. The engines come with their own dependencies; for example, cluster managers such as YARN or Kubernetes. On top of that, the frameworks have their own installation process as well and need to be configured to work along with their underlying engine. Hence, they cannot be considered as ”plug-and-play” tools to the casual user.

No C/C++ support Currently, no distributed spatial data management framework offers C/C++ support for their API, hence they require tedious low-level porting. C/C++ are usually the go-to language option for performance-focused implementations and, thus, there has been a huge gap in distributed spatial data management in C/C++ until now.

Contributions. We introduce Hecatoncheir, the first plug-and-play C/C++ library for in-memory distributed and parallel spatial data management. Hecatoncheir’s underlying layer is implemented using MPICH [9], a Message Passing Interface (MPI) [8] standard. Hecatoncheir offers optimized spatial partitioning and indexing techniques to support scalable distributed spatial query processing, without depending on external process and resource managers or engines. It functions like any other C/C++ library, compiled and linked using CMake, seamlessly integrating into the user’s projects while encapsulating all distribution-related complexities within a black-box logic for a plug-and-play experience. Currently, Hecatoncheir uses Boost Geometry [56] for the geometric operations, as we have found that it outperforms GEOS. To scale CPU-intensive tasks, such as data partitioning, spatial approximation generation [29, 50], and spatial query evaluation [57, 50], Hecatoncheir leverages intra-node parallelism with OpenMP, utilizing available threads to parallelize computationally intensive yet independent tasks on the CPU. By distributing and maintaining data in the main memory of each node, the system fully utilizes available resources for efficient, scalable query execution, where each machine operates *independently*, minimizing computation and communication overhead. We evaluate Hecatoncheir’s scalability, performance, and memory efficiency, via a comparison against Apache Sedona. An overview of both frameworks’ features can be seen in Table 1.1.

Table 1.1: Hecatoncheir and Apache Sedona features list.

Feature	Apache Sedona	Hecatoncheir
Language	Java/Scala	C/C++
Index	RTree, QuadTree	Two-level Grid
Queries	Range, (Distance) Join, kNN	Range, (Distance) Join, kNN

1.3 Spatial Reasoning using Retrieval-Augmented Generation

Recently, generative models such as large language models (LLMs), begun to be explored as AI assistants in GIS applications, where spatial reasoning plays a central role in interpreting and analyzing geographic data. Retrieval augmented generation (RAG) [58] improves the performance of LLMs by retrieving relevant information from external sources and providing it as context to improve the response quality of the models. RAG has been especially useful when we need to generate responses based on large and complex sources of knowledge that have not been used in the model training process. The success of RAG has brought opportunities for new research in data management and information retrieval toward improving LLM effectiveness [59].

Spatial data collections are typically in structured format and stored in database systems such as PostgreSQL¹ and Oracle Spatial², or GIS like QGIS³. The relations between all pairs of spatial data entities on a map are not explicitly stored or used in the training process of a foundation model, so existing models are not trained with such knowledge. Only a limited number of spatial relations are typically found in training sources of LLMs such as Wikipedia articles or public-domain books. For instance, the introductory paragraph of the Wikipedia entry on Greece states: *”Greece, officially the Hellenic Republic, is a country in Southeast Europe. Located on the southern tip of the Balkan Peninsula, Greece shares land borders with Albania to the northwest, North Macedonia and Bulgaria to the north, and Turkey to the east. The Aegean Sea lies to the east of the mainland, the Ionian Sea to the west, and the Sea of Crete and the Mediterranean Sea to the south.”* This passage conveys a significant amount of spatial information in natural language about Greece’s relative topology and location to its neighboring geographic

¹<https://www.postgresql.org/>

²<https://www.oracle.com/database/spatial/>

³<https://www.qgis.org/>

entities (e.g., countries and seas). In contrast, spatial databases may include precise coordinate data and geometrical representations spatial entities, but often lack such relational descriptions, especially in terms of natural, human-interpretable spatial context. However, when a spatial relation involves two entities that are not direct neighbors, such information is more than likely to not be explicitly stated in any pre-training data. In such cases, the relative spatial relation must be computed at inference time. This introduces additional complexity and latency, as it requires a dedicated spatial database system to be integrated with the LLM with a supporting toolchain, capable of categorizing the user’s question, translating it into a database query, executing the query on the spatial dataset, and returning the result. Only then can the LLM compose a meaningful and accurate natural language response grounded in factual spatial data. Moreover, when the spatial entities involved are less well-known, such as ZIP codes or parks, or belong to more specialized domains, like soil quality maps or mineral deposit distributions, it is far less likely that their spatial relations are explicitly described in any text used during pre-training, posing a significant challenge for LLMs to infer accurate spatial knowledge solely from prior textual data.

These observations motivate the central research question explored in this work: *To what extent can LLMs infer spatial knowledge over text, and how can this capability be enhanced?* To address this question, we introduce *SpaRAGraph*, a framework that enhances language model generation through RAG by supplying spatially enriched context. This enables the inference of spatial knowledge from text without requiring model re-training or fine-tuning. The first major challenge lies in the nature of spatial data: it is typically stored in structured, non-linguistic formats (e.g., records or geometries), making it inherently incomprehensible to LLMs in its raw form. Therefore, a spatial-to-text pre-processing step is required to translate this data into natural language descriptions that can both test and improve LLMs’ spatial reasoning abilities. The second challenge is ensuring that the system remains efficient and lightweight at inference time, avoiding computationally intensive operations that could slow down response generation. Additionally, we propose the Spatial Reasoning Benchmark (SRB), a novel benchmark designed to evaluate models on binary, multiclass, and multilabel classification tasks involving real-world geographic divisions within the United States and their actual topological relationships. We leverage SRB to evaluate *SpaRAGraph*, highlighting its effectiveness in enhancing the spatial

reasoning capabilities of small, open models across these tasks.

Compared to other Q&A frameworks employed on spatial data [60, 61, 62], SpaRAGraph differentiates in three dimensions; the first is that it generates per-question context at inference time for guiding the model’s response using factual information. Second, it leverages a relational RDF graph to efficiently retrieve and combine spatial relations, minimizing additional latency in the overall process. Third, the graph-based spatial context generation is non-stochastic and follows a pre-defined spatial relation composition matrix, enabling both faster and more accurate spatial reasoning than pure, LLM-based approaches.

Contributions We introduce SpaRAGraph, a novel end-to-end framework for spatial reasoning through inference over spatial text, which leverages graph indexing on spatial RDF data. SpaRAGraph comes with the *SpaTex* module, a scalable, specialized tool for spatial-to-RDF data generation that captures spatial relations between neighboring entities in a dataset. SpaRAGraph applies a graph traversal approach assisted by a spatial relation composition matrix on the RDF graph generated by the *SpaTex* module to conduct spatial reasoning and deterministically generate the appropriate context for the LLM. For the evaluation of SpaRAGraph, we introduce SRB, a benchmark that challenges LLMs on binary, multiclass and multilabel classification tasks on real-world, spatial entities. Our evaluation demonstrates that SpaRAGraph significantly boosts model performance across all tasks in the SRB datasets, with impressive improvements on the F1 score in different LLMs (an average improvement of 36 percentage points). All tested models benefit from SpaRAGraph to varying degrees, with minimal impact on response time.

1.4 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapter 2, we provide the necessary background and discuss related work, highlighting any room for improvement and our motivation.

In Chapter 3, we present our raster-based polygon approximation techniques and their application on spatial joins with various topological predicates. Our approaches introduce filters that perform fast and accurate spatial join evaluation over the raster-based approximations, improving the overall query execution time that can be integrated in all modern, real-world spatial databases. All our proposed approaches are

evaluated through a long series of experiments over real-world datasets.

In Chapter 4 we present Hecatoncheir, a prototype distributed spatial data management framework designed for performance and scalability. Hecatoncheir uses state-of-the-art indexing and query processing techniques, optimized for distributed environments, minimizing inter- and intra-process communications and focusing on load balancing, parallel query execution and low memory footprint. We validate Hecatoncheir’s performance and scalability through a wide range of experiments, comparing it to other modern distributed spatial data management frameworks.

In Chapter 5, we propose SpaRAGraph, a RAG technique that utilizes topological relations between entities to enhance the spatial reasoning capabilities of LLMs and facilitate domain-specific response generation to spatial questions. Our approach implements the RAG paradigm through a combination of named entity recognition, graph and vector indexing and rule-based context generation techniques. We experimentally evaluate SpaRAGraph employed over several small, open models.

In Chapter 6 we summarize the contributions of this dissertation and discuss future work.

CHAPTER 2

BACKGROUND & RELATED WORK

2.1 Complex Geometries & Approximations

2.2 Spatial Joins

2.3 Scalable Spatial Data Management

2.4 Spatial Reasoning in LLMs through Retrieval-Augmented Generation

In this chapter, we will present and discuss the related work and necessary background for this dissertation. Specifically, we will first discuss complex geometries and their approximations (Section 2.1), highlighting their key challenges and benefits. Section 2.2 will provide the necessary background and related work on the multi-step evaluation of spatial joins. In Section 2.3 we will discuss scalable solutions to spatial data management problems. Finally, in Section 2.4 we will discuss related work on RAG techniques for spatial reasoning tasks in LLMs.

2.1 Complex Geometries & Approximations

Real-world geographic entities such as user location, lakes, roads etc. are the most common types of spatial data stored in spatial database systems. Entities are as geometry representations, based on the type of geometry that best represents their shape. However, as shapes increase in both complexity and quantity, they may need to be approximated in memory, with their full geometries stored on disk. In such

cases, the spatial database can evaluate queries using these approximations instead of the full geometries, retrieving the actual geometries from disk only when necessary. This eliminates the need to keep all geometries in memory at all times and reduces overall I/O costs, as fewer objects need to be loaded.

However, most polygon approximations such as the MBR, are unable to detect true hits, leaving it still for the refinement phase to do so. These types of polygon approximations are referred to as *conservative approximations*, since they cover a larger area than the original geometry and thus can only be used to detect true negatives. Some of the most common conservative approximations [1] are the following:

- Rotated Minimum Bounding Rectangle (RMBR), which rotates the traditional MBR.
- Minimum Bounding Circle (MBC), that uses the minimum circle instead of the minimum rectangle to approximate a polygon.
- Minimum Bounding Ellipse (MBE), which similar to MBC, uses an ellipse to enclose a polygon.
- m -Corner (mC), which uses m points to enclose the object in a new m -point polygon. Usually $m = 5$ and it is referred to as 5C.
- Convex Hull (CH), a popular approximation that is well-studied by computational geometry.

Figure 2.1 illustrates these conservative approximations.

On the other hand, *progressive approximations* are those capable of detecting true positives. If two progressive approximations intersect, it is assumed that the original geometries of the objects also intersect. However, if the approximations do not intersect, the original geometries must still be evaluated to avoid potential false negatives.

Progressive approximations are more expensive to produce, especially if their optimized (or maximum) case is needed. For example, the Maximum Enclosed Rectangle (MER) and Maximum Enclosed Circle (MEC) [1] can both be used for identifying true positives, but their calculation is significantly more expensive than their conservative counterparts MBR and MBC, respectively. MER and MEC are illustrated in Figure 2.2. They usually detect around 1/3 of the total true hits [1] by themselves, so they need to be combined with a conservative approximation in order to perform

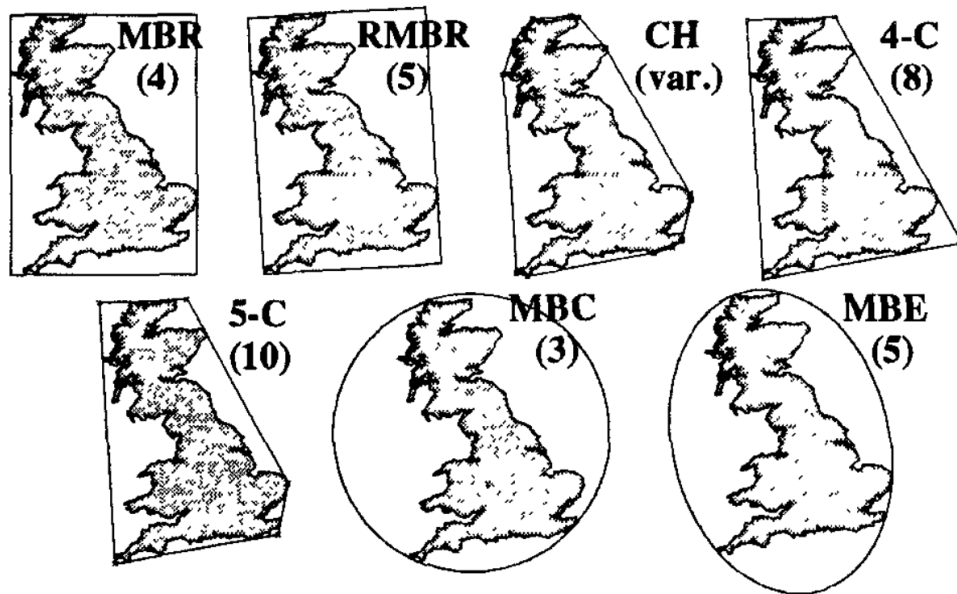


Figure 2.1: Examples of conservative polygon approximations. (Image from [1])

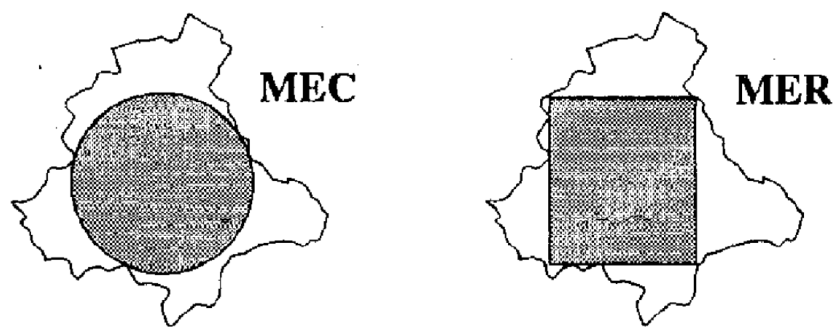


Figure 2.2: Examples of progressive polygon approximations. (Image from [1])

well. This further increases memory requirements and computational cost, raising the question of whether such filters are truly worth the trade-off.

Raster images of polygons [24, 28] have also been studied as an approximation tool in spatial data management. The process of rasterization can be either progressive, conservative or both and transforms a polygonal geometry into a set of raster cells. The raster image of the polygon in Figure 2.3 is based on the Raster Approximation [28]. Since it covers more area than the actual geometry, it is considered a conservative approximation, meaning that it can detect true negatives. However, it is specially constructed to identify some true positives as well. It calculates the percentage of each cell's area covered by the original polygon and then labels it as Full, Strong, Weak or Empty, for 100%, $> 50\%$, $\leq 50\%$ or 0% coverage respectively.

Thus, checking for intersection using the raster approximations of two polygons

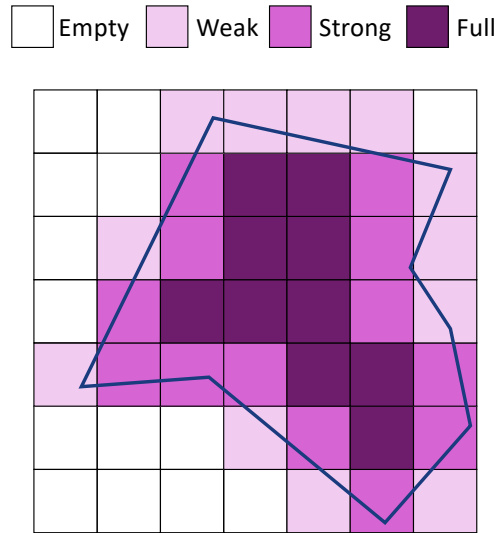


Figure 2.3: Raster image of a polygon.

	empty	weak	strong	full
empty	no	no	no	no
weak	no	<i>inconclusive</i>	<i>inconclusive</i>	yes
strong	no	<i>inconclusive</i>	yes	yes
full	no	yes	yes	yes

Table 2.1: Do two objects intersect in a cell, based on the cell's types in the two raster approximations?

instead of their geometries, enables detecting true positives based on their shared cells and their respective classes. This is seen in Figure 2.4, where the two raster approximations each intersect some common cells. However, in cells 8 and 13, both raster approximations cover more than 50% of the cells' area and are thus classified as Strong. Based on Table 2.1, this guarantees that the geometries intersect each other and is trivial to prove [28]. If two polygons share only cells that fall in the *inconclusive* cases, then they still need to be geometrically refined.

2.2 Spatial Joins

2.2.1 Join Predicates

Spatial joins query two collections of objects based on a join *predicate* that defines a topological relation. As mentioned in Section 1.1.3, DE-9IM [40] is used to define

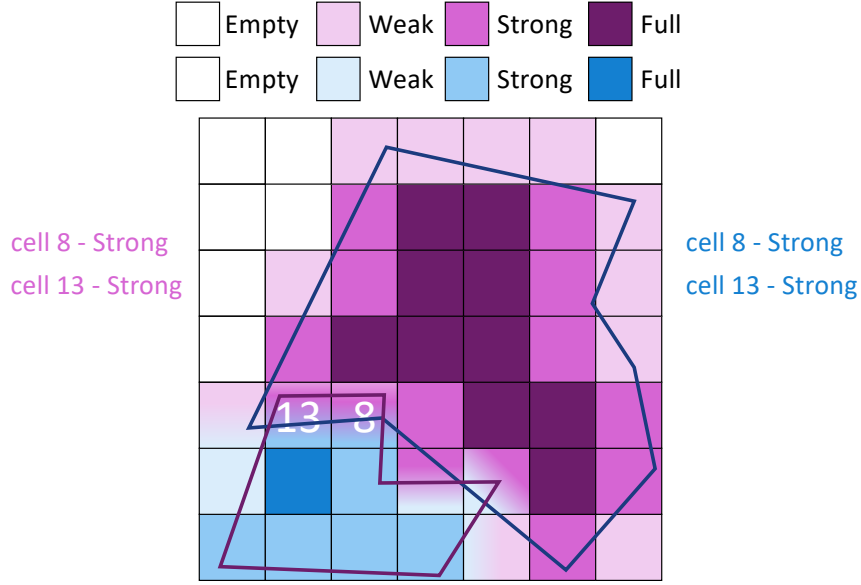


Figure 2.4: An intersection example between 2 raster approximations.

some of the most common topological relations that two objects may share. DE-9IM breaks down each of the input geometries r and s into three parts: the *interior*, the *boundary* and the *exterior*. The intersection between each pair of parts is computed to fill in a 3×3 matrix, where each row corresponds to a part of r and each column to a part of s . Each element of the matrix is a boolean value (T, F) that indicates intersection between the corresponding parts. A DE-9IM matrix can be flattened to a 9-element string code, where the first 3 values are the first row, the next 3 values the second row, etc. For example, the string code for the two objects r and s in the disjoint relation shown in Figure 1.1(a) is FFTFFTTTT. To compute the DE-9IM matrix, plane sweep [64] or overlay of trapezoidal decompositions [48] can be used.

As shown in Table 2.2, each topological relation can be detected by applying one or more DE-9IM *masks*, (i.e., strings) having T, F, or * at each position, where * denotes any of {T, F}. If the DE-9IM string code matches a mask, then the object pair satisfies the corresponding topological relation. Figure 2.5 shows a Venn diagram of the 8 topological relations and their relationships. The generalization hierarchy of relations is reflected by the relationships between their masks. For example, the *covers* masks are all included in the *intersects* masks.

Table 2.2: DE-9IM masks of topological relations.

	DE-9IM mask			
disjoint	FF*FF****			
intersects	T*****	*T*****	***T*****	****T*****
covers	T*****FF*	*T*****FF*	***T**FF*	****T*FF*
covered by	T*F**F***	*TF**F***	**FT*F***	**F*TF***
equals	T*F**FFF*			
contains	T*****FF*			
inside	T*F**F***			
meets	FT*****	F**T*****	F***T*****	

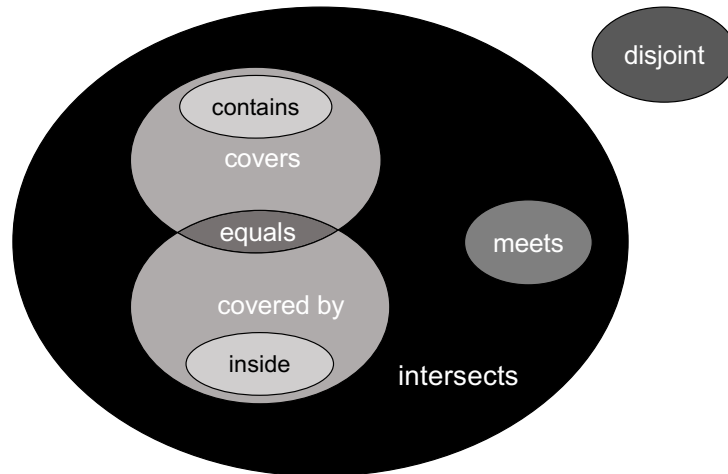


Figure 2.5: Venn diagram of the 8 most common topological relations.

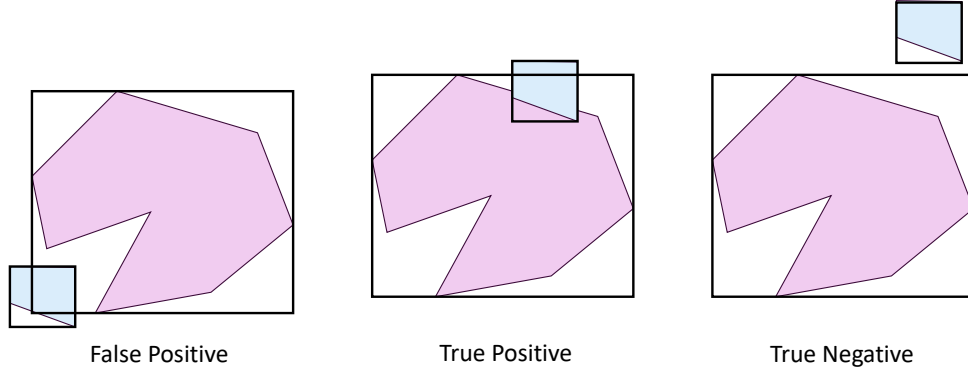


Figure 2.6: Example of MBR intersection tests. The first pair of polygons do not intersect, however their MBRs do.

2.2.2 Multi-step Evaluation

The detection of spatial relations between objects (or the evaluation of spatial joins) is typically conducted in two steps [33]. At the *filter* step [10], the MBRs of the objects are compared in $O(1)$ time, to possibly filter the pair from further consideration. As MBRs are conservative approximations (Section 2.1), they can only be used to detect disjointness. When a massive number of relations need to be detected, as in geo-spatial interlinking [35], the filter step can be evaluated efficiently as a spatial join between MBRs [27]. This problem has been well-studied with scalable solutions to the number of objects for in-memory data [27, 57] and disk-based [26, 49] or distributed data [51, 65]. Still, for the verification of the exact relation between two objects whose MBRs intersect, the current practice is to compute their DE-9IM matrix. In view of the high cost of DE-9IM matrix computations, Papadakis et al. [11] suggested to examine the intersecting MBRs pairs in an order that would maximize the chances of detecting non-disjoint relations. Learning techniques have also been developed to improve the efficiency of link completion in spatial knowledge graphs [66], some through link prediction [67, 68, 69]. These methods are approximate while having a high training cost.

In spatial intersection joins, the check between two MBRs can be performed using only their projections on the x and y axes. If at least one of the objects' projection ends before or starts after the other's in an axis, then the MBRs are disjoint. This can be calculated with 4 simple inequality tests, meaning it has a $O(1)$ time complexity instead of the plane sweep's $O(k \log k)$ for a pair of objects.

Figure 2.6 shows how the MBRs are used to detect intersections. Even though we

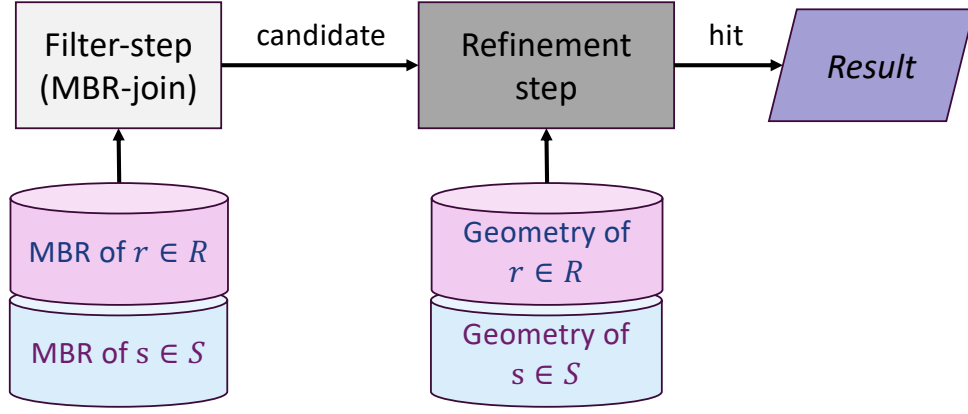


Figure 2.7: Common intersection query pipeline with a Filter and a Refinement stage.

can guarantee that when two MBRs do not intersect (right example, true negative), then their enclosed polygonal geometries also do not intersect, we cannot assume that the reverse stands. This can be seen in the example, where the left pair of MBRs intersect, even though the polygons do not.

Consequently, the pairs that pass the MBR filter are forwarded to the refinement stage, where their original geometries are checked with each other. As mentioned before, this step includes expensive tests (even using the plane sweep algorithm), which considerably increase the pipeline’s total runtime. It has been found that the refinement stage usually takes up to 99% of the total time [70] and most recent work has been focusing on improving the filter step, in order to reduce the amount of objects forwarded to the refinement phase. The spatial query processing *pipeline* is illustrated in Figure 2.7. The pipeline in this example is used to evaluate intersection join queries for two objects r and s of different data sets R and S respectively.

The majority of previous work on spatial joins focuses on the filter step [10]. Divide-and-conquer join evaluation techniques partition the data space explicitly [49, 14, 27] or implicitly [26, 71] with the help of pre-existing spatial indexes [72], 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 [73].

2.2.2.1 Intermediate Filtering

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. [1] suggested the use of the convex hull and

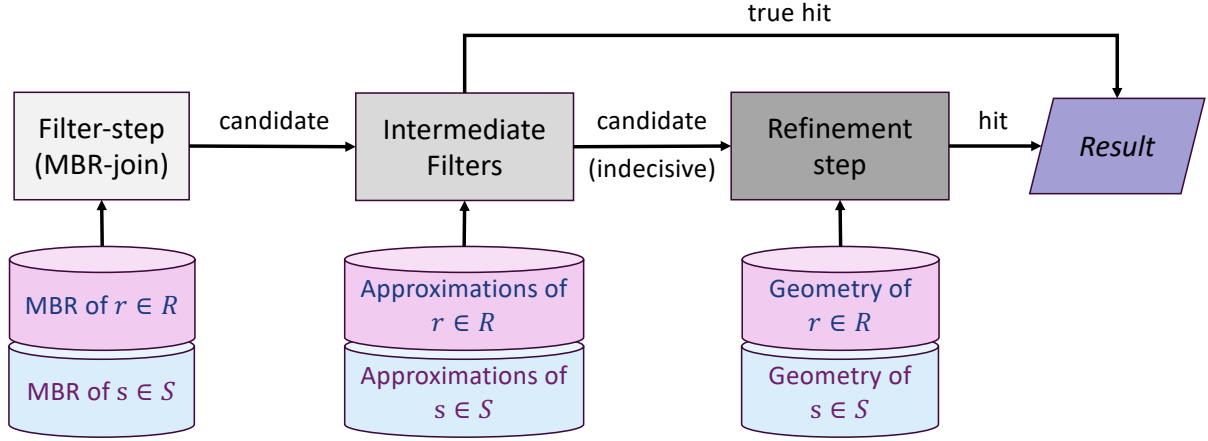


Figure 2.8: Query pipeline with an intermediate step.

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 [28], hence, we did not include it in our comparison. In follow-up work [28], 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 [74] for window and distance queries. In addition, Teng et al. [23] 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.

2.2.2.2 Speeding up the Refinement Step

When querying for a specific spatial predicate, the DE-9IM matrix can be omitted, as it carries additional computational overhead that is not necessary for determining a specific topological relation. For example, an LSI check combined with a PiP test is sufficient to determine whether the areas of two objects intersect (i.e., the general intersection relation), without the need to exhaustively compare all parts of their interiors, boundaries, and exteriors.

Identifying whether two polygons overlap requires point-in-polygon tests and

finding an intersection in the union of line segments that form both polygons [4]. A point-in-polygon test bears a $O(n)$ cost, while the second problem can be solved in $O(n \log n)$ time [75], where n is the total number of edges in both polygons. Given a pair of candidate objects, Aghajarian et al. [76] 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 [77] 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. [4] and alternative polygon decomposition approaches where suggested in [78]. PSCMBR [79] 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 [76] is fully integrated in our implementation; still the refinement cost remains high. Finally, the Clipped Bounding Box (CBB) [22] 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.

2.2.3 Raster-based approaches for other queries

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 [18, 25]. The approximation of spatial objects using space-filling curves (and approximate evaluation of spatial queries) was first suggested by Orenstein [80], however, we are the first to suggest the binary encoding of cells and merging the codes to bitstrings for identifying true hits in spatial intersection joins

(Section 3.1, extended in Section 3.2) and its applicability on topological joins (Section 3.3).

Hierarchical (quad-tree based) raster approximations based on a hierarchical grid have been used in the past [74] for window and distance queries. In addition, Teng et al. [23] propose IDEAL, a hybrid vector-raster polygonal approximation, targeting point-in-polygon queries and point-to-polygon distance queries.

The approximations in IDEAL are similar to those of APRIL (Section 3.2.1) in that they capture information about Full or Partial coverage of each cell, but they also have important differences that render IDEAL approximations not appropriate for spatial intersection joins. Specifically, in IDEAL, each polygon is approximated by its own (local) grid, defined by splitting the object's MBR. Hence, the IDEAL grids of two different objects are not necessarily aligned to each other and may have different resolutions, as shown in Figure 2.9(a). Hence, IDEAL approximations are not appropriate for intersection (or distance) joins because aligning the two different grids of two polygons (having different positions, cell size, and resolution) is hard and inference of polygon intersection from the cell types if the cells are not perfectly aligned is not trivial. Another difference between IDEAL and our work is that cells in IDEAL are not grouped into intervals and interval joins are not used as operations.

RAPTOR [81, 82, 83] joins a raster dataset (map of pixels, where each pixel is associated with values such as temperature) with a vector dataset (e.g., set of polygons, linestrings, or points). The objective of Raptor-Join is to identify, for each vector object o , the pixels that are relevant to o and associate with o the values of these pixels to the object (e.g., aggregate them). For example, if the vector object o is a polygon, the relevant pixels to o are those whose centroids are included in o . Hence, in Figure 2.9(b), the dashed polygon is relevant to the cells in light-gray and the solid-border polygon is relevant to the cells in dark-gray. To compute the Raptor-Join, in a pre-processing step, all relevant cells to each object are identified and stored in a tabular representation (called Flash Index), with intervals of contiguous cells per row of the raster matrix. For example, the solid-border polygon is represented by three tuples: $\{(5, [1, 2]), (6, [1, 2]), (7, [0, 3])\}$, implying that the polygon spans columns 1-2 in rows 5 and 6, and columns 0-3 in row 7. This is reminiscent to our APRIL approximations, where each object is represented by intervals of cells. However, raster representations of vector objects used by RAPTOR have several important differences to APRIL. First, for a cell to be included in a Raptor approximation, the center of

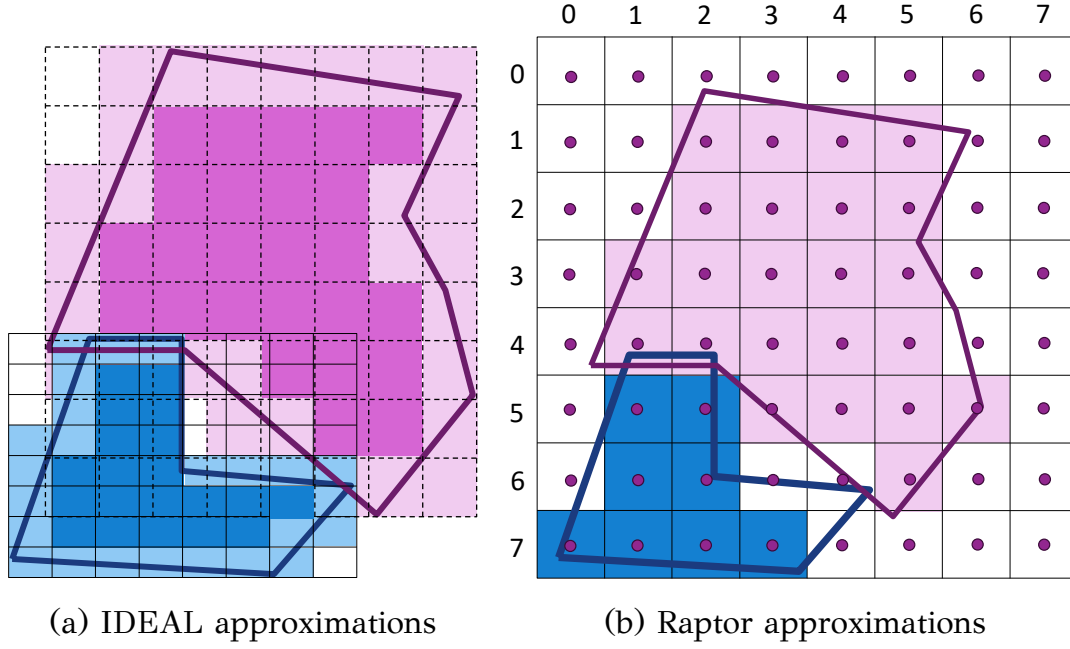


Figure 2.9: Examples of IDEAL and Raptor object approximations.

the cell should lie inside the polygon, whereas in APRIL the cell should overlap with the polygon. This means, for example, that cell (4,1) is not part of the solid-border polygon approximation in Raptor, whereas it is part of its APRIL approximation. Second, APRIL differentiates between Full and Partial cells, whereas Raptor only has one type of cells. The most important difference is that Raptor-join cannot be used for the problem of spatial intersection joins that we study in this thesis, as it is possible that two polygons intersect but their Raptor approximations share no common cell(s). In the example of Figure 2.9(b), the two polygons intersect each other in cells (4,1), (4,2), and (6,4). Cells (4,1), (4,2) are included in the approximation of the dashed-border polygon but not in the other one, whereas cell (6,4) is included in neither of the two Raptor approximations. Hence, Raptor-join, if applied for spatial intersection joins, would mistakenly prune this pair of objects as false positive.

2.3 Scalable Spatial Data Management

With the advent of cloud computing, there have been many efforts in scaling out spatial data management with a series of distributed spatial analytics frameworks and engines being proposed. Additionally, there is a trend in implementing spatial joins for GPUs [76, 84, 77, 79], with a focus on the refinement step. Polygon decomposition

and rasterization techniques for point-polygon joins using GPUs and CPUs have been explored in [24, 18].

2.3.1 Distributed Spatial Analytics Frameworks

The growing demand for efficient management of large-scale spatial data has driven the research community to develop a variety of modern spatial data analytics systems. Each system offers distinct features, making them valuable tools for research and development in both academia and industry. Most of these systems are built on top of large-scale data processing and distributed computing frameworks such as Apache Spark [85] and Hadoop [86], with Spark standing out in terms of performance and therefore being the preferred choice within the community. Regarding spatial joins, all the following systems focus on the filter step only.

Apache Sedona (formerly GeoSpark) [65] uses SJMP, an adaptation of the PBSM spatial join algorithm [49], 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. It supports spatial range queries, k NN queries, and spatial joins, though not k NN joins. Implemented in Java/Scala, it accommodates diverse spatial data types, including points, polygons, rectangles, and linestrings. The system provides several spatial proximity partitioning strategies, such as KDB-Tree, R-Tree, Quad-Tree, and others, with indexing available through either R-Tree or Quad-Tree. According to benchmarks, Sedona currently outperforms other spatial data analytics systems in terms of efficiency [21]. By combining broad query support with rich data type compatibility, it consistently leads query evaluation benchmarks and is regarded as the state-of-the-art system today.

Magellan is another distributed spatial analytics framework built on Spark. It supports a wide range of spatial data types, including points, rectangles, polygons, linestrings, multipoints, and multipolygons. The system enables range queries and spatial joins but lacks support for k NN queries, distance joins, and k NN joins. A distinctive feature is its integration of geometric predicates such as intersects, within, and contains. Data is indexed using a Z-curve, though it can also take advantage of pre-existing indices if available. The Z-curve is employed not only for indexing but also as a spatial partitioning strategy, achieved through inner joins followed by dataset filtering. Compared to Sedona, Magellan operates with a smaller memory footprint.

However, its indexing overhead (particularly for linestring data) can be substantial. While it scales effectively, its query performance generally falls short of Sedona's [21]. Moreover, due to the absence of query optimization mechanisms, Magellan resorts to scanning all partitions across datasets, which further limits its efficiency.

SIMBA (Spatial In-Memory Big Data Analytics) [6] is a distributed spatial analytics engine built on Apache Spark. It extends Spark SQL across the system stack to support expressive spatial queries and analytics through both SQL and DataFrame interfaces. To improve performance, SIMBA introduces native indexing over RDDs for efficient spatial operators and enhances Spark SQL's query optimizer with spatial-aware, cost-based optimizations that leverage available indexes and statistics. For data partitioning, SIMBA constructs an R-tree by sampling the input dataset and applying the STR algorithm [87] to form the first tree level, which defines partition boundaries. Users also have the flexibility to define custom partitioning schemes. By default, SIMBA employs R-tree indexing. It supports range queries (both rectangular and circular), k NN queries (for points), distance joins (points), and k NN joins (points), though it does not support spatial joins. A key strength of SIMBA is its ability to optimize indexes for parallel query execution, significantly improving analytical throughput. In terms of indexing efficiency, SIMBA is among the strongest frameworks for point datasets. However, when it comes to range query evaluation, Sedona continues to outperform it.

SpatialSpark [88] is another distributed spatial data processing framework built on Apache Spark, designed to support large-scale geospatial analytics. It implements core spatial operations such as range queries, k NN queries, and spatial joins, and relies on spatial partitioning techniques to minimize data shuffling across the cluster. By integrating directly with Spark's RDD abstraction, it offers a lightweight and scalable solution for spatial workloads. However, SpatialSpark does not provide advanced indexing structures or cost-based optimizations, which limits its efficiency compared to more recent systems like Sedona and Simba. Although competitive in terms of performance, it requires significantly more memory and is no longer actively maintained, which has ultimately rendered it an outdated framework [21].

Hadoop-based implementations (Hadoop-GIS [89], SpatialHadoop [51]) employ advanced partitioning, indexing, and spatial query processing techniques. However, they have been shown to underperform compared to Spark-based frameworks, primarily due to the in-memory processing advantages that Spark provides.

2.3.2 MPI

The Message Passing Interface (MPI) [8] is a standardized framework for message-passing, designed to operate across a wide range of parallel and distributed computing architectures. It enables communication between processes running on the same or different machines and supports multiple high-speed network technologies. MPI provides both point-to-point and collective communication mechanisms, making it the de facto standard for inter-process communication among parallel programmers and developers of shared-memory systems. Two leading MPI implementations, along with their respective compilers, dominate the field: OpenMPI [90] and MPICH [9]. Both projects are maintained by international collaborations that include contributors from academic institutions as well as industry.

MPI has been employed for parallelizing databases [91, 92], but it remains primarily popular within the High-Performance Computing (HPC) community. In data-intensive workloads and modern data management scenarios, however, MPI is increasingly viewed as unsuitable. Compared to higher-level abstractions like Spark, MPI is less flexible for several reasons. First, it relies on communication groups, which are costly to create and manage in dynamic environments such as the cloud, and its implementation is complex. Second, collective operations like Broadcast or Gather require synchronous method calls, which can hinder parallelization. Third, MPI offers limited elasticity, availability, and fault tolerance, making it poorly suited for service-oriented applications such as cloud-based database systems [93].

In tightly-coupled, cluster-based environments, MPI is a highly effective tool for improving database performance. Its low-latency communication and fine-grained control over process interactions allow parallel workloads to be executed efficiently, making it well-suited for scenarios where performance and precise coordination are critical. By enabling direct, high-speed message passing between nodes, MPI can significantly accelerate computationally intensive database operations, such as complex joins or large-scale query processing, in environments where the overhead of higher-level frameworks may be prohibitive.

2.3.3 Spatial joins on GPUs

The widespread availability of programmable GPUs has inspired several research efforts that leverage GPUs for spatial joins [94, 76, 84, 77, 79]. Sun et al. [94] accelerated

the join refinement step by incorporating GPU rasterization as an intermediate filter. This filter identifies *only true negatives* using a low resolution, and has thus limited pruning effectiveness. Aghajarian et al. [76, 84] proposed a GPU approach to process point-polygon and polygon-polygon joins for datasets that can be accommodated in GPU memory. Liu et al. [77, 79] also proposed GPU-accelerated filters to reduce the number of refinements. These filters [76, 84, 77, 79], in contrast to APRIL (Section 3.2.1), *do not identify true hits*, but rather focus on finding the intersection points between a candidate pair. Furthermore, the above approaches [76, 84, 77, 79] do not involve rasterization and rely on CUDA, which is exclusive to NVIDIA GPUs. A recent line of work [24, 16, 25, 17] proposes to use the GPU rasterization pipeline as an integral component of spatial query processing. Doraiswamy et al. [16, 17] introduced a spatial data model and algebra that is designed to exploit modern GPUs. Their approach leverages a data representation called *canvas*, which stores polygons as collections of pixels. The canvas includes a flag that differentiates between pixels that lie on the boundary of the polygon and those that are entirely covered by it. Although current-generation GPUs can handle millions of polygons at fast frame rates, the evaluation of spatial queries is still dominated by other costs, such as triangulating polygons and performing I/Os [17].

2.4 Spatial Reasoning in LLMs through Retrieval-Augmented Generation

2.4.1 Retrieval-Augmented Generation

Concerning how well an LLM exploits information beyond its pre-trained knowledge base, several RAG benchmarks exist to serve for evaluation. Most of them study the efficiency of the retrieval and the response generation by means of question-answering instances. Specifically, the main aspects studied are: context relevance, i.e., how pertinent the retrieved context to the query is; context utilization, i.e., the extent of the context that is used by the generator to produce the response; error handling, i.e., the ability to handle errors that exist in documents; and completeness, i.e., how well the response incorporates all the relevant information in the context. RGB [95] focuses on data that pertain to news while RAGBench [96] covers different domains. CRAG [97] is a comprehensive factual question-answering benchmark

that aims to define types of questions from different domains given their diverse and dynamic nature. BERGEN [98] emphasizes on the LLM-based semantic evaluation of answers, highlighting the importance of using efficient retrievers as they can affect the RAG response generation. MIRAGE [99] measures the accuracy of the predicted correct answer choices on multi-choice questions for the medical domain. Similarly, LegalBench-RAG [100] emphasizes in the legal domain measuring the effectiveness of the retrieval phase and the legal reasoning capacity of LLMs. UDA [101] focuses on the RAG assessment on lengthy and highly unstructured external data such as those found in PDFs and HTML tables. MultiHop-RAG [102] assesses multi-hop queries, i.e. queries that require retrieving information from multiple documents to reason and arrive at an answer. It evaluates the quality of the retrieved set for the query and the reasoning capability of the LLM.

Graphs have gained significant attention in RAG research [103, 104, 105], being applied to a variety of tasks such as summarization, question answering, and knowledge graph reasoning. However, to the best of our knowledge, no prior work has leveraged graph-based indexing and retrieval for spatial data, nor has it systematically evaluated graph-enhanced (or just plain) RAG in the context of spatial reasoning, an emerging and distinct research challenge for LLMs.

2.4.2 Spatial Reasoning and GeoAI

LLMs have demonstrated strong reasoning capabilities through chain-of-thought (CoT) prompting [106, 107]. This raises the possibility that LLMs could infer the spatial relation between two entities (even when it is not explicitly stored) by reasoning over known spatial relations involving intermediate entities. Li et al. [108] follow this approach, introducing the “Advancing Spatial Reasoning” (ASR) method, which enhances spatial reasoning on large GPT models using CoT and Tree-of-Thoughts (ToT) prompting on the StepGame benchmark [109]. Both ASR and the StepGame benchmark investigate and test directional relations (similar to cardinal directions but not for geographic entities) in a multi-hop reasoning setting that challenges the model’s capacity for spatial inference. Additionally, the StepGame benchmark emphasizes understanding various phrasings of positional directions (such as “6 o’clock” or “to the left”) which are inherently egocentric and context-dependent. While effective for evaluating spatial reasoning in local or embodied environments, these expressions

are not compatible with geospatial data, which relies on an absolute, coordinate-based reference system (e.g., latitude and longitude) tied to the Earth's surface. As such, StepGame is ill-suited for assessing models' understanding of global spatial relations like cardinal directions and topological relationships.

A more theoretical but preliminary analysis of various spatial relations between entities and the limitations of LLMs in comprehending them, is presented in [110]. However, this work does not examine geographic spatial relations, such as cardinal directions. Related work that leverages external spatial information to assist LLMs includes GeoLLM [60], GeoLLM-Engine [61] and GeoGPT [62]. GeoLLM focuses on regression tasks such as the prediction of population density; it uses auxiliary map data from OpenStreetMap from which the nearby locations of the given (query) location are fetched and passed to the LLM as a fine-tuned prompt. GeoLLM-Engine is an environment of tool agents for earth observation applications. It capitalizes a LLM in order to convert natural language instructions into a set of tasks over satellite images. For this, it performs function calls to geospatial APIs, dynamic maps/UIs and external multimodal knowledge bases. GeoGPT employs an LLM for interpreting the users' demands from the input and calls an external GIS tool from a pool of available ones to solve the task. Some of these tools serve processes that pertain to data collection, data loading and data analysis.

Another line of research fine-tunes an LLM to enhance its understanding of spatial context. MaaSDB [111] is a vision paper that envisions a spatial database system for enhanced user accessibility by training LLMs on data retained in a spatial database. In this way, the machine learning models can be utilized as a spatial database, enabling a new generation-based query paradigm that replaces the traditional retrieval-based one. LLM-Geo [112] is a prototype that operates as an autonomous GIS that can produce and execute Python code for spatial data loading and visualization. By exploiting the capabilities of the LLM natural language understanding, reasoning and code generation, it manages to generate at first a step-by-step workflow that is formed as a directed acyclic graph given users' data and spatial question. The graph consists of a series of connected operations and nodes. The LLM is reused, as the graph is passed to it in order to generate code in each operation node. Then, the generated code is collected and submitted to the LLM along with the graph and the users' input to create the final program. The program is executed producing the results that can be static maps, charts, new datasets, etc.

GS-SQL [113] is a graph-based text-to-SQL model that defines an abstract syntax for text-to-query translation, focusing on accurately transforming spatial queries from natural language into SQL. This approach relies on an existing external database with a known schema to generate precise SQL queries, which are then used to query the database either during inference or when required by the user.

2.4.3 Motivation

In contrast to our non-stochastic, rule-based approach, techniques such as CoT and ToT often introduce errors and are significantly slower at inference time, whilst SpaRAGraph is built upon the RAG paradigm, automatically retrieving and generating relevant context to support and enhance spatial reasoning through factual data.

Table 2.3 provides an overview of related work and their respective features when compared with our proposed framework, SpaRAGraph. Notably, none of the existing approaches employ RAG to dynamically generate context during response generation. Moreover, nearly all related work is evaluated on large, closed-source models (e.g., GPT models), overlooking smaller, open-source models that are more accessible to typical users due to their lower cost and ability to run on commodity GPUs. Furthermore, with the exception of ASR, all methods rely on post-processing spatial data after inference. This necessitates the use of external toolchains that must be both highly accurate and capable of fast spatial computation. ASR differs in that it utilizes CoT reasoning to infer spatial relations between entities at inference time, based on the StepGame benchmark. We elaborate on the specific distinctions between these approaches and SpaRAGraph below:

GeoLLM is a fine-tuning and prompt engineering approach tailored for geospatial tasks, leveraging contextual information extracted from OpenStreetMap [114] along with external toolchains. In contrast to SpaRAGraph, GeoLLM relies on including spatial data in the prompt, assuming that the model can interpret spatial formats, although such representations are not native to language models. Consequently, GeoLLM does not operate solely on text at inference time, unlike SpaRAGraph.

GeoLLM-Engine is a geospatial copilot application, allowing live interaction with the user through natural language. Its end-to-end pipeline translates and executes a user’s queries using geospatial API tools, maps and multimodal databases. Unlike

SpaRAGraph, the generation/LLM component of GeoLLM-Engine is used mostly to translate and breakdown the user’s query into a series of operations, rather than use it directly to generate a response.

GeoGPT employs CoT reasoning to decompose user queries, subsequently executing them by selecting appropriate tools from a pool of external GIS services that operate on spatial data stored in a dedicated spatial database. In contrast to SpaRAGraph, the language model in GeoGPT functions primarily as a translator between the user and the database, rather than playing an active role in generating the final response. **LLM-Geo** generates and executes geospatial analysis workflows by translating user queries into a sequence of operations and generating Python code to perform these tasks after inference. Unlike SpaRAGraph, LLM-Geo uses the LLM primarily as a scheduler and planner to identify and organize the necessary operations, rather than relying on it as the main component to directly generate the final response.

ASR is similar to SpaRAGraph in that both rely on the model as a central component for spatial reasoning and response inference. However, ASR performs each reasoning step internally within the LLM, whereas SpaRAGraph employs non-stochastic transitional algebra to accurately combine reasoning steps into a coherent overall summary. Additionally, ASR does not utilize dynamic retrieval or augmentation, focusing exclusively on the StepGame benchmark, which provides explicit context per question using synthetic data.

GS-SQL focuses exclusively on improving text-to-SQL translation accuracy for geospatial queries, a task that is fundamentally different from what SpaRAGraph accomplishes.

StepGame operates with per-story, randomly generated entities that exist solely within a localized, relative scope (without persistence across any broader topology). Our proposed Spatial Reasoning Benchmark (SRB) fundamentally differs from StepGame, as it focuses on real-world entities embedded in a shared global data space. This design allows SRB to be applied in specialized GIS scenarios that evaluate not only a model’s spatial reasoning capabilities but also its integration with RAG mechanisms. Leveraging our scalable topology detection techniques, SRB can be adapted to additional datasets, enabling the assessment of models on domain-specific knowledge for more specialized GIS applications.

Furthermore, StepGame evaluates model responses using multilabel classification, whereas SRB assesses models through binary, multiclass, and multilabel classification

Table 2.3: An overview of related work on Spatial Reasoning and Geospatial LLM approaches and their features.

	Text-to-Query	CoT	Prompt Engineering	Data Post-Processing	RAG
GeoLLM			✓	✓	
GeoLLM-Engine	✓			✓	
GeoGPT	✓	✓		✓	
LLM-Geo	✓			✓	
ASR		✓			
GS-SQL	✓			✓	
SpaRAGraph					✓

tasks. Additionally, StepGame relies on a variety of relative directional relations that are not applicable in geospatial contexts, whereas SRB employs cardinal directions and topological relations, which are standard in geospatial reasoning. Inspired by StepGame, in Chapter 5 we propose and utilize SRB to evaluate SpaRAGraph, as it better aligns with approaches aiming to enhance geospatial reasoning capabilities.

CHAPTER 3

EFFICIENT IN-MEMORY SPATIAL JOINS WITH COMPLEX GEOMETRIES

3.1 An Approximation Technique for Polygon Intersection Joins

3.2 Advancing Raster Interval Approximations for Spatial Joins

3.3 Scalable Spatial Topology Joins

3.4 Conclusions

Complex geometries encapsulate all non-point shapes such as linestrings and polygons with varying degree of vertices and regional characteristics. In real-world datasets, objects often have curvilinear shapes (e.g. water bodies or national parks), that are discretized during data curation and ultimately represented as polygons. This discretization process can result in polygons with hundreds, thousands, or even hundreds of thousands of vertices, creating highly complex geometries that closely approximate the original curved shapes. Hence, it is prohibitive to operate directly on such geometries during spatial join processing, unless it is absolutely necessary.

Multi-step spatial join processing [1] is the standard approach, as it leverages shape approximations before resorting to exact geometry calculations. However, as mentioned in Section 2.1, most spatial approximations cannot efficiently detect both result and non-result pairs. Thus, multiple approximations often need to be stored in memory for each object, along with a series of pipelined filters, ultimately increasing the overall cost of spatial join evaluation in terms of both time and memory.

In light of this, Section 3.1 introduces a novel polygon approximation technique that enables efficient identification of both result and non-result pairs in spatial intersection joins using a single intermediate filter. This approach significantly reduces the number of candidate pairs requiring geometric refinement, thereby improving the overall efficiency of spatial intersection join evaluation. Moreover, in Section 3.2, we extend our approach to be more flexible, more efficient to construct and store, and capable of supporting linestrings and multi-geometries, while maintaining high performance in spatial intersection joins. Finally, in Section 3.3, we explore how our method can be applied to spatial joins with arbitrary predicates that capture more specific topological relationships, such as adjacency and containment.

Outline In Section 3.1 we introduce Raster Intervals, an approximation technique for polygons that can be used in spatial joins to filter out non-candidate pairs and detect candidate pairs early, before resorting to accessing the original geometries. In Section 3.2, we enhance Raster Intervals (APRIL) to facilitate faster construction and filtering times, more flexible handling, smaller memory footprint and overall better performance. In Section 3.3, we discuss how can raster-based approximations be used to detect topological relations between polygons. We introduce a novel technique that uses our proposed APRIL approximations that surpasses all other alternatives. Finally, Section 3.4 summarizes our conclusions.

3.1 An Approximation Technique for Polygon Intersection Joins

3.1.1 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 [28]. 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.

3.1.1.1 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 the set of 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 [28], we do not include empty cells in C_o . To compute C_o for each object and the type of each cell, we apply the algorithm of [28]. In a nutshell, the algorithm first identifies the grid columns (stripes) that 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 3.1. 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 3.1.1.3, with insignificant cost.

Table 3.1: 3-bit type codes for each input dataset

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

3.1.1.2 Intervalization

We use the Hilbert curve [115] 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 3.1 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 $O(|C_o| \log |C_o|)$.

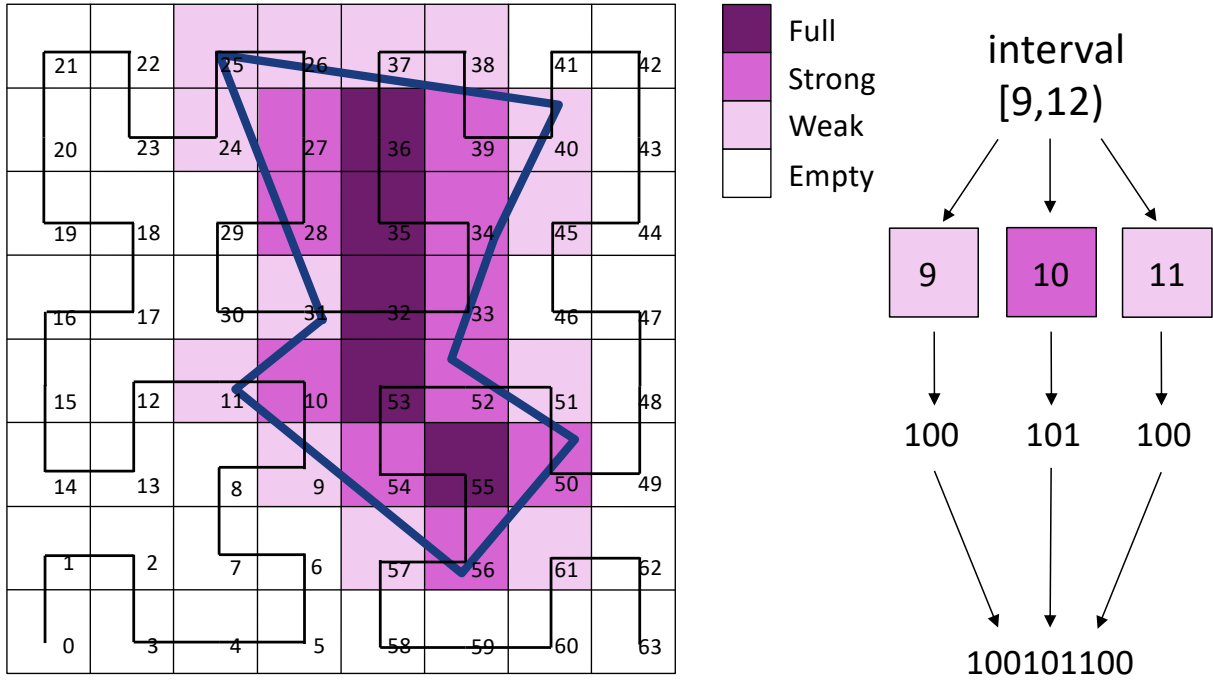


Figure 3.1: The Hilbert curve cell enumeration and interval generation for a polygon in a 8×8 space.

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 3.1 belongs to the left join input R . We replace cells 9, 10 and 11 in C_o with codes 100, 101 and 100, respectively, by interval $[9, 12)$ with binary code 100101100, 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 a bitstring that encodes 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 $\lceil (end - st) * 3/8 \rceil$ bytes to encode its cells. We may opt to compress binary codes consisting of many bytes and the RI approximation of an object, overall.

3.1.1.3 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 3.1).

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, it is possible to determine whether (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 $[\max\{X_i.st, Y_j.st\}, \min\{X_i.end, Y_j.end\}]$, i.e., the

Algorithm 3.1 RI-join algorithm

Require: $RI(r)$ as X , $RI(s)$ as Y

```
1:  $ovl \leftarrow False$ ; ▷ no overlapping interval pair found yet
2:  $i \leftarrow 0$ ;  $j \leftarrow 0$ 
3: while  $i < |X|$  and  $j < |Y|$  do
4:   if  $X_i$  overlaps with  $Y_j$  then
5:     if  $ALIGNEDAND(X_i.code, Y_j.code)$  then
6:       return true hit ▷ bitwise AND is non-zero
7:     end if
8:      $ovl \leftarrow True$ ; ▷ found an overlapping interval pair
9:   end if
10:  if  $X_i.end \leq Y_j.end$  then  $i \leftarrow i + 1$  else  $j \leftarrow j + 1$ 
11: end while
12: if  $ovl$  then ▷ at least one overlapping interval pair
13:   return indecisive
14: else
15:   return false hit ▷ no common cells in  $X$  and  $Y$ 
16: end if
```

intersection interval of X_i and Y_j . Then, we bitwise AND the fragments. If the fragments have the same encoding (i.e., both have R or S encoding as shown in Table 3.1), 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 3.1 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. In contrast, 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 3.2 shows two rasterized polygons and the pairs (X_i, Y_j) of intervals from the two raster intervals that overlap.

In general, the codes (bitstrings) 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. Additionally, bit-shifting is done at the bytes of one interval only (the one that starts earlier), ensuring that the necessary bits are carried over from the subsequent byte to

prevent 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, as 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 3.3 shows how the codes for the first pair (X_0, Y_1) of intersecting intervals from the example of Figure 3.2 are matched, where $X_0 = \langle [9, 13), 100101101101 \rangle$ and $Y_1 = \langle [11, 15), 100100101100 \rangle$ (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 3.1 continues to find the next pair of overlapping intervals (X_5, Y_2) and performs the corresponding code matching.

Analysis 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 3.1 is $O(|X| + |Y|)$ since the number of overlapping interval pairs is at most $|X| + |Y|$.

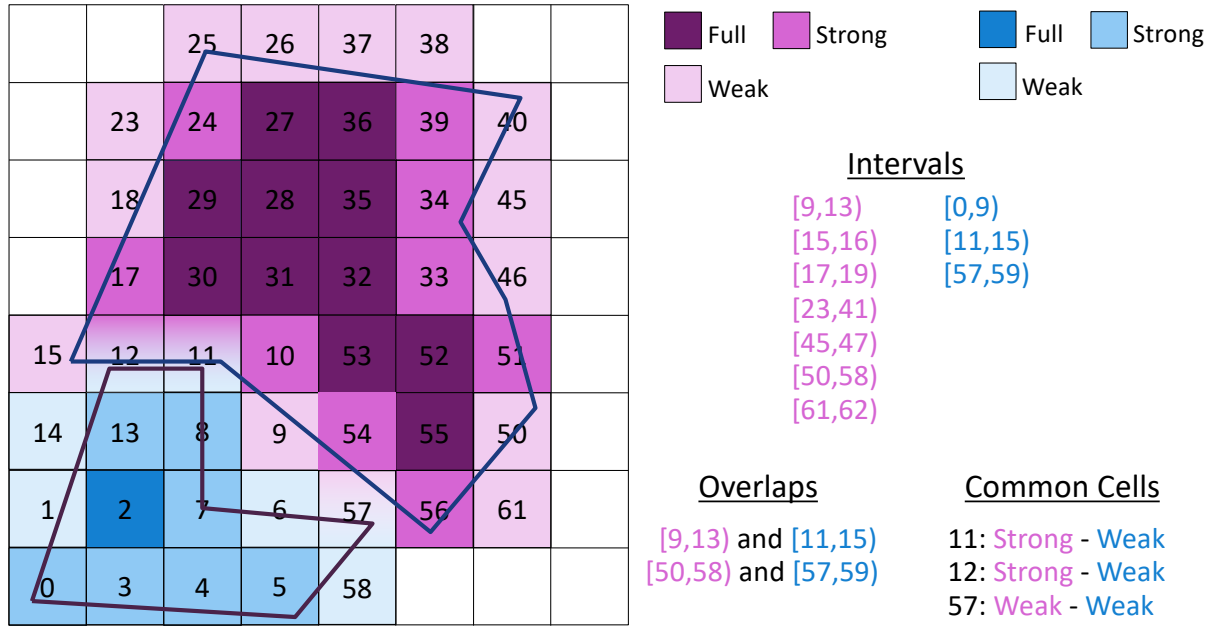


Figure 3.2: Two rasterized polygons, the overlaps between their raster intervals, and their common cells.

3.1.1.4 “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 3.1 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 $Y_j \in RI(s)$, 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 be 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.

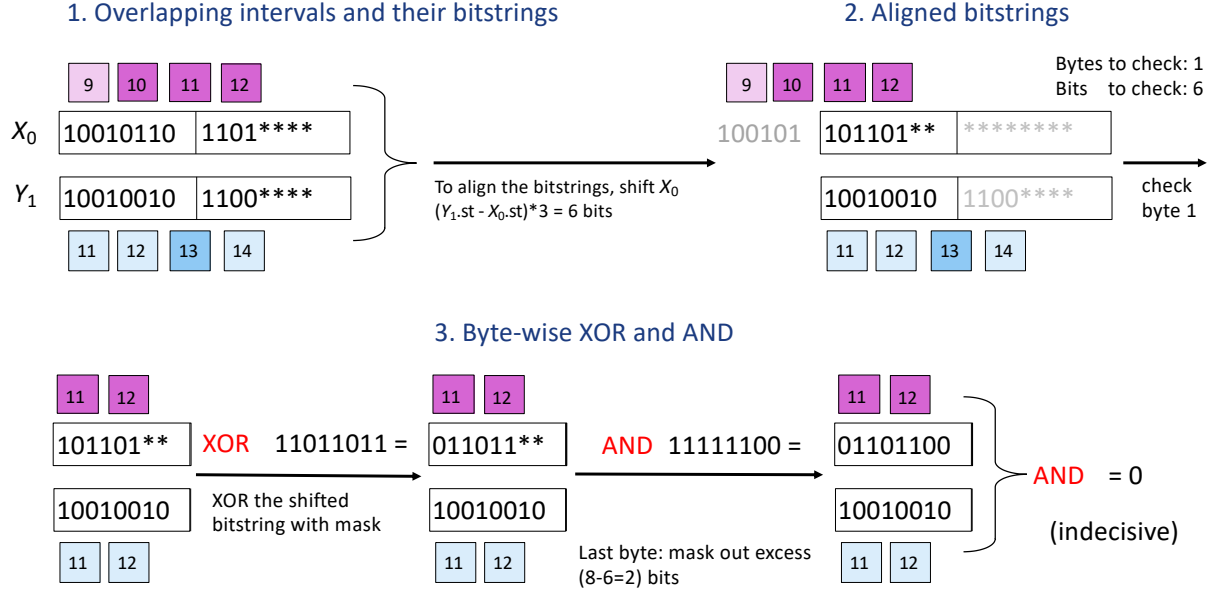


Figure 3.3: Intervals $[9, 13)$ and $[11, 15)$ of our two example polygons overlap but are not aligned. Byte truncation and bit shifting (if necessary) align their bitstrings before performing the bitwise operation(s).

3.1.2 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, in a single-machine setting without thread parallelism. The competitors include the 5-corner approximations comparison followed by a comparison of convex hulls (5C-CH) (i.e., the approach of [1]) and the raster approximation (RA) approach of [28]. 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. [27]. The refinement step was implemented using the approach of [76], 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.

Table 3.2: 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	107K	223K	1.97M	7.18M	448K	623K	72K	192K
Avg # of vertices	32.17	31.92	37.73	47.68	48.90	41.81	49.36	42.74	35.14	32.24	46.14	42.75	59.99	36.53
Avg $\frac{\text{obj MBR}}{\text{world MBR}}$	2.5E-07	2.8E-08	3.4E-08	1.3E-07	3.9E-07	6.7E-07	3.0E-08	6.0E-08	6.2E-08	5.9E-08	1.6E-07	1.3E-07	4.1E-07	2.5E-07
Geometries (MB)	65.67	1.17K	2.5K	771.50	163.90	208.30	84.20	151.30	1.1K	3.8K	334.30	431.30	70.20	113.60
MBRs (MB)	4.52	81.08	144.84	36.03	4.46	8.22	3.90	8.10	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	14.20	39.30	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	898.7	2.0K	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	28.8	57.70	515.40	1.7K	117.80	159.40	18.50	46.60

Table 3.3: Preprocessing costs (in sec) of intermediate filter approximations

	T1	T2	O5NA	O6NA	O5SA	O6SA	O5OC	O6OC	O5EU	O6EU	O5AS	O6AS	O5AF	O6AF
RI	42.109	93.161	243.30	230.630	43.472	201.617	123.960	373.544	164.22	761.78	73.96	168.83	29.56	70.27
RA	38.211	623.256	269.25*	67.455*	38.552	68.257	76.456	121.875	116.66*	408.76*	127.21	195.99	21.71	55.55
5C-CH	1.228	30.780	52.69	11.799	1.619	2.226	2.748	7.678	22.08	65.12	5.54	7.25	1.04	1.89

3.1.2.1 Datasets

We used datasets from SpatialHadoop’s [51] 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., $T1 \bowtie T2$, $O5NA \bowtie O6NA$, $O5SA \bowtie O6SA$, etc.). Table 3.2 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 3.2).

3.1.2.2 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 3.1.1.2, 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 3.4 analyzes the performance of RI for different values of N in spatial join $T1 \bowtie T2$. 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.

In Table 3.5, 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 [28]).

Table 3.4: Effect of N on the performance of RI in $T1 \bowtie T2$

	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 3.5: 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

3.1.2.3 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 [26]), 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 3.3 shows the costs for precomputing the three intermediate filter approximations for all real datasets. The corresponding space requirements are shown in Table 3.2 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 [28], 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 asterisc (*). 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 3.2, 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 3.6 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 3.2.

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

Table 3.6: 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	7.90	17.82
O6OC	7.61	34.73
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

Z-order [116]. Indicatively, Table 3.8 shows these statistics for some of our datasets (see Tables 3.5, 3.6, and 3.2, 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.

3.1.2.4 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 3.7 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

Table 3.7: Performance of intermediate filters in spatial intersection joins

	True hits	False hits	Indecisive	Avg #nodes R	Avg #nodes S	MBR-join (s)	Int. Filter (s)	Refine (s)	Total (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	17.06%	58.56%	24.37%	581.59	1155.79	0.04	0.44	1.62	2.09
RA ($K = 750$)	17.43%	49.51%	33.05%	1195.86	3349.01	0.05	2.13	9.08	11.26
5C-CH	0.00%	41.97%	58.03%	1088.95	2925.64	0.05	0.09	10.27	10.42
None	0.00%	0.00%	100.00%	1395.56	2254.00	0.04	-	13.27	13.32
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 3.8: 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

dataset (R or S) which are indecisive. The last four columns show the total cost of the filter step (MBR-join [27]), intermediate filter step (RI, RA, or 5C-CH), and refinement step [76] 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 3.1.2.3.

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 $O5OC \bowtie O6OC$, 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 $O5OC \bowtie O6OC$). 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 $T1 \bowtie T2$) to 59.4% (in $O5OC \bowtie O6OC$) 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 $T1 \bowtie T2$, 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 3.7). 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 3.2). 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 3.2) makes them a very attractive approximation approach.

Spatial within joins Table 3.9 shows the performance of all intermediate filters for spatial *within* joins. Section 3.1.1.4 explains how the 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 of 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 3.9: 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

3.2 Advancing Raster Interval Approximations for Spatial Joins

3.2.1 APRIL

We now propose *APRIL* (Approximating Polygons as Raster Interval Lists), a significant enhancement of Raster Intervals (RI), which can be used as an intermediate filtering method for spatial query processing and is more efficient and less space consuming compared to RI.

3.2.1.1 A- and F-Interval Lists

APRIL is a succinct polygon approximation for intermediate filtering, which categorizes raster cells into *Full*, *Partial*, and *Empty*, based on their coverage percentage with the object’s geometry (100%, less than 100%, and 0%, respectively). In other words, APRIL *unifies* the *Strong* and *Weak* cell classes used by RI and [28] to a single *Partial* class. Under this, APRIL approximates a polygon with two sorted interval lists: the *A*-list and the *F*-list. The *A*-list contains intervals that concisely capture all cells that overlap with the polygon, regardless of their type (Full or Partial), whereas the *F*-list includes only Full cells. An interval list having n intervals is stored as a simple sorted integer sequence in which the i -th interval’s *start*, *end* are located at positions $2i$ and $2i + 1$ respectively, for $i \in [0, n)$.

The *A*-list and *F*-list for the example polygon of Figure 3.1 are shown in Figure

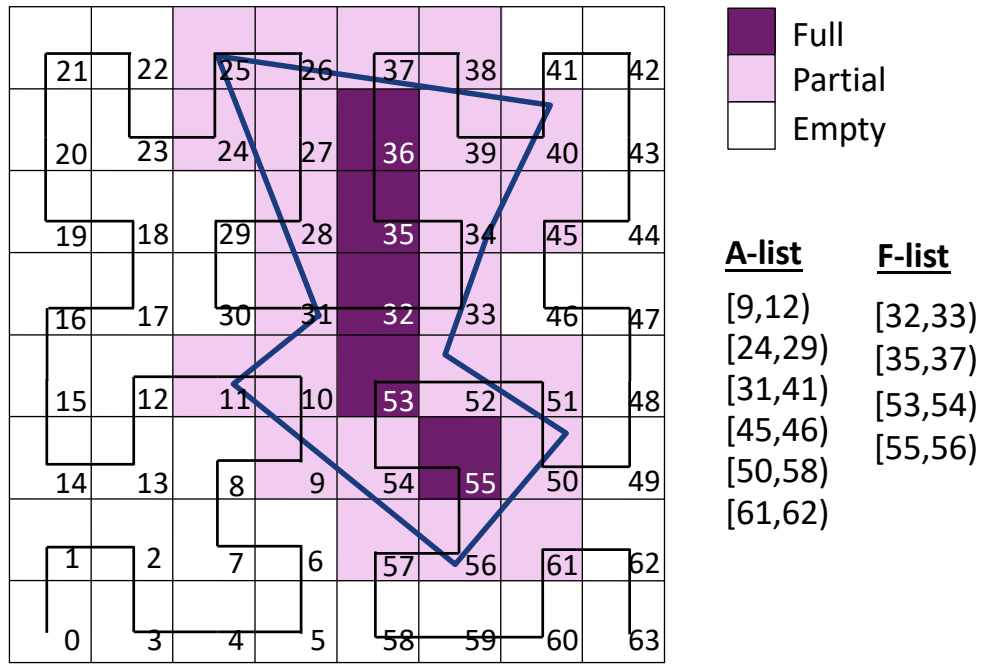


Figure 3.4: The interval generation for a polygon in a 8×8 space, without bit-coding and using interval lists.

Table 3.10: APRIL: Do two objects intersect in a common cell?

	Partial	Full
Partial	<i>Inconclusive</i>	yes
Full	yes	yes

3.4. Strong and Weak cell types become Partial, which results in a simpler representation than RI. Note that the set of intervals in each of the *A*- and *F*- lists are disjoint. The new relationship identification table for a cell shared by two polygons is shown in Table 3.10. Removing the Strong cell type renders the approximation unable to detect true hits for cells of the Strong-Strong case, as common cells that are both Partial cannot decide definite intersection between the two polygons.¹

Construction To construct an APRIL approximation, we first need to identify the cells intersected by the polygon's area in the grid and label each one as either Partial or Full. Then, *Intervalization* derives the *F*-list by sorting the set of Full cells by ID (i.e.,

¹As we have found experimentally (Sec. 3.2.4.4.2), this has minimal effect on the amount of true hits and true negatives that the intermediate filter manages to detect. This is due to the fact that the only cases of true hits missed are pairs of polygons that intersect with each other *exclusively* in cells typed Strong for both polygons and nowhere else.

Hilbert order) and merging consecutive cell IDs into intervals. To derive the A -list, we repeat this for the union of Full and Partial cells. In Section 3.2.3.2, we propose an efficient algorithm that derives the F - and A -list of a polygon without having to label each individual cell that intersects it.

3.2.1.2 APRIL Intermediate Spatial Join Filter

APRIL is used as an intermediate filter (Figure 2.8) between the MBR filter and the refinement phase. Given a pair (r, s) of objects coming as a result of an MBR-join algorithm [26, 49, 27], APRIL uses the A - and F -lists of r and s to detect fast whether the polygons (i) are disjoint (true negative), (ii) are guaranteed to intersect (true hit), or (iii) are inconclusive, and thus need to be forwarded to the refinement stage to verify their intersection.

Whether r and s are disjoint (i.e., do not intersect) can be determined by checking whether their A -lists have any pair of overlapping intervals. If they have no overlapping intervals, then r and s do not have any common cell in the grid and thus they cannot intersect. We check this condition by performing a merge join over the A -lists and stopping as soon as we detect two overlapping intervals.

Pairs of polygons that have at least one pair of overlapping intervals in their A -lists are then checked using their F -lists. We perform two more merge-joins: $A(r) \bowtie F(s)$ and $F(r) \bowtie A(s)$; detecting an overlapping intervals pair in one of these two joins means that there is a Full cell in one object that is common to a Full or Partial cell of the other object. This guarantees that the two objects intersect and the pair (r, s) is immediately reported as a spatial join result. If $A(r) \bowtie F(s)$ fails to detect (r, s) as a true hit, then $F(r) \bowtie A(s)$ is conducted; if the latter also fails, then (r, s) is an *inconclusive* candidate join pair, which is forwarded to the refinement step.

In summary, APRIL's intermediate filter sequence consists of three steps: the AA -join, AF -join, and FA -join, as illustrated in Figure 3.5 and described by Algorithm 3.2. Each step is a simple merge-join between two sorted interval lists. Since each list contains disjoint intervals, each of the three interval joins takes $O(n + m)$ time, where n and m are the lengths of the two interval join input lists. Hence, the total cost of the APRIL filter (i.e., Algorithm 3.2) is linear to the total number of intervals in the A - and F -lists of r and s .

Join Order Optimization The AA -join, AF -join, and FA -join could be applied in any order in Algorithm 3.2. For example, if (r, s) is a true hit, it would be more beneficial

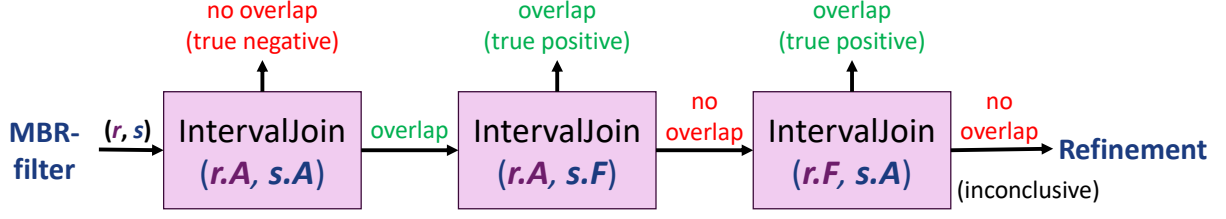


Figure 3.5: The three steps of the intermediate filter for a candidate pair of polygons.

to perform the AF -join and the FA -join before the AA -join, as this would identify the hit earlier. On the other hand, if (r, s) is a true negative, conducting the AA -join first avoids the futile AF - and FA -joins. However, it is impossible to determine a priori whether (r, s) is a true hit or a true negative. In addition, we experimentally found that changing the join order does not have a high impact on the intermediate filter cost and the overall cost. For a typical candidate pair (r, s) , the common cells are expected to be few compared to the total number of cells covered by either r or s , making AA -join the most reasonable join to start with. Our experiments confirm that the number of candidate pairs identified as true negatives is typically much larger than the number of true hits.

Algorithm 3.2 APRIL join algorithm.

Require: (r, s) such that $MBR(r)$ intersects $MBR(s)$

```

1: function INTERVALJOIN( $X, Y$ )
2:    $i \leftarrow 0$ ;  $j \leftarrow 0$ 
3:   while  $i < |X|$  and  $j < |Y|$  do
4:     if  $X_i$  overlaps with  $Y_j$  then return true ▷ overlap exists
5:     end if
6:     if  $X_i.end \leq Y_j.end$  then  $i \leftarrow i + 1$  else  $j \leftarrow j + 1$ 
7:   end while return false ▷ no overlaps detected
8: end function
9:
10: if not IntervalJoin( $A(r), A(s)$ ) then return false ▷ true negative
11: end if
12: if IntervalJoin( $A(r), F(s)$ ) then return true ▷ true hit
13: end if
14: if IntervalJoin( $F(r), A(s)$ ) then return true ▷ true hit
15: end if
16: return REFINEMENT( $r, s$ ) ▷ forward pair to refinement
17: end function

```

3.2.1.3 Generality

In this section, we demonstrate the generality of APRIL in supporting other queries besides spatial intersection joins between polygon-sets. We first show how we can use it as an intermediate filter in selection (range) queries. Then, we discuss its application in spatial *within* joins. Finally, we discuss the potential of using APRIL approximations of polygons and raster approximation of linestrings to filter pairs in polygon-linestring intersection joins.

3.2.1.4 Selection Queries

Similarly to joins, APRIL can be used in an intermediate filter to reduce the cost of selection queries. Consider a spatial database system that manages polygons and where the user can draw a selection query as arbitrary polygon QP ; the objective is to retrieve the data polygons that intersect with the query polygon QP . Assuming that we have pre-processed all data polygons and computed and stored their APRIL representations, we can process polygonal selection queries as follows. We first pre-process QP to create its APRIL approximation. Then, we use the MBR of QP to find fast the data polygons whose MBR intersects with the MBR of the query (potentially with the help of an index [6, 7]). For each such data polygon r , we apply the APRIL intermediate filter for the (r, QP) pair to find fast whether r is a true negative or a true hit. If r cannot be pruned or confirmed as a query result, we eventually apply the refinement step.

3.2.1.5 Spatial Within Joins

APRIL can also be applied for spatial joins having a *within* predicate, where the objective is to find the pairs (r, s) , where $r \in R$ and $s \in S$ and r is *within* s (i.e., r is completely covered by s). In this case, the intermediate filter performs only two of its three steps. The *AA*-join is applied first to detect whether r and s are disjoint, in which case the pair should be eliminated. Then, we perform a variant of the *AF*-join, where the objective is to find if *every* interval in the *A*-list of r is contained in one interval in the *F*-list of s ; if this is true, then (r, s) is guaranteed to be a within join result and it is reported as a true hit. In the opposite case, (r, s) is forwarded to the refinement step. We do not apply an *FA*-join, because this may only detect whether s is within r .

3.2.1.6 Linestring to Polygon Joins

Another interesting question is whether APRIL can be useful for intersection joins between other spatial data types, besides polygons. The direct answer is no, since APRIL is designed for spatially-extended objects. Still, our method can be useful in the case of joins between polygons and linestrings. A linestring is a sequence of line segments used to approximate geographic objects such as roads and rivers. The rasterization of a linestring results in only Partial cells, as linestrings have zero area and cannot cover a cell entirely. In addition, as exemplified in Figure 3.6, linestrings do not really benefit from merging consecutive cells into intervals, as linestrings that follow the Hilbert order (or any other fixed space-filling curve) are rare. Hence, it is more space-efficient to approximate a linestring as a sorted sequence of cell-IDs (which are guaranteed to be Partial). Having the linestring approximations, we can evaluate spatial intersection joins between a collection of polygons and a collection of linestrings by applying two of the three steps in the APRIL intermediate filter; namely, (i) a merge-join between the A -list of the polygon and the cell-ID list of the linestring to determine whether the pair is a true negative and (ii) a merge-join between the F -list of the polygon and the cell-ID list of the linestring to determine whether the pair is a true hit. Algorithm 3.2 can easily be adapted for polygon-linestring filtering by changing $\text{IntervalJoin}(X, Y)$ to take a sequence of cell-IDs Y and treat them as intervals of duration 1.

3.2.2 Customization

We have explored a series of optimization and customization options that can potentially reduce APRIL's space complexity and improve its performance in terms of filter effectiveness and speed.

3.2.2.1 Compression

Recall that the only information that APRIL stores for each polygon is two interval lists: the A -list and the F -list. The interval lists are essentially sorted integer arrays, so we can exploit delta encoding and more specialized lossless compression schemes to reduce their space requirements. Since any of the AA - AF - and FA -join that we may apply on the lists may terminate early (as soon as an interval overlap is detected), we should go for a compression scheme that does not require the decompression

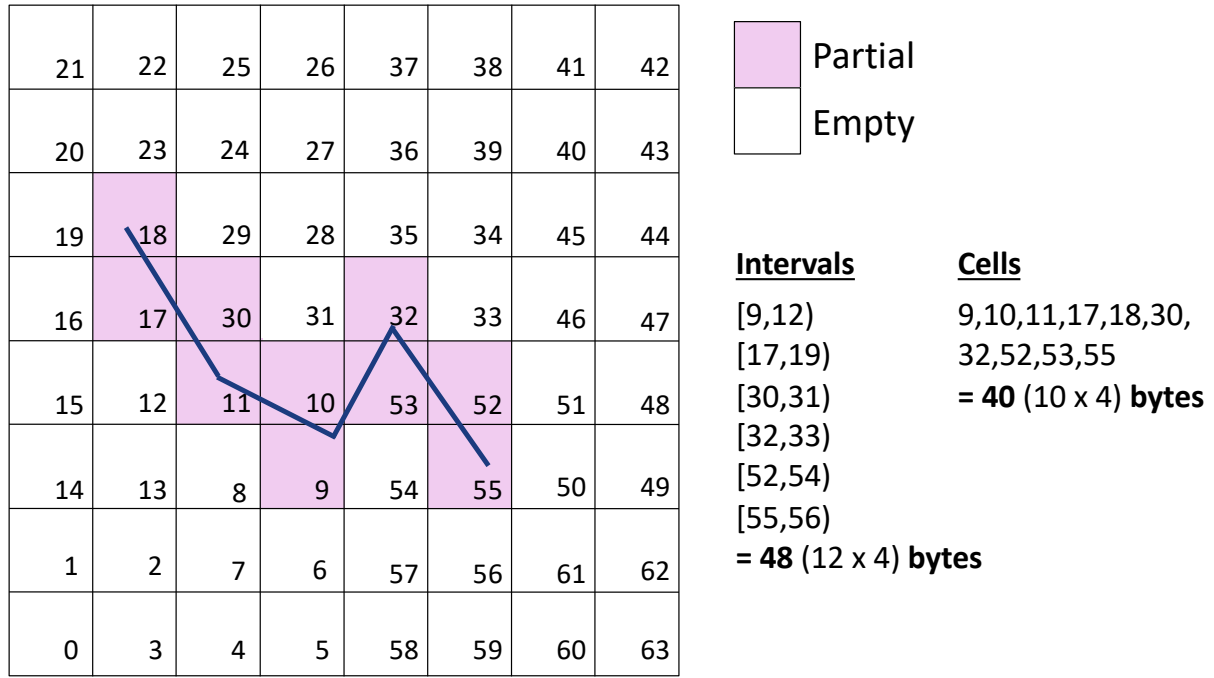


Figure 3.6: A linestring's APRIL approximation size in bytes, if stored as intervals versus cells.

a list entirely before starting processing it. In other words, we should be able to perform joins *while* decompressing the lists. This way, we may avoid uncompressing the lists in their entirety and still be able to perform the joins. Given this, we use delta encoding, where we store the first value of the list precisely and, from thereon store the differences (gaps) between consecutive numbers.

There are dozens of different compression schemes for gaps between ordered integers, each with its pros and cons. We chose the Variable Byte (VByte) method [117, 118], a popular technique that, even though it rarely achieves optimal compression, is adequately efficient and really fast [119]. We use the libvbyte [120] library that has an option for sorted integer list compression, which matches our case and boosts performance by utilizing delta encoding. Compression hardly affects APRIL's construction time, which is dominated by the rasterization/intervalization cost.

At the same time, we adapt our interval join algorithm to apply decompression and join at the same time, i.e., each time it needs to get the next integer from the list, it decompresses its value and adds it to the previous value in the list.

3.2.2.2 Partitioning

The accuracy of APRIL as a filter is intertwined with the grid granularity we choose. A more fine-grained grid results in more Full cells, increasing the chance of detecting true hits; similarly, empty cells increase, enhancing true negative detection. However, simply raising the order N is not enough to improve performance. Increasing N beyond 16 means that a single unsigned integer is not enough to store a Hilbert curve’s identifier, which range from $[0, 2^{2N} - 1]$. For $N = 17$ or higher, we would need 8 bytes (i.e., an unsigned long) to store each interval endpoint, exploding the space requirements and the access/processing cost.

Given this, we introduce a partitioning mechanism for APRIL that divides the data space into *disjoint* partitions and defines a dedicated rasterization grid and Hilbert curve of order $N = 16$ to each partition. This increases the global granularity of the approximation without using long integers while giving us the opportunity to define smaller partitions for denser areas of the map for which a finer granularity is more beneficial. Partitioning is done considering all datasets (i.e., layers) of the map. That is, the same space partitioning is used for all datasets that are joined together. The contents of each partition are all objects that intersect it; hence, the *raster* area of the partition is defined by the MBR of these objects and may be larger than the partition, as shown in the example of Figure 3.7. APRIL approximations are defined based on the raster area of the partition. The spatial join is then decomposed into multiple joins, one for each spatial partition. Duplicate join results are avoided at the filter step of the join (MBR-join), as shown in [121, 27]. This partitioning approach is particularly beneficial for big data management systems, such as Apache Sedona, where spatial queries are evaluated in a distributed manner after space and data partitioning. Partitioning allows (i) more accurate approximations through fine-grained partitions and (ii) parallel evaluation per partition that further reduces the end-to-end join time.

3.2.2.3 Different Granularity

If we use the same (fine) grid to rasterize all polygons, the APRIL approximations of large polygons may contain too many intervals, slowing down the intermediate filter. We can create approximations using a different order N of the Hilbert curve for different datasets based on the average sizes of their contents. There is a trade-off

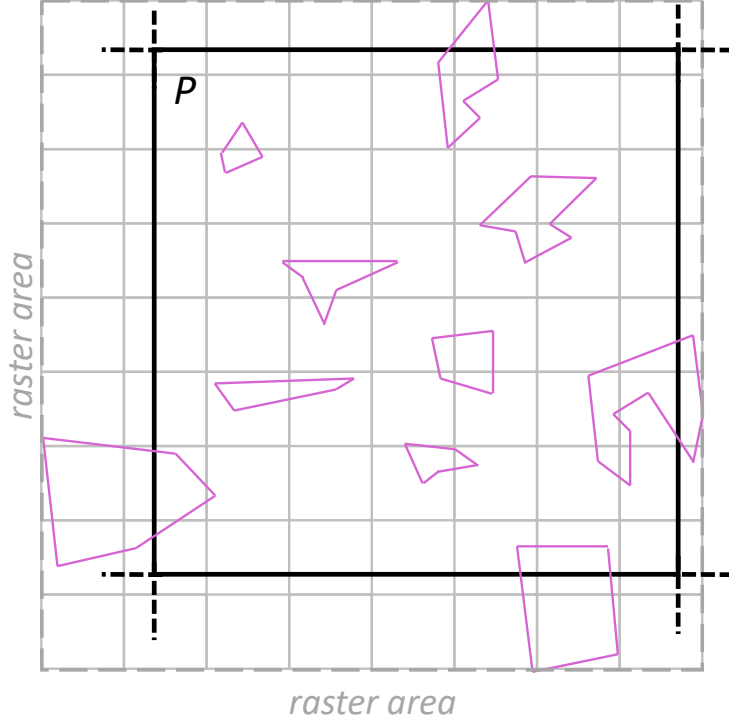


Figure 3.7: Example of a partition P , a group of polygons in it and P 's raster area with granularity order $N = 8$.

between memory and performance since an order lower than 16 means fewer intervals and thus lower memory requirements and complexity, but also means reduced APRIL accuracy.

When joining two APRIL approximations of different order, we need to adjust one of the two interval lists so that it can be joined with the other. For this, we scale down the list with the highest order. Specifically, before comparing two intervals $a = [a_{start}, a_{end})$ and $b = [b_{start}, b_{end})$ at orders N and L respectively, where $N > L$, the highest order interval a should be right shifted by $n = |N - L| \times 2$ bits, to form a transformed interval a' , as follows:

$$a' = [a_{start} \gg n, (a_{end} - 1) \gg n] \quad (3.1)$$

Right shifting creates intervals in a more coarse-grained grid, and thus, they may represent larger areas than the original. Therefore, this formula works only for A -intervals since there is no guarantee that a Full interval at order N will also be Full at order L . For this reason, in Algorithm 3.2, we perform only one of the AF - and FA - joins, using the F -list of the coarse approximation (which is not scaled down). This hurts the filter's effectiveness as a trade-off for the coarser (and smaller) APRIL

approximations that we may use for large polygons.

3.2.3 APRIL Construction

In this section, we present two methods for the construction of a polygon’s APRIL approximation. In Section 3.2.3.1 we present a *rasterization* approach that efficiently finds the cells that intersect an input polygon and their types, based on previous research on polygon rasterization, and then sorts them to construct the A - and F -interval lists. In Section 3.2.3.2, we propose a more efficient approach tailored for APRIL, which avoids classifying all cells but directly identifies the intervals and constructs the A - and F -interval lists.

3.2.3.1 Efficient Graphics-Inspired Rasterization

RI and the previous raster-based filter of [28] require the classification of each cell as Full, Strong, Weak, or Empty, based on the percentage of the cell covered by the original polygonal geometry. For this, they apply an algorithm that involves numerous polygon clippings and polygonal area computations at a high cost. On the other hand, to define an APRIL approximation, we only need to identify the cells that are partially or fully covered by the input polygon’s area. Inspired by rasterization techniques in the graphics community [122, 123], we propose a polygon rasterization technique that involves two stages. Firstly, we compute the Partial cells, which essentially form the boundary of the polygon in the grid. Next, we compute the Full cells using the previously computed boundary cells.

Identifying the Partial cells is closely related to the pixel drawing problem in graphics, which involves detecting which cells to “turn on” to draw a target line. While Bresenham’s algorithm [124] is a popular and fast pixel drawing algorithm, it approximates a line segment by turning on a minimal amount of cells and may thus not detect all intersected cells. In contrast, the Digital Differential Analyzer (DDA) method [125] is slower but identifies correctly and completely all intersected cells. To detect the Partial cells, we use an efficient variant of DDA [122] that uses grid traversal. We execute the grid traversal for each edge of the polygon and store the IDs of the identified Partial cells in a list. The leftmost grid in Figure 3.9 shows the Partial cells detected by the grid traversal algorithm for the polygon drawn in the figure.

Next, to identify the Full cells, we have to detect all cells that reside in the polygon's enclosed areas. This is closely related to the shape-filling problem in computer graphics, a very old and thoroughly studied problem. Most approaches are variants of two paradigms: sweep-line (scanline) rendering [48] or shape (flood) filling [123]. Both approaches have their pros and cons in terms of performance and accuracy.

Scanline Sweep-line algorithms use horizontal lines to find all intersections between them and the polygon per row in the grid and sort the intersections based on their x -values. These are called *event* points and are used to loop through all internal cells without performing any point-in-polygon (PiP) tests. All cells in-between consecutive pairs of the sorted event points are simply looped through and are labeled as Full. Note that this approach can be used as a standalone method without the grid traversal algorithm to detect all cells that overlap with a polygon regardless of type. However, to accurately classify each cell as Partial or Full and without performing a PiP test for every single one, we use it only for the Full cells right after the DDA algorithm has generated the set of Partial cells.

Flood Fill The classic flood fill algorithm first selects an unlabeled cell that is guaranteed to be within the polygon, called *seed*. Then, it traverses all neighboring cells of the seed until it finds the boundaries of the enclosed area, classifying the encountered cells as fully covered. We implemented a variant of this algorithm that minimizes the number of PiP tests required to identify whether a cell is inside or outside the polygon. Specifically, we iterate through the cells of the polygon's MBR area. If a cell c has not been labeled yet (e.g., as Partial), we perform a PiP check from c 's center. If the cell c is found to be inside the polygon, c is marked as Full, and we perform a flood fill using c as the seed, stopping at labeled cells and labeling all encountered unchecked cells as Full. If the cell c is found to be outside the polygon, c is marked as Empty, and we perform flood fill to mark Empty cells. The algorithm repeats as long as there are unchecked cells to flood fill from. This reduces the number of PiP tests that need to be performed, as it suffices to perform a single test for each contiguous region in the grid with Full or Empty cells.

Figure 3.8 shows an example of our sweep-line rendering variant method. After all Partial cells have been identified, the event points between the scanlines and the polygon's edges are calculated and sorted by their x -values. Then, for each two consecutive pairs of event points, their in-between unlabeled cells are labeled as Full. Figure 3.9 illustrates our Flood Fill process for an example polygon. The unchecked

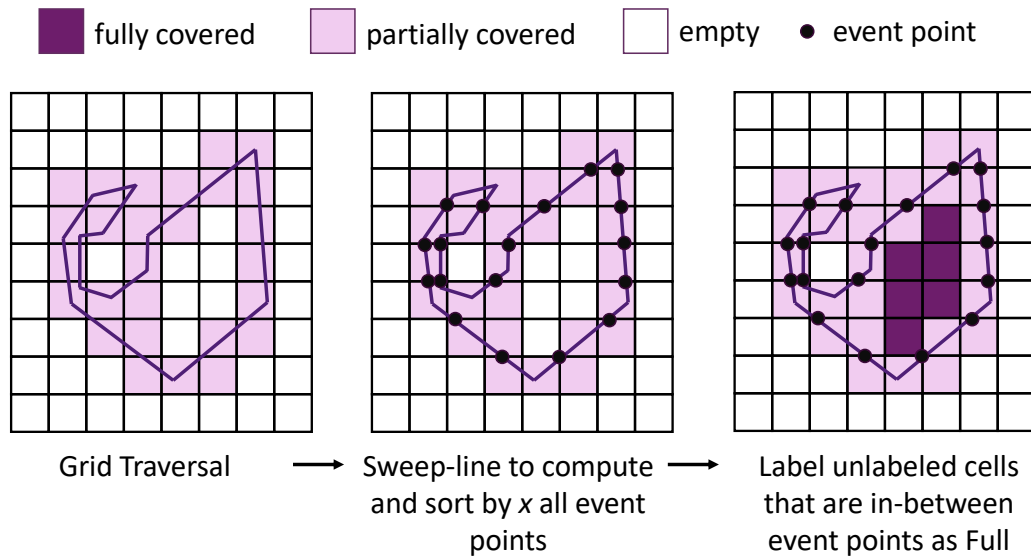


Figure 3.8: The Scanline rendering algorithm we implemented, filling the Full cells without performing any PiP tests.

cells form three contiguous regions bounded by Partial cells, two of them being outside the polygon and one inside. Instead of looking for cells within the polygon to flood fill starting from them, it is faster to fill both the inside and outside of the polygon (marking cells as Full and Empty, respectively), as the number of point-in-polygon tests is minimized.

Regardless of which rasterization approach was chosen and after all Partial and Full cells have been identified, the algorithm merges consecutive cell identifiers into intervals to create the *A*- and *F*-lists that form the APRIL approximation.

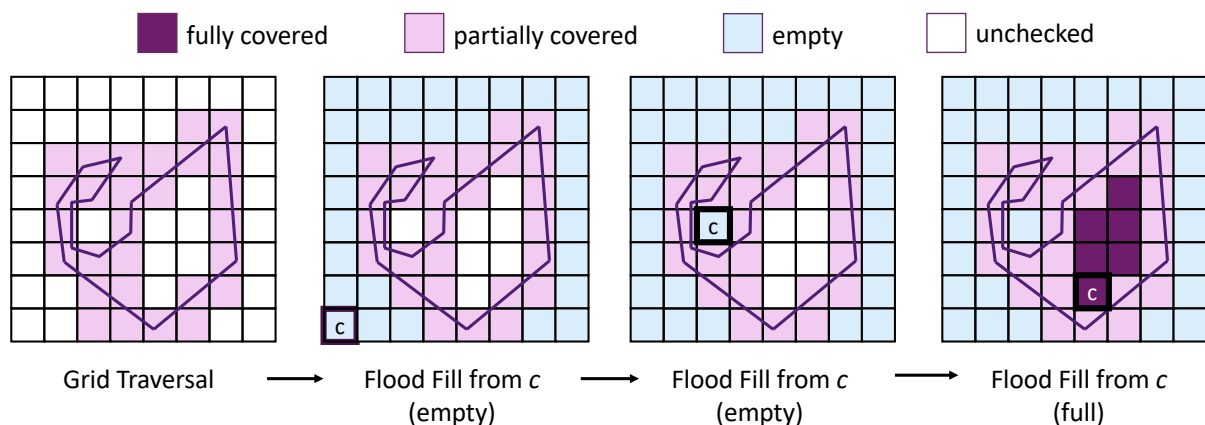


Figure 3.9: The flood fill algorithm, performing three iterations with different seeds *c* to completely fill all unchecked cells.

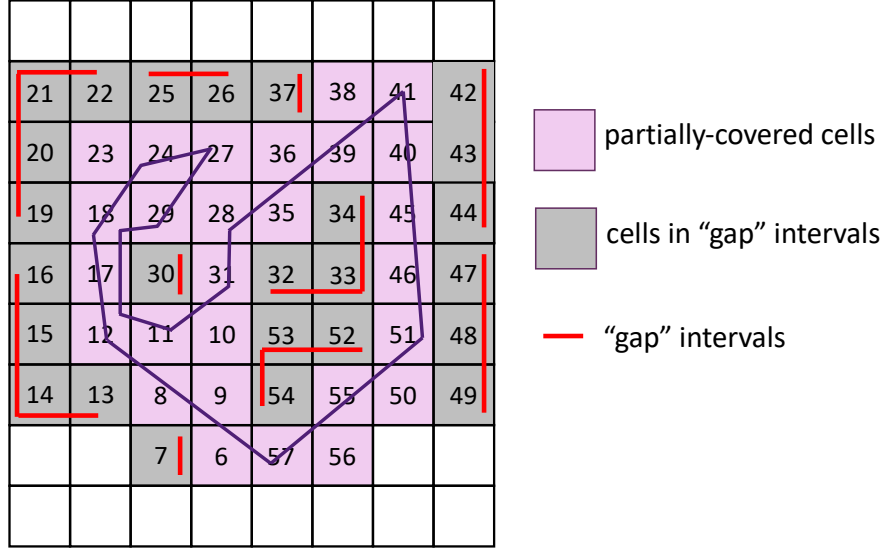
3.2.3.2 One-Step Intervalization

The approach described in the previous section identifies the types (Partial, Full, Empty) of all cells that intersect the MBR of the input polygon. For polygons that are relatively large and their MBRs define a large raster area this can be quite expensive. We propose an alternative approach that identifies the F -intervals of the APRIL approximation efficiently and directly uses them to identify the A -intervals that include them in one step, without the need to identify the types of all individual cells in them.

As in Sec. 3.2.3.1, we first apply DDA [122] to detect the Partial cells and sort them in Hilbert order. An important observation is that “gaps” between nonconsecutive identifiers in the sorted Partial cells list indicate candidate Full intervals on the Hilbert curve. Fig. 3.10 shows how these gaps are formed for an example polygon. Identifying the first cell c of each candidate interval as Full or Empty through a point-in-polygon (PiP) test is enough to label the whole interval as Full or Empty, respectively. The first “gap” interval is $[7, 8)$ containing just cell 7, which can be marked empty after a PiP test. From all “gap” intervals, those marked in **bold** (i.e., $32-34 = [32, 35)$ and $52-54 = [52, 55)$) are Full intervals and can be identified as such by a PiP test at their first cell (i.e., 32 and 52, respectively).

Additionally, we can skip some of these PiP tests by checking all adjacent cells (north, south, west, east) of the first cell c with smaller identifiers than c ; if any of them is Full or Empty, we can also give the same label to the candidate interval, as it should exist in the same inner/outer area of the raster image. For example, in Figure 3.10, when the algorithm moves to identify the interval $[52, 55)$, it can detect that its first cell 52 is adjacent to another Full cell with smaller order (cell 33), that has been previously identified. Thus, the interval $[52, 55)$ exists in the same inner area as cell 33, and it inherits its label (Full) without performing another PiP test for it. In this example, a total of 5 PiP tests will be performed for the intervals that start with the cells 7, 13, 30, 32 and 42, instead of 11 PiP tests that would be performed otherwise if we did not take into consideration the neighboring cells.

Algorithm 3.3 is a pseudocode for the one-step intervalization process, which takes as input the sorted Partial cells list P computed by DDA. The algorithm creates the A -list, F -list of the polygon in a single loop through P . In a nutshell, the algorithm keeps track of the starting point of every A -interval, and when an empty gap is identified,



[6,7,8,9,10,11,12,13-16,17,18,19-22,23,24,25-26,27,28,29,30,31,
32-34,35,36,37,38,39,40,41,42-44,45,46,47-49,50,51,52-54,55,56,57]

Figure 3.10: Example of the intervals/gaps for a set of Partial cells. Whether a gap will be labeled as Full or Empty depends on the outcome of the PiP test.

the algorithm “closes” the current A -interval and starts the next one from the next Partial cell in the list. On the other hand, Full intervals start with the identifier of the cell that is right after the last Partial cell of a consecutive sequence and end before the next Partial cell in order.

In detail, Algorithm 3.3, starting from the first cell p in P , keeps track of the starting cell-ID A_{start} of the current A -interval, while the next cell $p + 1$ in Hilbert order is also in P (Lines 3–9) the current A -interval is expanded. If the next cell $c = p + 1$ is not partial, it is the starting cell of a candidate F -interval. We first apply function $CheckNeighbors(c)$ to find whether there exists an adjacent cell of c which is part of a $FULL$ or $EMPTY$ interval. Specifically, for cell c and a neighbor n , we first check whether $n < c$ (if not, n is either Partial or unchecked); if yes, we binary-search P to check whether n is a P -cell. If not, we apply a special binary search method on the current F -list to find out whether n is part of an interval in it. If we find n as part of an F -interval, then c is definitely a Full cell. If we do not find n , then c is definitely an Empty cell because $n < c$ and n is not Partial. If for all neighbors n of c , either $n > c$ or n is Partial, then we cannot determine the type of c based on the current data, so we perform a PiP test to determine c ’s type (i.e., Full or Empty). If c is Full, then we know that the entire interval $[c, p)$ is $FULL$ and append it to the

F -list (Line 16). Otherwise (c is Empty), c is the end of the current A -interval, so the interval is added to the A -list, and the start of the next A -interval is set to the next Partial cell p . The algorithm continues until the list P of partial cells is exhausted and commits the last A -interval (Line 23).

Algorithm 3.3 The One-Step Intervalization algorithm.

Require: Sorted Partial cell array P

```

1: function ONESTEPINTERVALIZATION( $P$ )
2:    $i \leftarrow 0$  ▷ current position in array  $P$ 
3:    $Astart \leftarrow P_i; p \leftarrow P_i$  ▷ cell-IDs of current  $A$ -interval and partial cell
4:   while  $i < |P|$  and  $p + 1 = P_{i+1}$  do ▷ while next cell is partial
5:      $i \leftarrow i + 1$ 
6:      $p \leftarrow P_i$ 
7:   end while
8:    $c \leftarrow p + 1$  ▷ next uncertain cell
9:    $i \leftarrow i + 1; p \leftarrow P_i$  ▷ next partial cell
10:  while  $i < |P|$  do
11:     $type \leftarrow CheckNeighbors(c)$ 
12:    if  $type \neq FULL$  and  $type \neq EMPTY$  then ▷  $type$  is still uncertain
13:       $type \leftarrow PointInPolygon(c)$  ▷ PiP test gives  $FULL$  or  $EMPTY$ 
14:    end if
15:    if  $type = FULL$  then
16:       $AppendFullInterval([c, p])$ 
17:    else ▷  $type$  is  $EMPTY$ 
18:       $AppendAllInterval([Astart, c])$  ▷ current  $A$ -interval finalized
19:       $Astart \leftarrow p$  ▷ start new  $A$ -interval
20:    end if
21:    Execute Lines 3–9 ▷ go through partial cells until next gap
22:  end while
23:   $AppendAllInterval([Astart, P_{i-1} + 1])$  ▷ save last  $ALL$  interval
24: end function

```

Our one-step intervalization approach performs $|P| - 1$ PiP tests in the worst case, which dominate its cost. Compared to the FloodFill-based approach of Section 3.2.3.1, which explicitly marks and then sorts all Full and Partial cells, Algorithm 3.3 is expected to be much faster for polygons that are large compared to the cell size and include a huge number of Full cells. On the other hand, flood filling may be a better fit for small polygons with a small MBR and relatively few Full cells.

Table 3.11: Statistics of the datasets and space requirements of the data and the approximations

	T1	T2	T3	O5AF	O6AF	O5AS	O6AS	O5EU	O6EU	O5NA	O6NA	O5SA	O6SA	O5OC	O6OC
# of Polygons	123K	2.25M	3.1K	72K	191K	447K	622K	1.9M	7.1M	4.0M	999K	123K	228K	107K	223K
Avg # of vertices	25.4	31.9	2285.0	58.9	36.3	45.3	41.9	35.1	32.1	37.6	47.5	47.5	41.6	48.4	42.7
Avg obj MBR area	1.77E-04	4.03E-05	3.95E-01	2.03E-03	1.23E-03	1.03E-03	9.98E-04	1.25E-04	1.19E-04	1.11E-04	4.40E-04	1.34E-03	2.37E-03	5.00E-04	5.27E-04
Geometries size (MB)	51.1	1168.1	115.3	68.9	112.7	327.9	422.1	1120.7	3746.2	2453.4	767.4	94.9	153.7	84.2	151.3
MBR size (MB)	4.4	81.1	0.1	2.6	6.9	16.1	22.4	70.9	258.4	144.8	36.0	4.5	8.2	3.9	8.1
APRIL size (MB)	14.4	134.0	57.2	14.2	25.4	55.2	64.5	180.3	968.0	251.0	155.0	25.4	44.4	7.3	15.0
APRIL-C size (MB)	6.6	75.3	16.0	5.1	10.6	23.3	28.6	84.8	406.5	138.0	62.4	9.2	16.7	3.8	7.8
RI size (MB)	19.5	138.2	968.7	18.6	55.7	57.5	109.8	180.9	942.9	238.1	213.5	31.2	143.4	14.2	39.3
RA size (MB)	1100.0	20000.0	26.9	617.2	1700.0	3700.0	5700.0	342.2	11400.0	6200.0	1500.0	1100.0	2100.0	898.7	2000.0
5C-CH size (MB)	28.7	705.4	1.6	18.5	46.6	117.8	159.4	515.4	1700.0	1200.0	257.7	30.4	52.9	28.8	57.7

3.2.4 Experimental Analysis

We assess the performance of our proposed methods (i.e., RI and APRIL) by experimentally comparing them with previously proposed polygon approximations for intermediate filtering of spatial joins in a single machine without any thread parallelism. These include the 5-corner approximations comparison followed by a comparison of convex hulls (5C+CH) (as proposed in [41]), and Raster Approximation (RA) of [28]. We also included a baseline approach (None), which does not apply an intermediate filter between the MBR-join and the refinement step. For RA, we set the grid resolution to $K = 750$ cells, except for a few datasets where we use $K = 100$, due to memory constraints. For our methods (RI and APRIL), unless otherwise stated, we use a granularity order $N = 16$ for the rasterization grid, meaning that the Hilbert order of each cell can be represented by a 32-bit unsigned integer. The MBR filter of the spatial join pipeline was implemented using the algorithm of [27]. The refinement step was implemented using the Boost Geometry library (www.boost.org) and its functions regarding shape intersection. All code was written in C++ and compiled with the -O3 flag on a machine with a 3.6GHz Intel i9-10850k and 32GB RAM, running Linux.

3.2.4.1 Datasets

We used datasets from SpatialHadoop’s [51] collection. T1, T2, and T3 represent landmark, water, and county areas in the United States (conterminous states only). We also used two Open Street Maps (OSM) datasets (O5 and O6) that contain lakes and parks, respectively, from all around the globe. We grouped objects into continents and created six smaller datasets representing each one: Africa (O5AF, O6AF),

Asia (O5AS, O6AS), Europe (O5EU, O6EU), North America (O5NA, O6NA), Oceania (O5OC, O6OC) and South America (O5SA, O6SA). From all datasets, we removed non-polygonal objects as well as multi-polygons, self-intersecting polygons, and polygons with holes because they need special handling by the refinement algorithms and the Boost Geometry library that we are using does not support some of these data types. The construction algorithms for APRIL approximations, presented in Section 6 can easily be adapted to handle these special polygon classes with minor modifications; our raster approximations for such polygons can be used directly as intermediate filters. The first three rows of Table 3.11 show statistics about the datasets. The cardinalities of the datasets vary from 3.1K to 7.1M. The smallest dataset (T3) includes complex polygons (thousands of edges), each having a relatively large area (see the third row of Table 3.11). The other datasets are larger and include medium (e.g., T1, OSM data) to small and relatively simple polygons (e.g., T2). We conducted spatial joins only between pairs of datasets that cover the same area (i.e., $T1 \bowtie T2$, $T1 \bowtie T3$, $O5AF \bowtie O6AF$, etc.).

3.2.4.2 Optimizations and Customizations

In this set of experiments, we showcase how the added features of APRIL perform both independently and compared to RI. Additionally, we compare APRIL with RI in terms of space complexity, filter effectiveness, filter cost, and creation time.

3.2.4.2.1 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 3.1.1.2, we set the value of N to 16, to have a fine granularity and be able to store the interval endpoints in 4-byte unsigned integers. We confirm the appropriateness of this choice by evaluating the effectiveness of both RI and APRIL in spatial joins for various values of N .

Table 3.12 analyzes the performances of RI and APRIL for different values of N in spatial join $T1 \bowtie T2$. The first three columns of the table show the percentage of candidate pairs identified by the intermediate filters 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 intermediate filters that use RI and APRIL 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 the RI filter increases with N as the number and length of intervals increase. Eventually, for the largest value of N , the overall join cost converges to less than 1 second.

In Table 3.13, we show the total time required to compute the RI and APRIL 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 the intermediate filters are 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 the intermediate filters are most 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 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 do 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 [28]).

3.2.4.2.2 Join Order

So far, the interval joins in APRIL are assumed to be applied in a fixed order: AA , AF , and FA . As discussed in Section 3.2.1.2, the joins can be performed in any order. Table 3.14 tests different join orders for $T1 \bowtie T2$ and $T1 \bowtie T3$. $T1 \bowtie T2$ (like the majority of tested joins) has a high percentage of true negatives, so the original order is the most efficient one (changing the order of AF and FA does not make a difference). On the other hand, for $T1 \bowtie T3$, where the true hits are more, pushing the AA -join at the end is more beneficial. Since knowing the number (or probability) of true negatives and true hits a priori is impossible, and because the join order does

Table 3.12: Effect of N on the performance of RI and APRIL in $T1 \bowtie T2$

	True hits	False hits	Indecisive	MBR-join (s)	RI-filter (s)	Refinement (s)	Total time(s)
	$T1 \bowtie T2$ (RI)						
$N = 10$	5.68%	24.96%	69.36%	0.03	0.03	1.44	1.50
$N = 13$	13.34%	46.88%	39.79%	0.03	0.06	0.63	0.72
$N = 14$	17.74%	52.20%	30.06%	0.03	0.09	0.48	0.60
$N = 15$	21.65%	56.07%	22.28%	0.03	0.15	0.37	0.54
$N = 16$	24.50%	59.42%	16.08%	0.03	0.28	0.27	0.59
	$T1 \bowtie T2$ (APRIL)						
$N = 10$	5.67%	24.96%	69.37%	0.03	0.03	1.45	1.52
$N = 13$	13.46%	46.88%	39.66%	0.03	0.04	0.61	0.68
$N = 14$	17.99%	52.20%	29.81%	0.03	0.04	0.45	0.52
$N = 15$	21.85%	56.07%	22.08%	0.03	0.04	0.34	0.41
$N = 16$	24.29%	59.42%	16.29%	0.03	0.05	0.26	0.34

Table 3.13: Effect of N on the cost and space of RI and APRIL for $T1$ and $T2$

$T1$	RI constr. cost (s)	APRIL constr. cost (s)	RI Size (MB)	APRIL Size (MB)
$N = 10$	0.98	0.29	2.6	3.0
$N = 13$	5.32	0.55	3.5	3.6
$N = 14$	13.90	0.85	4.7	4.4
$N = 15$	43.17	1.37	8.2	7.7
$N = 16$	148.72	2.37	19.0	13.8
$T2$	RI constr. cost (s)	APRIL constr. cost (s)	RI Size (MB)	APRIL Size (MB)
$N = 10$	15.29	5.68	46.0	53.0
$N = 13$	43.95	8.08	53.0	58.4
$N = 14$	87.35	11.23	62.0	66.7
$N = 15$	214.04	16.57	82.0	84.1
$N = 16$	620.57	26.76	132.0	128.0

Table 3.14: Join order effect on APRIL filter cost.

Join Order	True hits	True negatives	Indecisive	Int. Filter (s)
T1 $\bowtie$ T2				
AA-AF-FA	24.29%	59.42%	16.29%	0.0505
AA-FA-AF	24.29%	59.42%	16.29%	0.0501
AF-FA-AA	24.29%	59.42%	16.29%	0.0585
FA-AF-AA	24.29%	59.42%	16.29%	0.0601
T1 $\bowtie$ T3				
AA-AF-FA	69.84%	28.13%	2.03%	0.1872
AA-FA-AF	69.84%	28.13%	2.03%	0.1891
AF-FA-AA	69.84%	28.13%	2.03%	0.1737
FA-AF-AA	69.84%	28.13%	2.03%	0.1773

not make a big difference in the efficiency of the filter (especially in the end-to-end join time), we suggest using the fixed order, which is the best one in most tested cases. In the future, we will investigate the use of data statistics and/or object MBRs to fast guess a good join order on an object pair basis.

3.2.4.2.3 Partitioning

Tables 3.15 and 3.16 illustrate the effect of data partitioning (Section 3.2.2.2 on the effectiveness, query evaluation time, and space requirements of APRIL approximations. A higher number of partitions means finer-grained grids per partition and thus, more intervals per polygon (i.e., more space is required). Even though this reduces the number of inconclusive cases, it can slow down the intermediate filter since more intervals need to be traversed per candidate pair. For example, T1 $\bowtie$ T3 has already a small percentage of inconclusive pairs, so partitioning may not bring a significant reduction in the total join time. On the other hand, for joins with a high inconclusive percentage, such as O5AS $\bowtie$ O6AS, partitioning can greatly reduce the total cost. In summary, partitioning comes with a time/space tradeoff.

3.2.4.2.4 Different Granularity

As discussed in Section 3.2.2.3, we can define and use APRIL at lower granularity than $N = 16$ for one or both datasets, trading filter effectiveness for space savings.

Table 3.15: # partitions per dimension effect on join time.

#	Indecisive	Int. Filter (s)	Refinement (s)	Total time (s)
T1 $\bowtie$ T2				
1	16.29%	0.08	0.27	0.39
2	12.81%	0.06	0.22	0.32
3	11.36%	0.08	0.20	0.30
4	10.50%	0.09	0.20	0.32
T1 $\bowtie$ T3				
1	2.03%	0.47	0.34	0.86
2	1.77%	0.29	0.29	0.62
3	1.67%	0.37	0.27	0.69
4	1.64%	0.49	0.26	0.80
O5AF $\bowtie$ O6AF				
1	26.92%	0.06	0.36	0.45
2	21.24%	0.06	0.29	0.37
3	18.26%	0.07	0.25	0.34
4	16.63%	0.08	0.24	0.35
O5AS $\bowtie$ O6AS				
1	30.76%	0.43	7.48	8.04
2	24.07%	0.41	5.30	5.83
3	20.52%	0.46	4.34	4.93
4	18.39%	0.55	3.61	4.29
O5EU $\bowtie$ O6EU				
1	34.32%	5.83	30.55	38.01
2	27.97%	5.35	24.24	31.22
3	24.84%	6.06	21.55	29.24
4	22.60%	6.61	19.99	28.23
O5NA $\bowtie$ O6NA				
1	22.26%	3.56	24.08	28.49
2	17.58%	3.14	18.81	22.81
3	15.68%	3.65	17.13	21.64
4	14.45%	4.52	16.02	21.40
O5SA $\bowtie$ O6SA				
1	25.80%	0.17	1.44	1.66
2	20.74%	0.14	1.21	1.39
3	18.39%	0.17	1.12	1.33
4	17.03%	0.20	1.07	1.30
O5OC $\bowtie$ O6OC				
1	24.42%	0.10	1.51	1.65
2	18.89%	0.12	1.09	1.25
3	16.17%	0.14	0.95	1.13
4	14.65%	0.16	0.88	1.08

Table 3.16: # of partitions per dimension, effect on APRIL size (MB).

#	T1	T2	T3	O5AF	O6AF	O5AS	O6AS	O5EU	O6EU	O5NA	O6NA	O5SA	O6SA	O5OC	O6OC
1	14.4	134.0	57.2	14.2	25.4	55.2	64.5	180.3	968.0	251.0	155.0	25.4	44.4	7.3	15.0
2	26.1	236.3	112.0	29.2	49.2	106.9	124.2	336.9	1900.0	453.4	311.8	51.5	86	14.3	49.2
3	37.1	352.6	166.7	44.7	74.2	164.0	188.3	492.5	2800.0	654.2	459.6	76.9	129.8	35.2	76.3
4	47.2	465.9	224.9	61.4	99.5	219.1	255.1	653.0	3700.0	875.1	619.0	104.2	172.3	49.1	107.7

Table 3.17: Join between T1 (order 16) and T3 (order N).

N	True hits	True negs.	Indecisive	Int. Filter (s)	Refinement (s)	Total (s)	T3 size (MB)
16	69.84%	28.13%	2.03%	0.19	0.33	0.57	57.2
15	69.63%	27.85%	2.52%	0.13	0.41	0.59	28.3
14	69.18%	27.46%	3.36%	0.11	0.54	0.70	14.0
13	68.39%	26.86%	4.75%	0.09	0.78	0.92	6.9
12	66.63%	25.70%	7.67%	0.09	1.23	1.37	3.4

In Table 3.17, we study the effect of reducing N for T3 in $T1 \bowtie T3$. The size of T3’s APRIL approximations halves every time we decrease N by one. The filter time also decreases due to the reduced amount of intervals from T3 in the interval joins. However, the percentage of indecisive pairs increases, raising the refinement cost. $N = 15$ is the best value for T3 because it achieves the same performance as $N = 16$ while cutting the space requirements in half.

3.2.4.3 APRIL Construction Cost

We now evaluate the APRIL construction techniques that we have proposed in Section 3.2.3, comparing them with the rasterization method used in previous work [28] (and for RI). Note that RA [28] and RI essentially apply polygon clipping and polygon-cell intersection area computations, because they need to classify the cells that intersect the polygon to Weak, Strong, and Full. On the other hand, APRIL uses two classes: Partial and Full, which enables the application of the techniques that we proposed in Section 3.2.3. Table 3.18 shows the time taken to compute the APRIL approximations of all polygons in each dataset (for $N = 16$), using (i) the rasterization+intervalization approach of RI, after unifying Strong and Weak cells, (ii) the Scanline and FloodFill approaches tailored for APRIL, and (iii) two versions of our novel OneStep intervalization approach (Section 3.2.3.2): one that performs a point-in-polygon (PiP) test for each first cell c of a candidate Full interval and one that checks the Neighbors of c

Table 3.18: Total construction cost (sec) for all datasets.

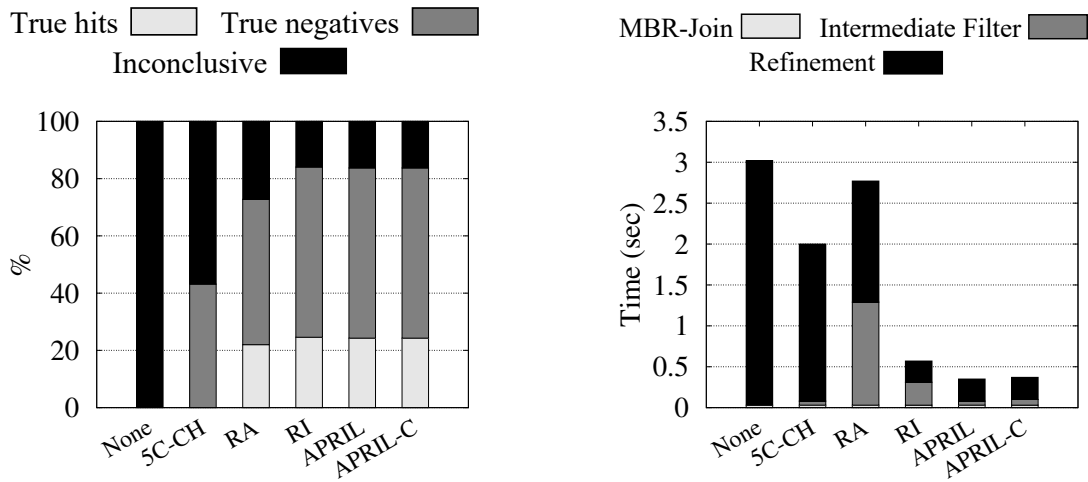
Dataset	RI	Flood Fill	Scanline	IDEAL	OneStep (PiPs)	OneStep (Neighbors)
T1	143.62	3.90	3.63	9.76	3.74	2.19
T2	601.67	28.05	23.06	37.40	33.76	23.43
T3	9919.06	265.72	278.50	666.19	75.40	28.33
O5AF	264.45	4.25	3.98	13.02	11.00	4.72
O6AF	468.47	13.06	12.62	32.45	5.66	4.17
O5AS	486.86	11.69	10.07	27.42	21.28	11.78
O6AS	994.93	28.98	24.76	56.14	65.01	25.07
O5EU	1193.71	36.08	30.30	58.18	55.79	33.71
O6EU	5493.15	172.20	147.29	244.95	243.17	156.94
O5NA	1530.92	53.33	45.26	72.34	133.39	66.60
O6NA	1630.29	43.40	40.89	76.62	51.79	30.71
O5SA	361.87	6.67	5.79	20.69	14.74	6.77
O6SA	1478.05	34.56	34.15	98.20	22.86	10.52
O5OC	39.99	2.88	2.48	6.17	3.82	2.49
O6OC	113.99	9.32	8.49	18.29	20.75	8.56

before attempting the PiP test. All costs in Table 3.18 include the intervalization cost as well to generate the final interval lists needed for APRIL. The intervalization for the rasterization techniques is performed by merging cells with consecutive Hilbert order identifiers into intervals, while our methods from Section 3.2.3.2 generate the intervals straight away.

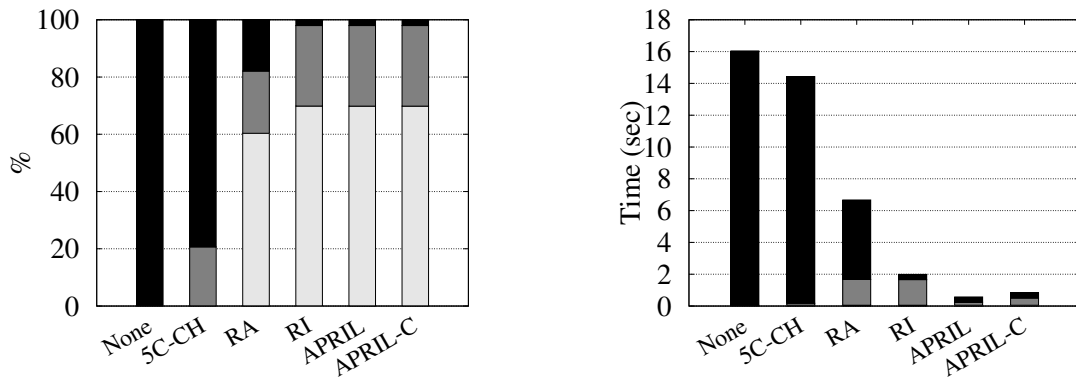
We also included in the comparison the rasterization technique proposed for IDEAL [23], which also detects Full and Partial cells, as implemented in [126]. We modified IDEAL’s granularity definition formula accordingly to match APRIL’s Hilbert space grid of order $N = 16$.

Observe that our OneStep intervalization algorithm employing the Neighbors check (Section 3.2.3.2) and our Scanline rendering (Section 3.2.3.1) are the fastest approaches in most cases. Scanline and Flood Fill show little to no difference in performance between them, with Scanline being overall faster for small polygons and Flood Fill being faster for large and more complex shapes. This is because Flood Fill performs some PiP tests while Scanline does not, as well as Flood Fill filling some unnecessary (Empty) pixels outside of the polygon, while Scanline focuses entirely on the interior of the shape specified by the event points. OneStep (Neighbors) ap-

plies 40% – 70% fewer PiP tests compared to OneStep (PiPs) that does not apply the Neighbors check. Only in datasets containing relatively small polygons OneStep (Neighbors) is up to 32% slower than the Scanline method, however, in most such cases, their difference is negligible. On the other hand, in some datasets containing large polygons (e.g., T3, O6AF, O6SA) OneStep is up to one order of magnitude faster than Scanline (T3) and 33% to 224% faster than the rest of the methods. All methods proposed in Section 3.2.3 are orders of magnitude faster compared to rasterization for RI because the latter has to perform expensive detection for Strong and Weak cells.

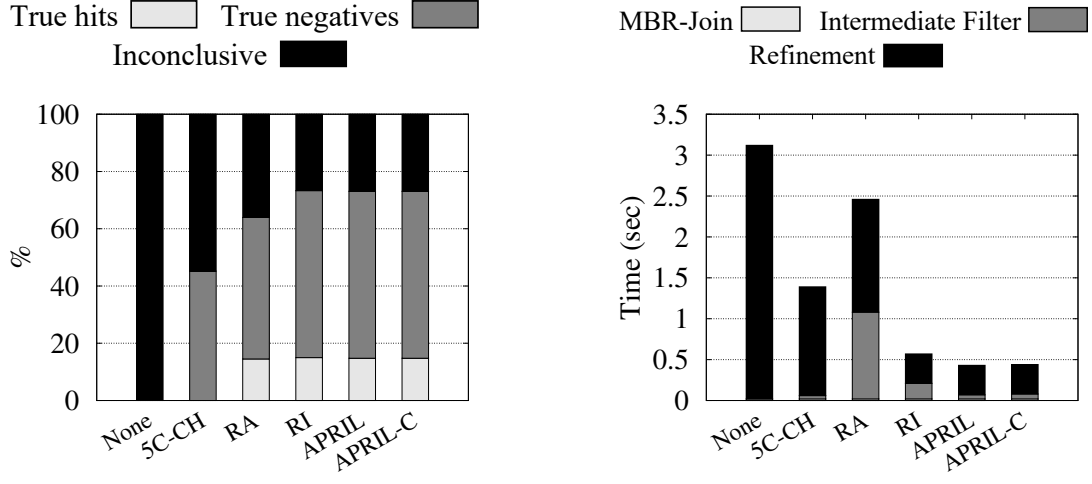


(a) $T1 \bowtie T2$

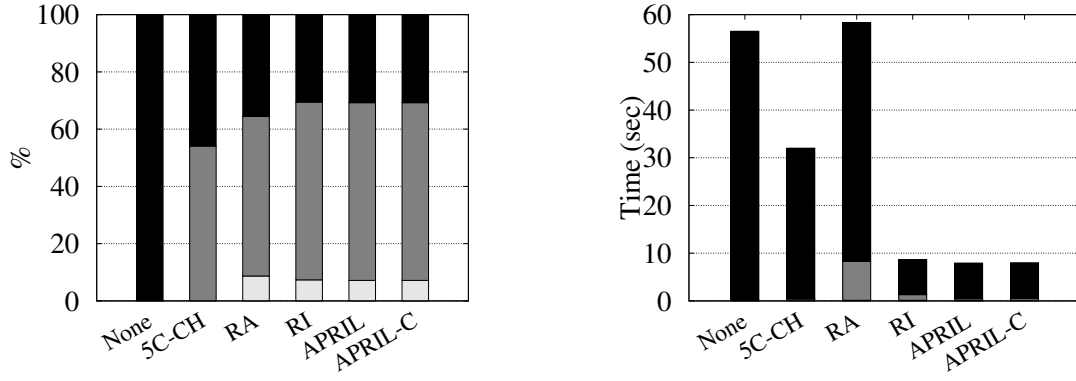


(b) $T1 \bowtie T3$

Figure 3.11: Filter effectiveness and spatial join cost for T1 dataset joins.



(a) O5AF $\bowtie$ O6AF



(b) O5AS $\bowtie$ O6AS

Figure 3.12: Filter effectiveness and spatial join cost for O5O6 AF and AS dataset joins.

3.2.4.4 Comparative Study

Finally, we compare RI and APRIL with other intermediate filters in terms of space complexity, filter effectiveness, and filter cost. For all experiments, we created RI and APRIL using a single partition (i.e., the map of the two datasets that are joined in each case), rasterized on a $2^{16} \times 2^{16}$ grid, which is the best-performing granularity for both methods. We used a fixed order (AA-, AF-, FA-) for the interval joins of APRIL, as shown in Algorithm 3.2.

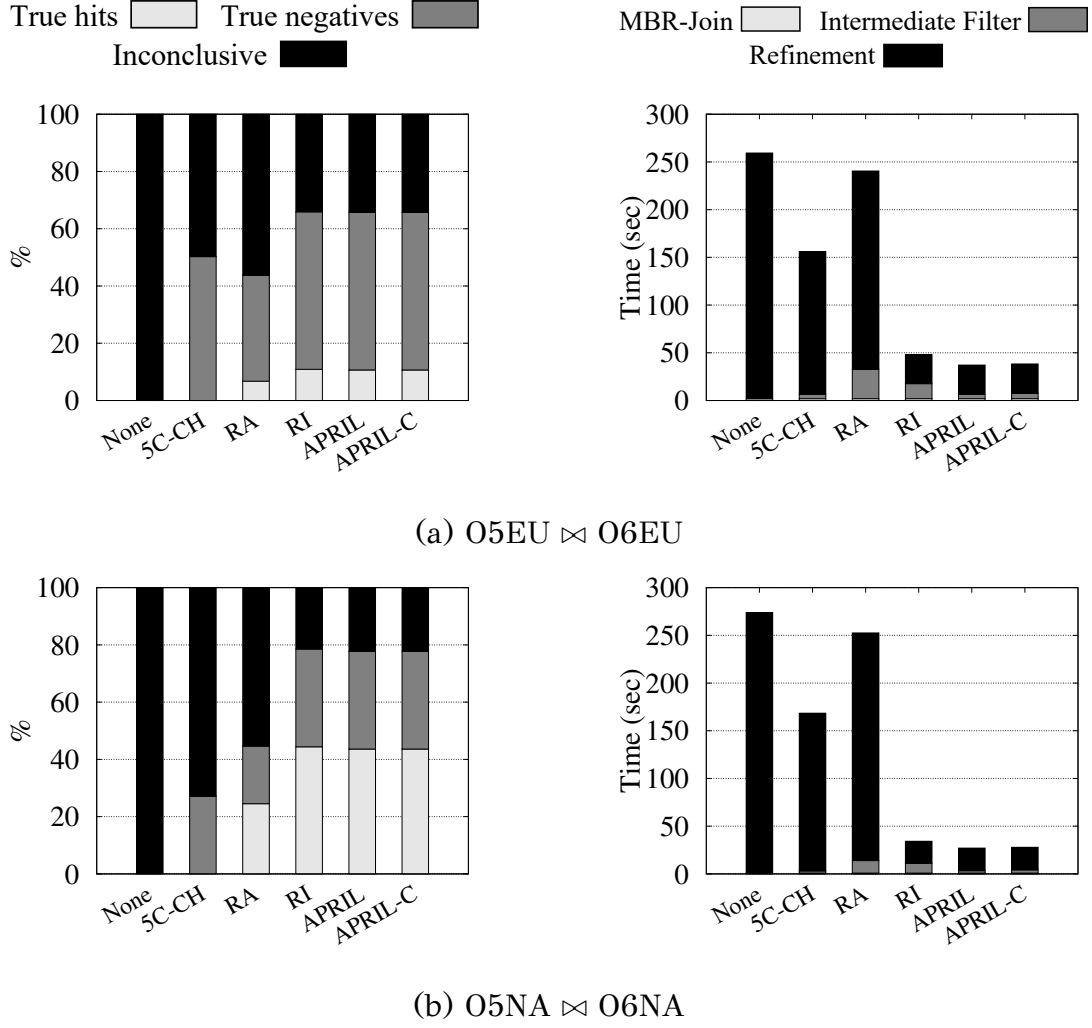
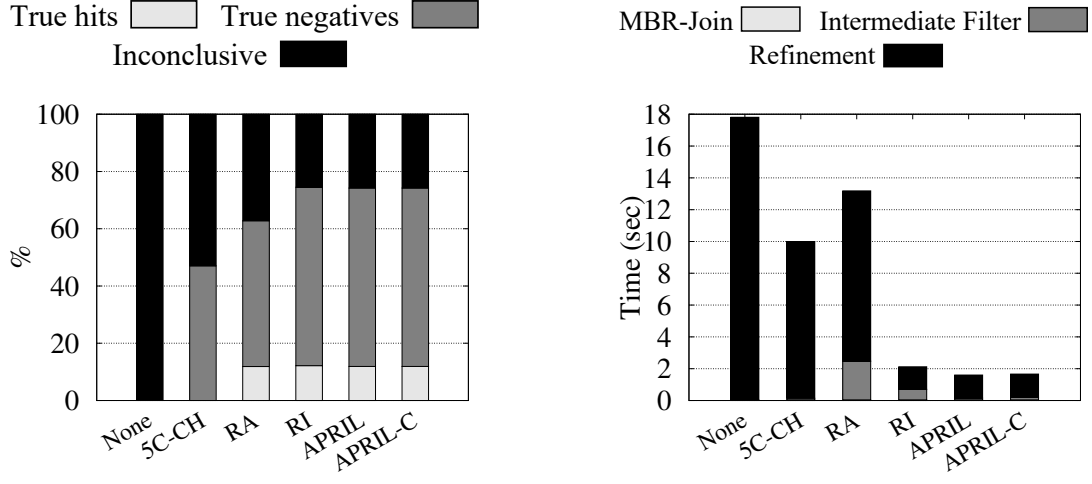


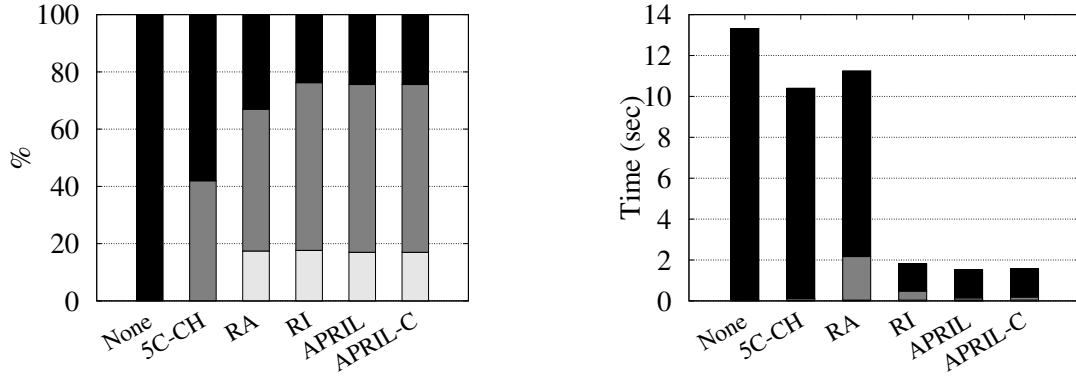
Figure 3.13: Filter effectiveness and spatial join cost for O5O6 EU and NA dataset joins.

3.2.4.4.1 Space Complexity

Table 3.11 shows the total space requirements of the object approximations required by each intermediate filter for each of the datasets used in our experiments. APRIL and APRIL-C refer to the uncompressed and compressed version of APRIL, respectively. As a basis of comparison, we also show the total space required to store the exact geometries of the objects and their MBRs. Note that, in most cases, our methods (RI, APRIL and APRIL-C) are significantly more space efficient compared to RA and have similar or lower space requirements to the 5C-CH. The only exception is T3, which includes huge polygons that are relatively expensive to approximate even by APRIL-C. Notably, for most datasets, the compressed APRIL approximations have similar space requirements as the object MBRs, meaning that we can keep them in



(a) O5SA $\bowtie$ O6SA



(b) O5OC $\bowtie$ O6OC

Figure 3.14: Filter effectiveness and spatial join cost for O5O6 SA and OC dataset joins.

memory and use them in main-memory spatial joins [20] directly after the MBR-join step without incurring any I/O.

3.2.4.4.2 Comparison in Spatial Intersection Joins

We evaluate APRIL (both compressed and uncompressed versions), 5C+CH, RA, and RI, on all join pairs, in Figures 3.11, 3.12, 3.13 and 3.14. We compare their ability to detect true hits and true negatives, their computational costs as filters, and their impact on the end-to-end cost of the spatial join.

Filter Effectiveness APRIL and RI have the highest filter effectiveness among all approximations across the board. APRIL's true hit ratio is slightly smaller compared to that of RI because APRIL fails to detect the (rare) pairs of polygons which only

have Strong-Strong common cells. However, this only brings a marginal increase in the refinement step's cost, with the benefit of having a faster and more space-efficient filter. In $O5AS \bowtie O6AS$ and $O5OC \bowtie O6OC$, APRIL and RI have marginally lower true hit ratio compared to RA; however, in these cases, their true negative ratio is much higher than that of RA. The least effective filter is $5C+CH$, mainly due to its inability to detect true hits.

Intermediate Filter cost $5C+CH$ are simple approximations (a few points each) thus, the corresponding filter is very fast to apply. Notably, APRIL has a filtering cost very close to that of $5C+CH$ and sometimes even lower. This is due to APRIL's ability to model a raster approximation as two sequences of integers, which are processed by a sequence of efficient merge-join algorithms. $5C+CH$ has poor filtering performance, which negatively affects the total join cost (last column), whereas APRIL is very fast and very effective at the same time. The state-of-the-art filter RI is more expensive than APRIL, because it requires the alignment and bitwise *AND*ing of the interval bit-codes. As a result, APRIL is 3.5-8.5 times faster as an intermediate filter compared to RI (note the "Intermediate Filter" part of the cost in the bars). A comparison between the filter costs of APRIL and APRIL-C reveals that decompressing the interval lists while performing the joins in APRIL-C only brings a small overhead, making compression well worthy, considering the space savings it offers (see Table 3.14). The decompression cost is significant only in $T1 \bowtie T3$, because $T3$'s *A*-lists and *F*-lists are quite long. Still, even in this case, APRIL-C is much faster than RI. **Refinement cost** The refinement cost is intertwined with the percentage of indecisive pairs. The detection of fewer candidate pairs as true hits or true negatives leads to a higher refinement workload; this is why APRIL and RI result in the lowest refinement cost compared to the rest of the approximations.

Overall join cost APRIL (Section 3.2.1) reduces the overall cost of end-to-end spatial joins up to 3.5 times compared to using our RI intermediate filter (Section 3.1.1), while also achieving a speedup of 3.23x-25x against the rest of the approximations. Adding the APRIL intermediate filter between the MBR filter and the refinement step reduces the spatial join cost by 7x-28x. APRIL's high filtering effectiveness, low application cost, and low memory requirements render it a superior approximation for filtering pairs in spatial intersection join pipelines.

APRIL vs. RI In summary, APRIL prevails over RI on all aspects including space complexity, construction time, filtering efficiency, and overall spatial intersection join

Table 3.19: Improvement of APRIL over RI on all join pairs

APRIL vs. RI	range	average
APRIL size (times smaller)	0.95x–16.94x	2.59x
APRIL-C size (times smaller)	1.73x–60.54x	7.39x
Construction (times faster)	13.32x–350.13x	70.71x
Intermediate Filter (times faster)	3.45x–8.56x	4.86x
End-to-end Join (times faster)	1.10x–3.51x	1.58x

time. Table 3.19, summarizes the improvement that APRIL achieves over RI on all join pairs.

3.2.4.4.3 Effect of variance in object sizes

We now test the effect that the variance between the sizes of joined objects has on the performance of APRIL compared to previous work. For this, besides T1, T2, and T3, we used two more Tiger datasets, i.e., T9 (States) and T10 (Zip codes). Table 3.21 lists statistics of all the Tiger datasets that we used in this experiment ordered by average area of the objects in them.

To demonstrate APRIL’s performance on dataset pairs of varying object sizes, we joined the Water Areas dataset (T2), which has polygons with the smallest areas on average compared to other Tiger datasets, in increasing order of average area per object. Note that as the objects which are joined with T2-objects grow larger (T1, T10, T3 and then T9) the indecisive cases reduce drastically. This can be explained by the fact that as a polygon grows larger, it generates more Full intervals, and thus, the probability of detecting a true hit between it and another polygon using APRIL increases. Table 3.20 shows a detailed performance comparison for the spatial intersection join between the TIGER datasets. In all cases, APRIL retains the best filtering effectiveness and total execution time. The performance gap between APRIL and the other methods grows with the difference in sizes between the objects in the candidate pairs, due to APRIL’s effectiveness in detecting true hits, avoiding their costly refinement.

Table 3.20: End-to-end join performance between T2 and datasets with varying average object area.

Method	Accepted	Rejected	Inconclusive	Int. Filter (s)	Refinement (s)	Total time (s)
T2 $\bowtie$ T1						
None	0.00%	0.00%	100.00%	0.00	2.94	2.98
RA (K=750)	21.98%	50.76%	27.26%	1.16	1.20	2.40
5C+CH	0.00%	43.15%	56.85%	0.05	1.65	1.74
APRIL	24.29%	59.42%	16.29%	0.05	0.27	0.35
T2 $\bowtie$ T10						
None	0.00%	0.00%	100.00%	0.00	296.01	297.51
RA (K=750, K=150)	27.49%	25.15%	47.36%	27.61	228.51	257.55
5C+CH	0.00%	29.11%	70.89%	2.28	204.66	208.68
APRIL	51.92%	45.59%	2.50%	2.32	6.46	9.51
T2 $\bowtie$ T3						
None	0.00%	0.00%	100.00%	0.00	310.81	312.38
RA (K=750, K=150)	58.27%	23.15%	18.58%	31.82	102.67	135.79
5C+CH	0.00%	22.03%	77.97%	2.32	224.14	228.40
APRIL	68.47%	29.88%	1.64%	3.22	5.29	9.32
T2 $\bowtie$ T9						
None	0.00%	0.00%	100.00%	0.00	2595.44	2596.96
RA (K=750, K=150)	49.34%	20.26%	30.40%	24.55	1352.44	1378.21
5C+CH	0.00%	21.70%	78.30%	2.36	2003.41	2007.63
APRIL	68.04%	31.80%	0.16%	20.52	3.03	24.46

Table 3.21: TIGER dataset statistics, sorted by ascending average object MBR area.

Dataset	# objects	avg # vertices	avg MBR area
T2 (Water areas)	2252316	31.9	4.03E-05
T1 (Landmarks)	123045	25.4	1.77E-04
T10 (Zip codes)	26091	1404.8	4.14E-02
T3 (Counties)	3043	2316.2	3.95E-01
T9 (States)	43	18140.4	2.59E+01

Table 3.22: APRIL vs. RI (polygonal range queries).

	True hits	True negatives	Indecisive	Int. Filter (s)	Refinement (s)	Total (s)
	x1000 T3 queries against T1					
RI	69.28%	28.60%	2.12%	0.52	0.10	0.64
APRIL	69.27%	28.60%	2.13%	0.06	0.10	0.18
	x1000 T3 queries against T2					
RI	68.46%	29.87%	1.67%	9.26	1.58	11.07
APRIL	68.46%	29.87%	1.67%	1.02	1.58	2.84

3.2.4.4.4 Performance in other queries

We now evaluate the performance of APRIL in other queries, besides spatial intersection joins. We start with selection queries of arbitrary shape (see Section 3.2.1.4). For this experiment, we sampled 1000 polygons from T3 and applied them as selection queries on T1 and T2, simulating queries of the form: find all landmark areas (T1) or water areas (T2) that intersect with a given US county (T3). As Table 3.22 shows, compared to RI, APRIL achieves a 3.5x-4x speedup in the total query cost.

Next, we compare all methods in spatial *within* joins, where the objective is to find pairs (r, s) such that r is within s (see Section 3.2.1.5). As Table 3.23 shows, APRIL again achieves the best performance due to its extremely low filtering cost. APRIL is even faster than 5C+CH, because 5C+CH performs two polygon-in-polygon tests, which are slower compared to a polygon intersection test.

Finally, we test the effectiveness of APRIL in polygon-linestring joins, as described in Section 3.2.1.6. For this experiment, we join the polygon sets T1, T2, and T3 with dataset T8 (from the same collection), which contains 16.9M linestrings (roads in the United States), each having 20.4 vertices on average. In this comparison, we do not include RI and RA because Strong cell types cannot be used to detect true hits. Table 3.24 compares APRIL with 5C+CH and the skipping of an intermediate filter (None). 5C+CH only detects true negatives (in the case where the 5C+CH approximations do not intersect). APRIL outperforms 5C+CH by at least three times in total join time and by orders of magnitude in $T3 \bowtie T8$, where it can identify the great majority of join results as true hits.

Table 3.23: Performance of filters (spatial within joins)

	True hits	True negatives	Indecisive	Int. Filter (s)	Refinement (s)	Total (s)
	T2 $\bowtie$ T1 (Tiger water in landmark areas)					
None	0.00%	0.00%	100.00%	0.00	3.61	3.64
5C+CH	0.00%	34.71%	65.29%	0.10	1.33	1.46
RA	13.48%	29.18%	57.34%	0.14	1.11	1.28
RI	18.48%	59.46%	22.06%	0.20	0.48	0.71
APRIL	18.48%	59.42%	22.11%	0.05	0.49	0.58
	T1 $\bowtie$ T3 (Tiger landmark in county areas)					
None	0.00%	0.00%	100.00%	0.00	20.14	20.19
5C+CH	0.00%	20.72%	79.28%	0.37	14.02	14.44
RA	44.35%	14.29%	41.36%	0.51	8.26	8.82
RI	68.05%	28.13%	3.82%	1.56	0.80	2.41
APRIL	68.05%	28.13%	3.82%	0.21	0.80	1.06
	T2 $\bowtie$ T3 (Tiger water in county areas)					
None	0.00%	0.00%	100.00%	0.00	383.49	384.23
5C+CH	0.00%	22.17%	77.83%	7.70	274.54	282.98
RA	42.50%	15.25%	42.25%	9.53	165.50	175.77
RI	67.36%	29.88%	2.75%	27.08	12.22	40.04
APRIL	67.36%	29.88%	2.75%	3.47	12.22	16.43

Table 3.24: Polygon-linestring spatial intersection joins.

	True hits	True negatives	Indecisive	Int. Filter (s)	Refinement (s)	Total (s)
	T1 $\bowtie$ T8 (Tiger landmarks and roads)					
None	0.00%	0.00%	100.00%	0.00	27.82	28.25
5C+CH	0.00%	45.24%	54.76%	1.07	15.99	17.49
APRIL	12.70%	55.01%	32.29%	0.93	3.82	5.18
	T2 $\bowtie$ T8 (Tiger water areas and roads)					
None	0.00%	0.00%	100.00%	0.00	238.91	241.59
5C+CH	0.00%	68.13%	31.87%	6.24	90.60	99.52
APRIL	0.08%	90.22%	9.71%	5.58	19.92	28.17
	T3 $\bowtie$ T8 (Tiger county areas and roads)					
None	0.00%	0.00%	100.00%	0.00	2546.48	2543.37
5C+CH	0.00%	22.79%	77.21%	16.21	1855.63	1878.73
APRIL	66.25%	30.77%	2.98%	25.64	58.23	90.77

3.3 Scalable Spatial Topology Joins

This section describes our methodology for efficiently determining topological relations. Formally, given two objects r and s whose MBRs intersect (i.e, r and s pass the filter step of spatial join), the *find relation* problem aims to identify their most specific topological relationship. We use the notation C and P to denote APRIL's A - and F -lists [50], corresponding to *Conservative* and *Progressive*, respectively, in order to more clearly reflect the spatial approximation category of each list. We assume that the P and C approximations of r and s have been precomputed. Depending on how the MBRs of r and s intersect, we apply a tailored sequence of binary merge-join operations on P and C interval lists to solve the *find relation* problem without having to compute the DE-9M matrix.

3.3.1 MBR Filter

The possible topological relations between a pair of shapes, r and s , can be constrained based on how their MBRs intersect. For example, consider Figure 3.15(a), where the MBR of r is fully contained within the MBR of s . From this, we can infer that r and s cannot be equal, because if they were equal, their MBRs would also be equal. Additionally, r cannot contain or cover s , eliminating 3 out of the 8 possible relations. Figure 3.15 illustrates, for each type of MBR intersection, the possible topological relations between the corresponding objects. Relations marked in bold font are certain to hold. A notable case is Figure 3.15(d), where the only valid relation is *intersects*, and no refinement step is necessary to specialize it.

3.3.2 Intermediate Filter

Our enhanced MBR filter (Sec. 3.3.1) allows for more specialized handling in the intermediate filter and refinement. A pair of objects with intersecting MBRs is forwarded to the appropriate intermediate filter that focuses only on the possible relations, minimizing the computations that need to be performed. The intermediate filters perform merge-join operations on the objects' sorted C - and/or P -lists based on the candidate relations. We present the workflows of the intermediate filters using the following relations between two *interval* lists X and Y taken from $\{r_P, r_C, s_P, s_C\}$.

' X, Y overlap' At least one pair of intervals $x \in X, y \in Y$ intersect; i.e. x and y include

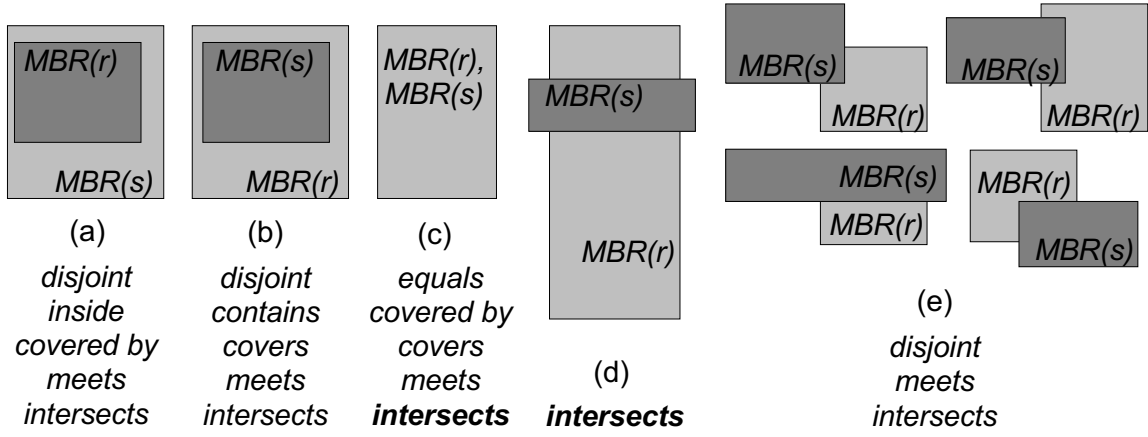


Figure 3.15: The candidate and definite (bold) topological relations of two polygons, based on how their MBRs intersect.

at least one common cell identifier.

‘ X, Y match’ The two interval lists X and Y are identical; i.e., for each $x \in X$ there is a $y \in Y$, such that $x = y$ and vice versa.

‘ X inside Y ’ Every interval in X is contained in one interval of Y ; i.e., for each $x = [x_{start}, x_{end}) \in X$, there exists an interval $y = [y_{start}, y_{end}) \in Y$, such that $x_{start} \geq y_{start} \wedge x_{end} \leq y_{end}$.

‘ X contains Y ’ is defined inversely to ‘ X inside Y ’; every interval in Y is contained in one interval of X .

Note that each of the above relations takes linear time to evaluate, as the intervals within each list are disjoint. Moreover, the number of intervals per list is expected to be low (in the order of square root of the number of cells that intersect with the object) [50].

Algorithm 3.4 describes our proposed approach for solving *find relation* problems using MBRs and C/P raster approximations. The algorithm only handles candidates whose MBRs intersect (line 2). Depending on *how* the MBRs of r and s intersect, one of the *individual intermediate filters* (IF), shown as flow diagrams in Figures 3.16 and 3.17 is applied. Each IF returns either the most specific relation, in which case a *refinement step* that would compute the DE-9IM matrix is unnecessary, or a set of possible relations. In the latter case, the DE-9IM matrix is computed and then compared against the masks of possible relations in a specific-to-general order, to find the most specific topological relation between r and s . We now elaborate on each MBR intersection case.

Algorithm 3.4 Evaluation of *find relation*.

Require: r, s

```
1: function MBRFILTER( $r, s$ )
2:   if  $MBR(r), MBR(s)$  are disjoint then
3:     return disjoint
4:   else if  $MBR(r) = MBR(s)$  then
5:      $result \leftarrow$  IFEQUALS( $r, s$ )
6:     if  $result = coveredby, covers$  then
7:       return  $result$ 
8:     else
9:       return Refine(covers, coveredby, intersects)
10:    end if
11:  else if  $MBR(r)$  inside  $MBR(s)$  then
12:     $result \leftarrow$  IFINSIDE( $r, s$ )
13:    if  $result = disjoint, inside, intersects$  then
14:      return  $result$ 
15:    else if  $result = ref_{inside}$  then
16:      return Refine(inside, coveredby, intersects)
17:    else
18:      return Refine(disjoint, inside, coveredby, meets, intersects)
19:    end if
20:  else if  $MBR(r)$  contains  $MBR(s)$  then
21:     $result \leftarrow$  IFCONTAINS( $r, s$ )
22:    if  $result = disjoint, contains, intersects$  then
23:      return  $result$ 
24:    else if  $result = ref_{contains}$  then
25:      return Refine(contains, covers, intersects)
26:    else
27:      return Refine(disjoint, contains, covers, meets, intersects)
28:    end if
29:  else if  $MBR(r), MBR(s)$  cross then
30:    return intersects
31:  else
32:     $result \leftarrow$  IFINTERSECT( $r, s$ )
33:    if  $result = disjoint, intersects$  then
34:      return  $result$ 
35:    else
36:      return Refine(disjoint, meets, intersects)
37:    end if
38:  end if
39: end function
```

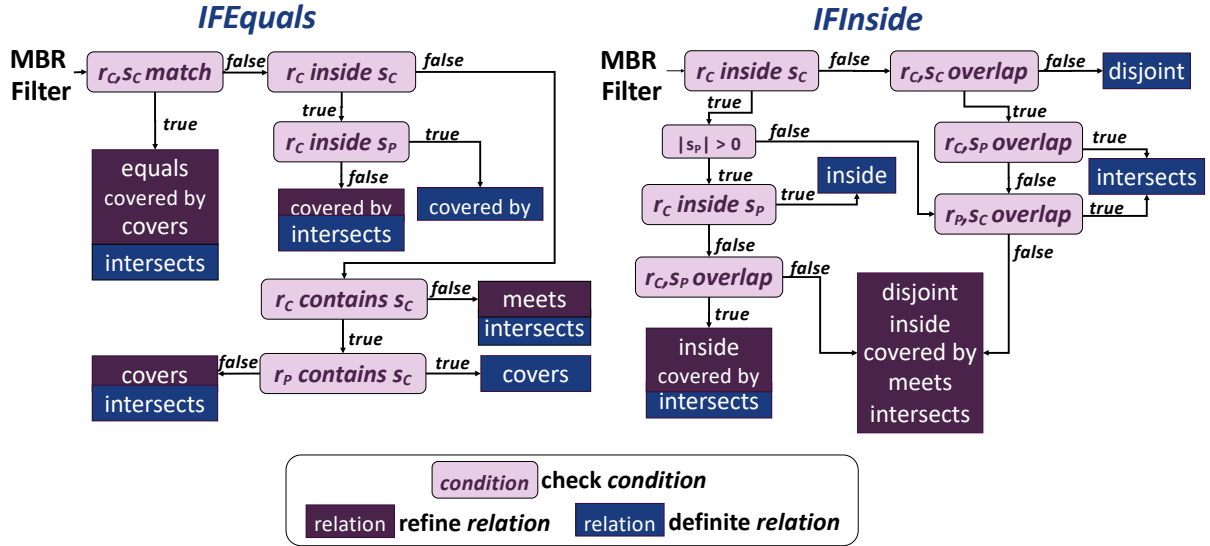


Figure 3.16: The intermediate filters for different cases of intersections between the MBRs of r and s (Part 1).

Equal MBRs If the MBRs are equal (Figure 3.15(c)), then the pair is forwarded to the *IFEquals* intermediate filter, shown first in Figure 3.16. This filter is able to detect exactly, i.e., without refinement, the *covers* and *covered by* relations. If the C lists of the two objects are identical, then the pair is forwarded to refinement for the most specific of *equals*, *covered by*, *covers*, and *intersects*. Although the objects definitely *intersect*, this might not be their most specific relation. If the C lists do not match, the algorithm proceeds to check all possible relations. In general, if the most specific relation cannot be detected from the C and P lists of the two objects, the pair is forwarded to *selective refinement*, where the not definite relations are verified after the DE-9IM matrix computation.

One MBR inside the other When one of the MBRs is contained inside the other (Figure 3.15(a) or 3.15(b)), Algorithm 3.4 forwards the candidate pair to *IFInside* or *IFContains* intermediate filters, respectively, based on which MBR is inside the other. The two filters work in the same way, with the difference being that the first one looks for r inside s or r covered by s , whereas the second looks for r contains s or r covers s , besides *disjoint*, *meets*, and *intersect*. The check to determine whether an interval list r_C is completely contained in a list s_P has $O(|r_C| + |s_P|)$ time complexity.

Other cases If the two MBRs cross each other, as in Figure 3.15(d), we definitely know that the most specific relation is *intersects* and no intermediate filter or refinement is necessary. All other cases, shown in Figure 3.15(e) are handled by the *IFIntersect*

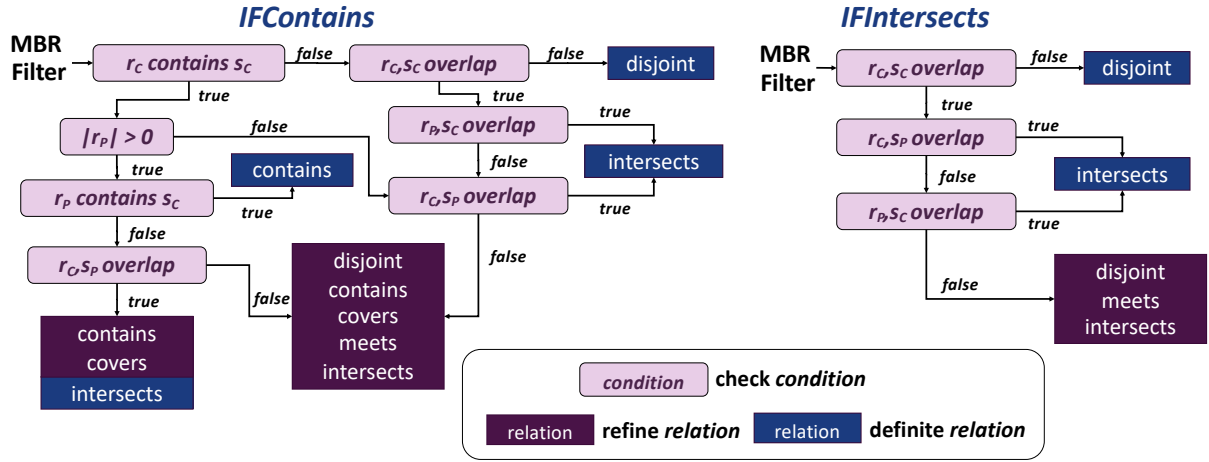


Figure 3.17: The intermediate filters for different cases of intersections between the MBRs of r and s (Part 2).

intermediate filter, which detects *disjoint*, *meets* and *intersects* relations.

Complexity of Algorithm 3.4 Up to four relations between the P and C interval lists of r and s are evaluated in each of the flow diagrams of Figure 3.15, until we can conclude about the topological relation(s) between r and s . For each relation, the cost is linear to the lengths of the two merge-joined lists, because the intervals in a list are disjoint to each other. Hence, the overall cost of the IF applied on a given pair of objects r and s is $O(|r_P| + |r_C| + |s_P| + |s_C|)$.

3.3.3 Relate

In certain scenarios, instead of looking for the most specific topological relation between two objects, we need to quickly check if a given relation is satisfied. For example, spatial joins may take a topological relation as a predicate. Formally, given a pair (r, s) of objects such that $MBR(r)$ intersects $MBR(s)$ and a topological relation p (e.g., $p=\text{meets}$), the $relate_p$ problem finds whether r and s satisfy p . Figure 3.18 (left) shows the intermediate filter's flow diagrams for *relate* predicates that are specializations of *intersects*. The figure is self-explanatory; for a given predicate a sequence of merge-join operations are applied on the C and/or P lists to potentially verify if the corresponding relation definitely holds or not. For instance, for a pair (r, s) of objects, if not r_C inside s_C , then definitely r is not inside s ; if r_C inside s_P , then definitely r is inside s . In all other cases, (r, s) is sent to the refinement step.

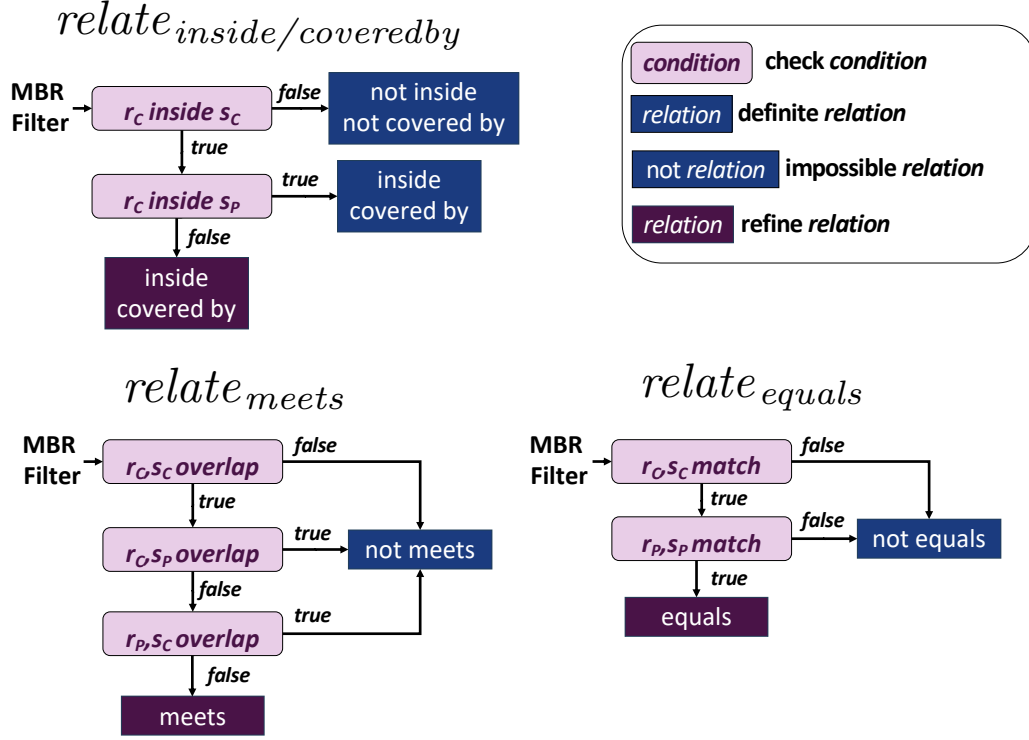


Figure 3.18: Intermediate filter flow diagrams for $relate_p$ tests.

3.3.4 Experimental Analysis

We evaluated the performance of our topological relation detection approach, compared to the current practice and the use of intermediate filters for intersection detection [50]. We compare the following approaches in a single-machine environment without thread parallelism:

ST2: standard 2-phase This method applies the current MBR filter+refinement practice used in most previous work that detect topological relations [35, 11]. If the object MBRs do not intersect, then the pair is disjoint, otherwise we compute the DE-9IM matrix to determine the topological relation.

OP2: optimized 2-phase This method uses the relation between the MBRs, as described in Sec. 3.3.1 to limit the possible topological relations and, in turn, reduce the number of DE-9IM masks to compare with the DE-9IM matrix.

APRIL: optimized MBR filter + APRIL intermediate filter + refinement This method applies APRIL [50] to detect non-intersection between two objects whose MBRs intersect, before applying the refinement step. As the intermediate filter in this case cannot detect more special relations than *intersects*, the DE-9IM matrix computation is necessary even for pairs that are found to definitely intersect, to detect a

Table 3.25: Description of datasets.

Dataset	Entity type	# polygons	Size (MB)	MBRs (MB)	$P + C$ (MB)
TL	US Landmarks	123K	52.5	3.7	6.3
TW	US Water areas	2.25M	1.2K	68.6	73.0
TC	US Counties	3.04K	112.8	0.1	15.4
TZ	US Zip Codes	26.1K	587	0.8	170.1
OBE	EU Buildings	90.4M	10.9K	2.8K	2.3K
OLE	EU Lakes	1.96M	1.1K	59.8	82.0
OPE	EU Parks	7.17M	3.7K	218.8	389.0
OBN	NA Buildings	9.38M	1.3K	286.3	201.0
OLN	NA Lakes	4.02M	2.5K	122.7	133.0
OPN	NA Parks	999K	767.4	30.5	60.0

potentially more specific relation.

P+C: optimized MBR filter + *Progressive/Conservative* filter + refinement This is our algorithm, presented in Section 3.3.2.

3.3.4.1 Datasets & Setup

We used popular benchmarking datasets from the TIGER 2015 and Open Street Map (OSM) collections [51], containing real-world areas from the USA and the entire globe, respectively. As the *find relation* problem makes sense for objects that exist in the same region, we also split each OSM dataset to two parts, one for Europe and one for North America (the continents with the highest concentration of objects). Additionally, we cropped the TIGER datasets to include only entities in the contiguous (lower 48) United States. Table 3.25 describes the datasets that we used, and the space occupied by their polygons and their approximations. T- and O- prefixes of datasets denote TIGER and OSM collections, respectively.

Table 3.26 shows dataset combinations that we used in our experiments. For each combination, the table also shows the number of object pairs that pass the MBR filter. The dataset pairs represent real-world scenarios where the detection of topological relations between areas is meaningful:

- TL-TW: US landmarks (TL) may include parks, lakes, canyons, buildings, etc.,

Table 3.26: Semantically meaningful dataset combinations for the *find relation* and *relate_p* problems.

Datasets	TL-TW	TL-TC	TC-TZ	OLE-OPE	OLN-OPN	OBE-OPE	OBN-OPN
Candidate pairs	63.3K	168K	65.7K	5.18M	2.77M	79.3M	2.17M

so geospatially interlinking pairs from TL and TW may reveal unexpected relations and statistics about all landmark and water areas in the US.

- TL-TC: As US counties (TC) are relatively large areas, it is expected that most of the relations in this scenario will be *inside*. However, unique cases can be found, such as cross-county landmarks or even landmarks containing entire counties.
- TC-TZ: This pair provides insights about the county and zip code relations in the US.
- OBx-OPx(E/N): Topological relations in this scenario may provide insight on the amount of human intervention (buildings) in green areas (parks) in Europe and North America.
- OLx-OPx(E/N): Interlinking entities between these datasets provides information about the presence of water elements (currents, lakes, marshes etc.) in the parks of Europe and North America.

For each of the scenarios above, we run a spatial intersection join algorithm [27] that produces the pairs of objects whose MBRs intersect. Each such pair was fed to our topological relation detection pipeline. For each of the evaluated methods, we measured their *throughput*; i.e., the number of MBR-filtered pairs that the method processed per second. We did not account for the cost of the filter step (i.e., the time to produce the pairs of MBRs that intersect), which is negligible compared to the cost of identifying the topological relation for each such pair.

All raster approximations that we used for the experiments, were created using independent $2^{16} \times 2^{16}$ grids overlayed on each data scenario’s dataspace. As shown in Table 3.25, the *Progressive* and *Conservative* interval lists occupy much less space compared to the polygons. For the refinement, we use the boost geometry’s *relation* function [56] that calculates the DE-9IM matrix for two input geometries (boost’s

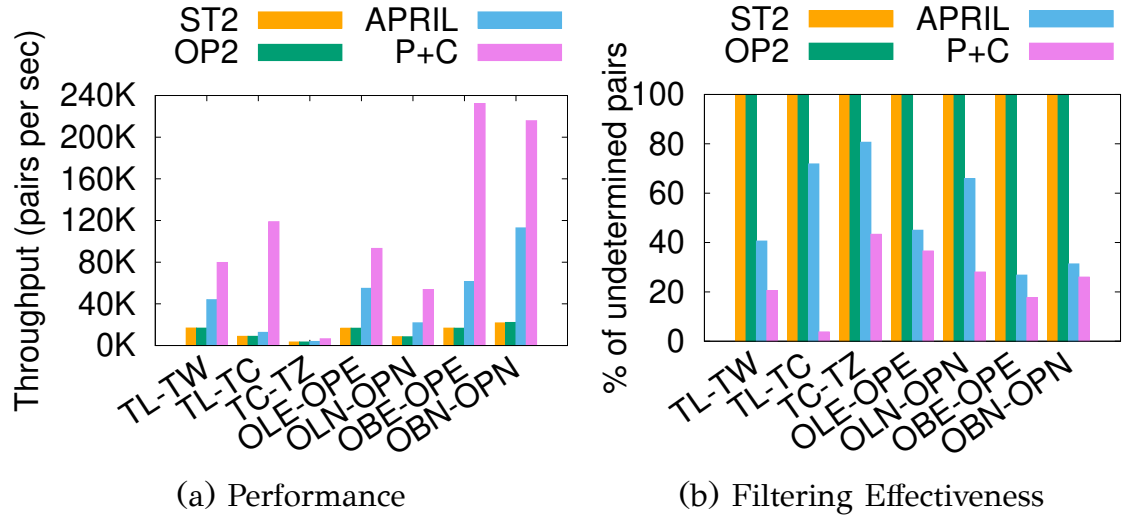


Figure 3.19: Throughput and Effectiveness for *find relation*.

implementation was found to be much faster than GEOS [46]). We ran our experiments on a machine with a 3.6GHz Intel i9-10850k and 64GB of RAM, running Linux. The code was written in C++ and compiled with the -O3 flag.

3.3.4.2 Performance

Figure 3.19(a) shows the throughput of the compared methods on the test dataset pairs. Observe that, in practice, there is no improvement when using an optimized MBR filter (OP2), compared to directly moving the pair to refinement (ST2). This is because the computation of the DE-9IM matrix is the bottleneck of the entire process, so the savings of OP2 by reducing the number of masks (relations) that are being checked are marginal. Using the APRIL intermediate filter greatly improves the throughput, by detecting many cases of object pairs that are disjoint. Our P+C method (Sec. 3.3) guides the comparison between the *C* and *P* lists of the objects, and detects more topological relations than APRIL (on top of intersects [50]). Overall, our P+C approach improves the throughput of *find relation* by one order of magnitude compared to the current practice (ST2, OP2) [35, 11] and by a few to several times compared to using the APRIL intermediate filter [50].

The improvement in the throughput when transitioning from 2 stages in the pipeline (ST2, OP2) to 3 (APRIL, P+C) is related to the effectiveness of the intermediate filters. For each pair of datasets, we measure filter effectiveness in terms of undetermined pairs, i.e. pairs that need to be refined using DE-9IM to detect their

Table 3.27: OLE-OPE post-MBR filter polygon pairs, grouped by complexity level (sum of their vertex count).

Complexity level	Sum of vertices	Pair count
1	[8,41]	525K
2	[42,67]	518K
3	[68,104]	513K
4	[105,163]	520K
5	[164,265]	520K
6	[266,447]	517K
7	[448,786]	518K
8	[787,1354]	518K
9	[1355,2629]	518K
10	[2630,60398]	518K

topological relation. As shown in Figure 3.19(b), the introduction of the APRIL intermediate filter reduces the DE-9IM matrix computations to around half compared to ST2, OP2, on average. Our specialized intermediate filter workflows (Sec. 3.3) reduce the object pairs to be refined even more, to about 25% on average.

3.3.4.3 Scalability

We tested the scalability of our approach (P+C) with respect to the complexities of the object pairs whose topological relation is being detected. For a pair of polygons r and s , we define complexity as the sum of their vertices, as this determines the cost of computing intersections between their parts [48], and in turn, the cost of DE-9IM matrix computations. We selected a fairly large scenario (OLE-OPE) and divided its post-MBR candidate pairs into 10 groups of increasing complexity, shown in Table 3.27. The ranges were selected, such that each complexity level contains around the same number of object-pairs; hence, we test the scalability of our method on similar data workloads per complexity level.

Figure 3.20(a) shows P+C’s intermediate filter effectiveness in detecting relations without DE-9IM matrix computations for polygon pairs from the OLE-OPE scenario, having different complexities. High complexity pairs are resolved more frequently by the intermediate filters of P+C compared to low complexity pairs. This is due

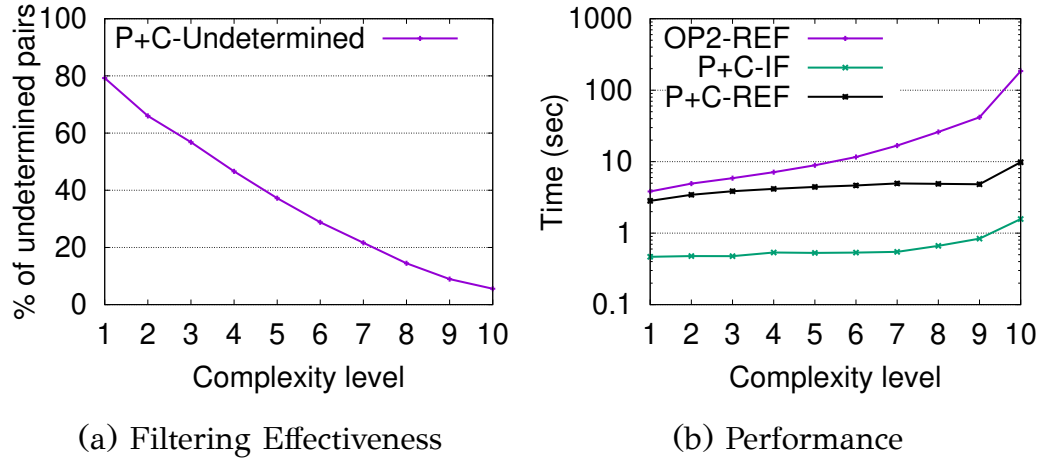


Figure 3.20: *Find relation* filter effectiveness and total cost for polygons pairs grouped by complexity level.

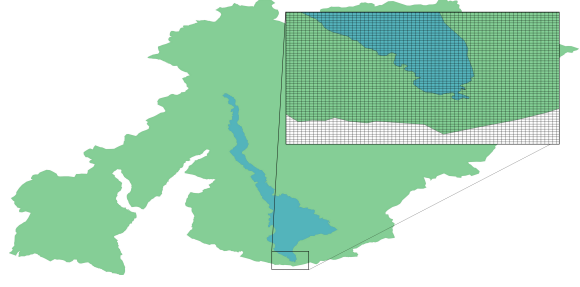
to the fact that most of the intermediate filters of P+C utilize the polygons' P lists to detect early definite topological relations. Raster-based interval approximations of small polygons usually result in very few, if any at all, full cells and progressive intervals, rendering refinement necessary to detect their topological relation. As shown in Figure 3.20(a), 4 out of 5 pairs of complexity level 1 in the OLE-OPE scenario need to be geometrically refined, contrary to pairs of complexity level 10 where only around 5% are refined.

Polygon pairs of low complexity are cheap to refine and, as complexity increases, the refinement becomes more expensive, with polygons of complexity level 10 being the bottleneck of the entire pipeline. This is reflected in Figure 3.20(b), where the cost of OP2 which refines almost all pairs increases superlinearly with the complexity level. On the other hand, the cost of P+C is almost insensitive to the complexity of the pairs due to the fact that the increase in the refinement per pair is compensated by the decrease of the number of pairs that need to be refined, as seen in Figure 3.20(a). This experiment unveils an important value of our approach, which achieves its highest filtering power on objects that have the largest need for it. Besides saving computations, our approach also manages to load less data compared to OP2. For example, in the OLE-OPE scenario, the P+C approach accesses only 48.5% of the unique objects from OLE and OPE that are accessed by OP2, as the topological relations involving the remaining objects are detected by the P+C intermediate filters.

Figure 3.21 showcases a pair of objects at complexity level 10, where their relation (*inside*) is identified by the P+C intermediate filter and the DE-9IM matrix compu-

	Lake	Park
Vertices	2240	2616
MBR area	0.0616	0.5030
<i>C</i> -intervals	498	1793
<i>P</i> -intervals	481	1845

(a) Statistics



(b) Illustration

Figure 3.21: A level-10 complexity pair of a lake (blue) residing inside a park (green), from the OLE-OPE scenario.

tation is avoided. This pair is forwarded to refinement by ST2, OP2, and APRIL, as these methods cannot detect the most specific relation (APRIL would only be able to detect intersection). As a result, our P+C approach is 50x faster than the other methods for this pair. The pair represents a lake residing inside a park; even though the park is a lot bigger than the lake, they both have an adequate amount of P intervals, due to the fine-grained grid (2^{16} cells per dimension) that was used to create their approximations.

3.3.4.4 *Relate* Performance

In the last experiment, we compare the effectiveness of our $relate_p$ algorithms for specific topological predicates p (Section 3.3.3) against our general $find\ relation$ algorithm (Section 3.3.2). We chose the OLE-OPE scenario because of the two datasets' relative balance in terms of polygon size and complexity. For three different predicates p , Table 3.28 shows that the throughput for $find\ relation$ is independent of p , as the algorithm does not consider p . On the other hand, the throughput of $relate_p$ queries is sensitive to the predicate, as a different intermediate filter is used in each case. The more focused $relate_p$ approach is faster in all tested cases, due to the specialized steps it takes, depending on p . The difference is huge for some predicates (e.g., *meets*), where non-satisfaction is fairly easy to identify using the object approximations.

Table 3.28: Throughput (pairs/sec) of *find relation* and *relate_p* methods using our P+C approach (OLE-OPE).

Method	Equals	Meets	Inside
find relation	93160.2	93160.2	93160.2
<i>relate_p</i>	107265.6	7000989.2	565509.7

3.4 Conclusions

In this chapter, we proposed a technique (RI and its extension, APRIL) that represents 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, the first version of our approach, approximates each object as a single list of intervals that include the raster cells that intersect the object; together with each interval we store a bitstring that encodes the classes of cells (Full, Strong, Weak) in the interval. APRIL is an enhanced version of our method that captures the cells that are partially or fully covered by the object using two separate lists of intervals, eliminating the need for the space-consuming and complex bitstring. APRIL’s intermediate filter is different from that of RI in that it performs a pipeline of three interval joins instead of a single interval join paired with bitwise operations on the bitstrings.

As we have shown experimentally, compared to previous approaches [1, 28], APRIL is (i) lightweight, as it represents each polygon by two lists of integers that can be effectively compressed; (ii) effective, as it typically filters the majority of MBR-join pairs as true negatives or true positives; and (iii) efficient to apply, as it only requires at most three linear scans over the interval lists. Specifically, RI and APRIL offer at least 3x speedup in end-to-end spatial intersection joins compared to previous intermediate filters (raster approximations [28], 5C-CH [1]). At the same time, the space complexity of RI and APRIL is relatively low and the approximations can easily be accommodated in main memory. Compared to RI approximations, APRIL approximations are much cheaper to construct, occupy significantly less space, offer a much faster intermediate filter, and significantly improve the end-to-end cost of spatial intersection joins.

APRIL is a general approximation for polygons that can also be used in selection queries, within-joins, and joins between polygons and linestrings. We propose a compression technique for APRIL and customizations that trade space for filter effec-

tiveness. Finally, we propose an efficient construction technique for APRIL approximations that is orders of magnitude faster than the rasterization-based techniques used for other filters.

Additionally, we presented a scalable technique for detecting topological relations between complex spatial objects (i.e., polygons) by leveraging raster approximations to minimize expensive DE-9IM matrix computations. Our approach not only reduces the number of object pairs requiring refinement but also becomes increasingly effective as object complexity (and thus refinement cost) grows.

CHAPTER 4

HECATONCHEIR: SCALING UP AND OUT SPATIAL DATA MANAGEMENT

4.1 Architecture

4.2 Experimental Evaluation

4.3 Conclusions

In this chapter, we present *Hecatoncheir*, a research prototype for distributed spatial data management that integrates raster interval approximations and their specialized query evaluation pipelines from our previous work. Hecatoncheir supports beyond-join query types including k-nearest neighbor (kNN), range, and distance joins, across all major spatial data types: points, linestrings, and polygons. Implemented in C++ and built on MPI, Hecatoncheir is designed for efficient and scalable query evaluation across distributed nodes, while at the same time parallelizing computationally intensive tasks on the CPU when possible. Its black-box architecture and lightweight C++ API ensure ease of deployment and use, with no reliance on underlying database engines. Hecatoncheir is particularly well-suited for tightly coupled cluster environments, and experimental results demonstrate its scalability, high performance, and memory efficiency in comparison to Apache Sedona.

Outline In Section 4.1, we provide an overview of Hecatoncheir’s architecture and describe in-depth its individual components and layers. In Section 4.2 we experimentally evaluate Hecatoncheir’s performance, scalability and usability using real-world

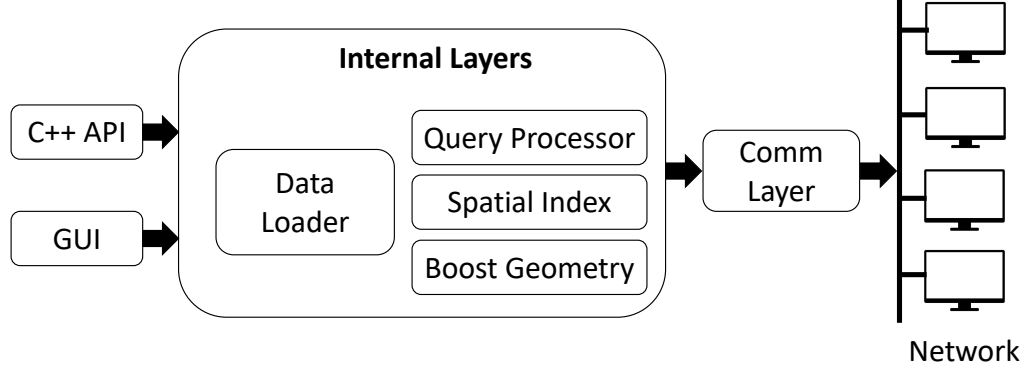


Figure 4.1: Hecatoncheir architecture overview.

data in spatial query scenarios. We additionally compare Hecatoncheir against Apache Sedona, highlighting their differences in terms of design and performance. Section 4.3 summarizes our conclusions.

4.1 Architecture

Hecatoncheir’s architecture is summarized in Figure 4.1. The MPI Layer is where all inter- and intra- process communication takes place. Internally, the Query Processor uses the Index Layer [57] for spatial queries, the results of which are potentially refined using Boost Geometry (boost.org). The API and the GUI are the entry points to the system’s functionalities for the user.

4.1.1 Data Loading and Distribution

The Host partitions user data across Workers using a coarse-grained spatial grid, whose tiles are assigned to Workers in a round-robin manner [49]. An object is sent to the Worker responsible for any tile its minimum bounding rectangle (MBR) overlaps. This coarse grid acts as a global spatial index, and its granularity (e.g., 100×100 for 10 Workers) ensures more tiles than Workers, promoting load balancing and limiting fragmentation of nearby objects, which is important for distance-based queries. An illustration for a (global) dataset is shown in Figure 4.2 (bold lines).

Each Worker stores and indexes its assigned tiles using a much finer grid, with granularity set so that the number of cells is divisible by the number of threads, facilitating intra-node parallelism during query evaluation. Both grid granularities can be set manually or optimized automatically based on data distribution.

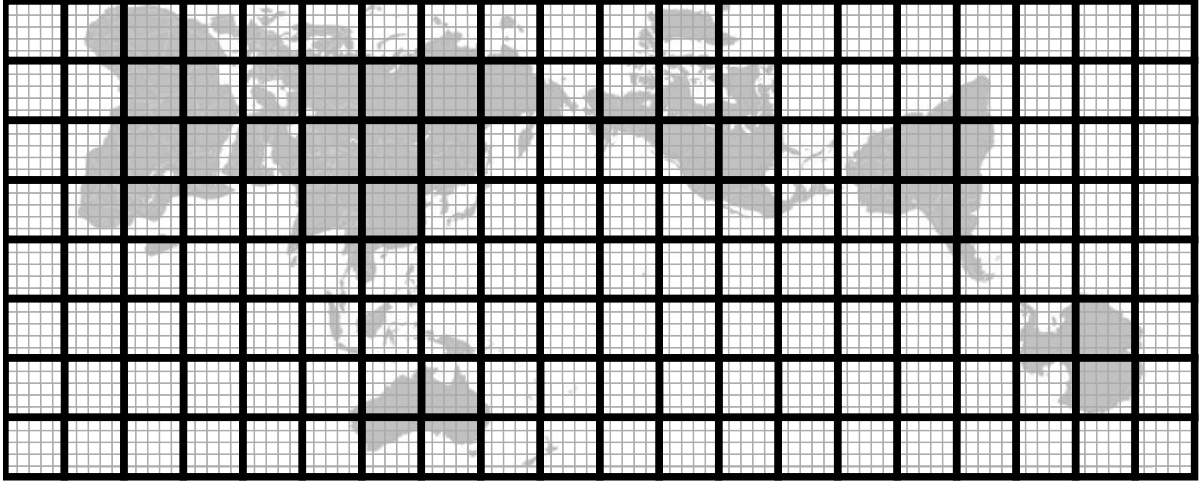


Figure 4.2: Spatial partitioning using a coarse grid.

The Host is solely responsible for partitioning the data across Workers, which introduces considerable overhead. To mitigate this, if supported by the storage medium (e.g., SSD, NVMe), the Host spawns threads to parallelize both data reading and distribution. Each thread performs its own MPI calls, sending object batches to Workers based on a coarse partitioning grid. The batch size is configurable to (i) avoid exhausting Host memory and (ii) allow Workers to begin processing before the entire dataset is read. Since each thread maintains a batch per Worker, the Host’s memory usage is approximately $num_{threads} \times num_{workers} \times batch_{size}$, plus additional space for in-memory partitioned data. Hence, batch size must be chosen with memory limits and input size in mind.

4.1.2 MPI Layer

Communications in Hecatoncheir are illustrated in Figure 4.3. All machines in the cluster are connected via (passwordless) SSH within the same network, though they may or may not also be physically interconnected. The user provides a program, referred to as the Driver process, which communicates exclusively with the Host, either through the C++ MPI API or via the GUI. The Driver runs as a separate process on the user’s machine (or, alternatively, on one of the cluster machines).

Each machine runs a single Worker process. On the Host machine, an additional process called the Host, is responsible for coordinating Workers and communicating with the Driver. The Host handles message passing but performs no CPU-intensive computations. Upon receiving requests from the Driver, the Host propagates them

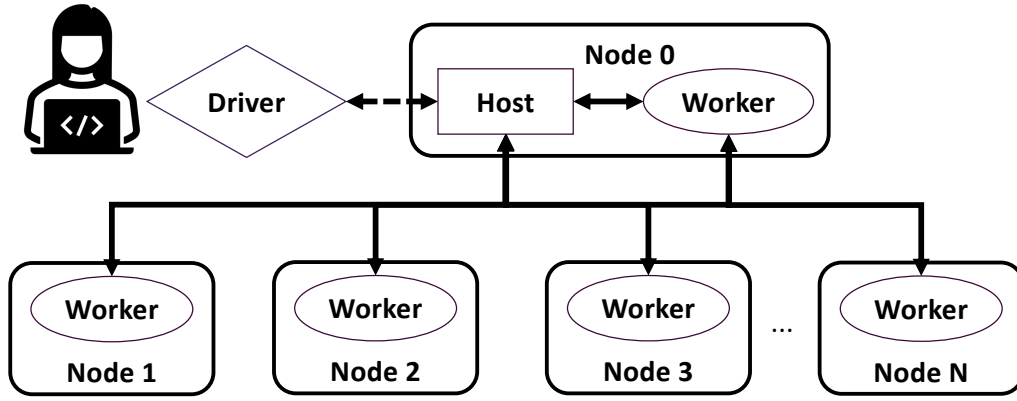


Figure 4.3: Communication setup.

to the Workers and, when necessary, returns results or status updates to the Driver. Communication between the Driver and the Host is carried out through a dedicated MPI communicator

The Workers, in contrast, carry out all computationally intensive operations, leveraging the available threads on their respective machines to parallelize assigned tasks. In certain cases, Workers may need to exchange information directly (e.g., distance queries). Such communication is supported but carefully optimized to remain minimal during query evaluation, thereby reducing the overhead typically associated with inter-node communication

4.1.3 Internal Layers

Hecatoncheir's internal layers are hidden from the user, and the MPI layer is not considered part of them. All system operations are accessed exclusively through the C++ API. Although the Driver communicates with the Host using MPI, this interaction is fully abstracted behind the API, eliminating the need for users to write MPI calls and significantly reducing usage complexity.

4.1.3.1 Boost Geometry

Boost Geometry offers a wide variety of geometric operations and data structures, which can be used to post-process objects that pass the filter step of spatial queries on complex object geometries, such as range selections and intersection joins. Hence, for objects or object pairs that pass the MBR-based filter step of such queries, the Workers access their geometries and use the Boost Geometry library for the refinement step.

Hecatoncheir defines and employs a custom wrapper around Boost Geometry’s shape concepts, encapsulating relevant metadata such as object identifiers, approximations (MBR, APRIL), and data types, within the wrapper class. Consequently, Hecatoncheir’s refinement capabilities rely on the operations and cross-type support provided by Boost Geometry.

4.1.3.2 Index Layer

During data loading, each Worker re-partitions its local data by a fine-grained uniform grid. Non-point geometries are further indexed using Two-Layer partitioning [57]; in each fine cell, the objects are divided into four classes, depending on whether the bottom-left point of their MBR begins (A) inside that cell, (B) in a previous cell on Y axis, (C) in a previous cell on X axis, or (D) in a previous cell on both axes. Object MBRs that overlap with more than one cell are replicated and classified. Fine-grid cells indexed by a Worker w store only the objects assigned to w and are not visible to other workers. The local indices are stored in the main memories of the Workers. Figure 4.4 illustrates an example index for a coarse grid partition P and a worker W , where the fine grid has four times the granularity of the coarse grid i.e., each coarse grid cell contains 16 fine grid cells. Objects o_1 (linestring), o_2 (polygon) and o_3 (polygon) are further indexed using the Two-Layer index classes in W ’s index, whilst points o_4 , o_5 and o_6 are simply assigned to their respective fine partitions. Note that object o_3 will also be assigned to the Worker responsible for the next coarse grid partition in row $P + 1$, as it overlaps with that partition as well. Consequently, it will be assigned class C by the Worker responsible for $P + 1$, not by W . In addition, each Worker computes and stores a raster interval approximation [50] for each object assigned to it. The granularity of the fine grid is in the order of a thousand partitions for each dimension. Contrary to tree-based indexes, Hecatoncheir’s grid-based partitioning and indexing facilitate both fast data updates and parallel query evaluation.

4.1.3.3 Query Processing Layer

Query processing takes place after Hecatoncheir has been initialized successfully and all data has been already partitioned to and indexed by the Workers. Since our query processing algorithms do not require re-partitioning the input data, the preprocessing cost of data files is one-off. Queries are evaluated in parallel by each Worker using

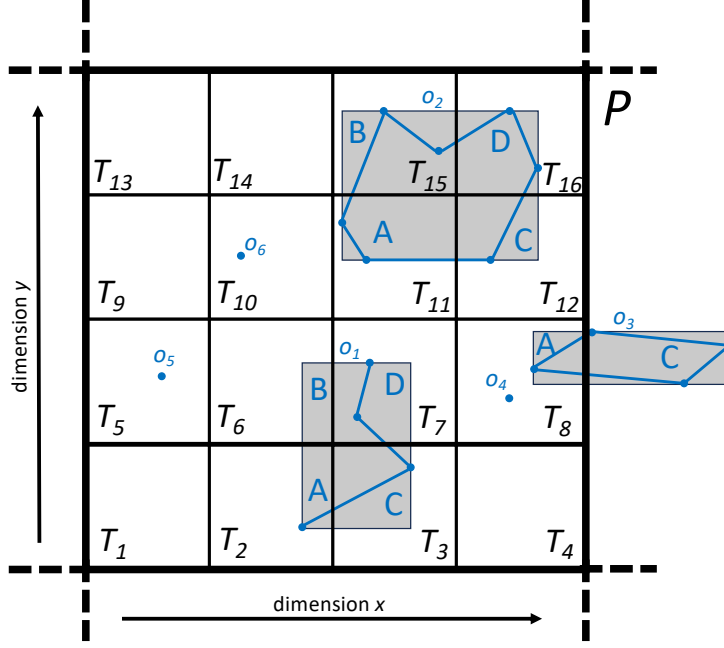


Figure 4.4: The Two-Layer index on a Worker W 's local fine-grained grid.

their local (fine-grid) index. The results are then collected by the Host and returned to the Driver. The following queries are currently supported by Hecatoncheir:

Range Queries (Polygonal or Box) The user (Driver) submits a range query or a batch of range queries to the Host. The Host determines which partitions of the coarse grid overlap with the query window to identify the Workers responsible for evaluating it and sends the query to these Workers. Range queries on non-point data are evaluated using the Two-Layer partitioning scheme [57] that identifies candidates fast using their MBRs and does not generate duplicate results. For point data, a simple, equally fine-grained uniform grid is used to prune non-candidates and no refinement is required for rectangle query windows. However, for polygonal query windows, refinement must be performed for each candidate object to verify whether the geometry is actually intersecting the query range using Boost Geometry. For non-point data, to avoid refining all MBR-filter candidates, the Worker performs further filtering stage by accessing their raster interval approximations [50] and applying the query to those, before forwarding any indecisive cases to the refinement. The contents of cells that are fully covered by the query range are automatically returned as results, without any comparisons. For a query batch, the Host distributes the queries to Workers in sub-batches and each Worker evaluates its sub-batch in-parallel using its local threads. Parallel evaluation is implemented with OpenMP, assigning each thread

to a separate range query.

Topological Joins Spatial join queries are initially broadcast to all Workers. Based on the predicate, Workers use their local indexes to perform the join between the already partitioned and indexed datasets. Supported spatial predicates for joins include: *Equals*, *Disjoint*, *Intersects*, *Inside*, *Contains*, *Covered by*, *Covers* and *Meets*. The system computes the result of *Disjoint* implicitly as the complement of *Intersects*, which is faster to derive. Hecatoncheir also supports a special predicate *Find Relation* that identifies the exact topological relation of all non-disjoint pairs between the datasets. The partitioning of objects within cells to classes [57] accelerates the filter step of the spatial join. The pre-computed raster interval approximations of objects [127] further reduce the number of pairs for which the computationally expensive refinement step is applied (using Boost Geometry). Each Worker performs the join on its own partitions independently, using OpenMP to parallelize the processing across threads. Since the partitions are independent, their contents can be processed simultaneously.

Distance queries Hecatoncheir supports distance spatial queries, including ϵ -range queries (given a query point q , retrieve all points with distance at most ϵ to q), k NN queries (find the k nearest neighbors to query point q), and ϵ -distance joins (given two input datasets, find the pairs of objects in them having distance at most ϵ to each other). For ϵ -range queries and k NN queries, processing of each query is done independently at each Worker which may include query results and the results are aggregated by the Host. For instance, k NN queries near a coarse-grid boundary require neighboring (W) Workers to compute their results (Wk total objects), which are then reported to and refined by the Host. For distance joins, the Workers exchange data at the borders of the coarse grid to facilitate correct and duplicate-free join computation [128]. To optimize communication and minimize data transfer, each Worker first notifies its neighboring Workers of the number of objects it intends to send, specifically, the number of points located within a distance of ϵ from their shared borders. When data exchange is required between two nodes, only the Worker with fewer objects to send initiates the transfer, while the other node receives the data and evaluates it against its own. Additionally, Hecatoncheir supports specialized indexing of non-point geometries to facilitate fast and duplicate-free filtering in distance-joins [129].

4.1.4 API & Interface

Hecatoncheir's C++ API (header-based) facilitates access to the system's features, without burdening the user with low-level implementation details. The Driver initializes Hecatoncheir by calling the appropriate initialization method, specifying the desired number of Workers and their IP addresses. Passwordless SSH connection between the Driver's machine and the Workers must be properly setup. The user then can develop their own program to load and query their datasets with just a few method calls.

4.1.4.1 C++ API

Hecatoncheir's C++ API is exposed to the user through a single C++ header file `Hecatoncheir.h` and all methods are encapsulated in its namespace called `hec`. The first call that the user's driver program must make is the `hec::init(...)`, which takes care of all internal initialization steps automatically. The method's parameters and behavior are defined as follows:

- `int init(int N, vector<string> &IPAddresses)`
 - **Parameters:**
 - * `N`: the number of worker nodes to initialize.
 - * `IPAddresses`: a list containing the IP addresses or the aliases of the nodes to use.
 - **Returns:** An integer status code that indicates success/failure.
 - **Description:** Initializes the system by connecting to the specified worker nodes, spawning the processes and establishing communication channels.

After successful initialization, all Worker nodes remain on standby, awaiting user requests propagated through the Host.

In order to load and query a dataset, the user must first notify the system of certain metadata in order for it to "prepare the ground" for the partitioning and indexing of the data. This is realized by the `prepareDataset(...)` method which is described below:

- `DatasetID prepareDataset(string path, string fileType, string dataType, bool persist)`

– **Parameters:**

- * **path:** the local path to the dataset on disk.
- * **fileType:** the dataset file type. Supported file types include WKT and CSV.
- * **dataType:** the dataset's contents are defined by a spatial data type. Currently, Hecatoncheir expects all objects within a dataset to share the same spatial type. Supported types include points, linestrings, and polygons.
- * **persist:** a flag indicating whether Worker nodes should persist their assigned partitioned data on local disks for future use.

– **Returns:** An integer identifier that uniquely represents the dataset, allowing both the user and Hecatoncheir to reference it.

– **Description:** Prepares the dataset for partitioning and indexing. This includes defining internal structures, generating spatial approximations, and assigning partitions to nodes.

After the preparation of the dataset, the user can initialize the partitioning through the `partition(...)` method which is described below:

- `int partition(vector<DatasetID> &identifiers)`

– **Parameters:**

- * **identifiers:** a list containing the unique identifiers of the datasets to partition.

– **Returns:** An integer status code that indicates success/failure.

– **Description:** Performs the distributed partitioning of the specified datasets across the Worker nodes.

In order to query the partitioned data efficiently, it is mandatory that they are indexed in-memory. Hecatoncheir's API offers the `buildIndex(...)` method that performs the parallel index building across all Worker nodes.

- `int buildIndex(vector<DatasetID> identifiers, IndexType indexType)`

– **Parameters:**

- * **identifiers:** a list containing the unique identifiers of the datasets to index.
- * **indexType:** a value that represents the index to use. Currently, Hecatoncheir supports the uniform grid (for points) and the two-layer indexes (for non-point data).
- **Returns:** An integer status code that indicates success/failure.
- **Description:** Tasks all Worker nodes to construct their local indexes in parallel. The specified index type must be compatible with the dataset's spatial data type.

After these steps, all Worker nodes are standing by ready with all data partitioned and indexed in-memory. The user can now use Hecatoncheir's API to generate and evaluate spatial queries.

The API supports a variety of spatial query types and provides custom structures to define them uniquely. Users first construct a query object representing the query they wish to execute and then pass it to Hecatoncheir for evaluation. The following query classes inherit from a general `hec:Query` class and may share members; however, each derived query class primarily specifies how the query is processed internally.

- `hec::RangeQuery`

- **Members:**

- * **DatasetID identifier:** the identifier that specifies the dataset to be queried with this range query.
- * **string wktText:** the WKT defining the range window. Currently, range queries must be specified using WKT format.
- * **DataType shapeType:** a value that represents the type of the window (rectangle or polygon).
- **Description:** Range (selection) queries specify a rectangular or polygonal window and return all objects that intersect with the window's border or area.

- `hec::PredicateJoinQuery`

- **Members:**

- * `DatasetID identifierR`: the identifier that specifies the left (R) relation in the $R \bowtie S$ spatial join.
 - * `DatasetID identifierS`: the identifier that specifies the right (S) relation in the $R \bowtie S$ spatial join.
 - * (inherited from base `hec::Query` class) `QueryType queryType`: always specifies the type of the query, but in spatial joins it also specifies the predicate for the join.
- **Description:** Spatial joins are always performed between two datasets R and S with a predicate that defines the relation to evaluate for (intersection, adjacency etc.).
- `hec::KNNQuery`
 - **Members:**
 - * `DatasetID identifierR`: the identifier that specifies the left (R) relation in the $R \bowtie S$ spatial join.
 - * `string wktText`: the reference object over which the k NN query will be evaluated. Currently, it must be in WKT format.
 - * `int k`: a positive integer value that specifies the number of nearest neighbors to look for in the query evaluation.
 - * `DataType dataType`: the reference object's spatial data type. Currently, k NN queries expect the reference object to be a point.
 - **Description:** k NN queries are executed on a single dataset and return the k objects closest to the reference object.
 - `hec::DistanceJoinQuery`
 - **Members:**
 - * `DatasetID identifierR`: the identifier that specifies the first relation in the distance join.
 - * `DatasetID identifierS`: the identifier that specifies the second relation in the distance join.
 - * `double distance`: a positive value specifying the maximum distance ϵ for evaluating the distance join. This value must use the same coordinate system as the queried data.

- **Description:** Distance joins are always performed between two datasets using a distance threshold ϵ , and return all pairs of objects that are within ϵ units of each other.

After constructing a query object, the user may use the `hec::query(...)` method described below, to initialize the query evaluation.

- `hec::QResultBase* query(hec::Query* query)`

- **Parameters:**

- * `hec::Query* query`: a derived query-class object from those described above, cast as a pointer to the base `hec::Query` class.

- **Returns:** A reference to a `hec::QResultBase` object that encapsulates the results of the evaluated query. This object provides an interface for convenient retrieval and projection of the results.

- **Description:** Evaluates the spatial query specified by the `hec::Query` derived class object.

For flexibility, Hecatoncheir’s API provides the `hec::unloadDataset(DatasetID identifier)` method, which removes a previously partitioned and indexed dataset from the system’s memory. This allows users to interchange datasets and execute queries on different datasets without needing to terminate or re-initialize the entire system.

Finally, the user can safely terminate a Hecatoncheir instance by calling the `hec::finalize()` method, which frees memory, saves any necessary data to disk, and stops all operations in a structured and safe manner.

4.1.4.2 GUI

Additionally, Hecatoncheir comes with a web-based GUI that enables users to interact with Hecatoncheir without writing any code. The interface abstracts the underlying communication with Hecatoncheir’s API, allowing users to configure and execute spatial queries through an easy-to-use point-and-click workflow. A snapshot of a spatial query execution through Hecatoncheir’s GUI is shown in Figure 4.5.

When launching the GUI, users provide the addresses (or aliases) of their cluster nodes, after which the system automatically manages all initialization steps internally.

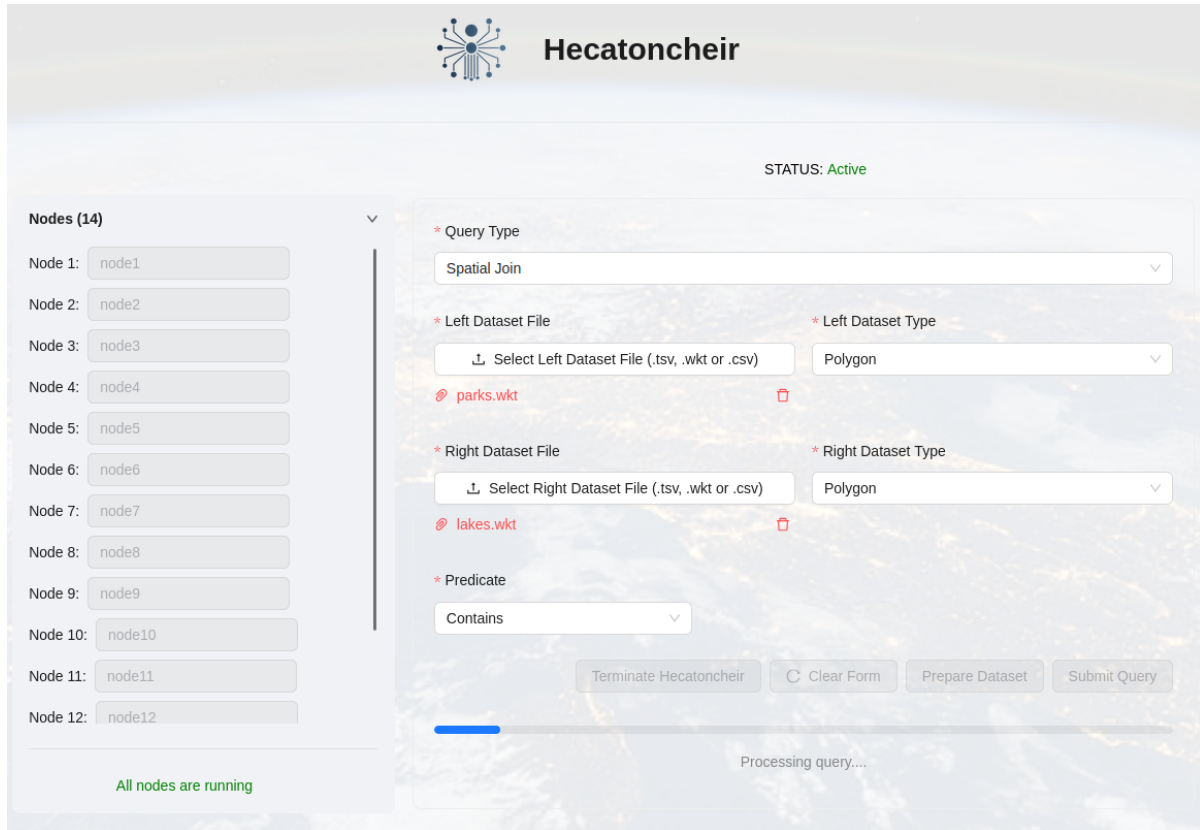


Figure 4.5: Snapshot of spatial query execution through Hecatoncheir’s GUI.

From that point on, the intuitive interface allows users to select and configure queries, load datasets, and retrieve results entirely through mouse interactions, eliminating the need for any manual coding, configuration or know-how of the system’s internals.

4.1.5 In-Memory Management

Hecatoncheir is designed to perform all operations in memory without requiring all data to be memory-resident. For instance, non-point geometries, which can have a significant memory footprint, can be stored on disk and loaded on-demand during query processing, ensuring that only the required objects are brought into memory when needed. This is attributed to Hecatoncheir’s filter-and-refine approach, that uses space-efficient approximations such as MBRs [1] and APRIL [29, 50] as effective filters.

If the cluster’s memory is constrained, the user has the option to forego storing geometries in memory, instead loading only the required ones on demand during query processing, which is expected to impact performance negatively, due to additional I/O

operations. This trade-off between memory usage and performance enhances Hecatoncheir’s flexibility in managing memory requirements.

4.2 Experimental Evaluation

We experimentally evaluate Hecatoncheir’s performance, scalability, and its ability to balance load and memory across a wide range of real-world datasets and query scenarios.

4.2.1 Setup

Hecatoncheir’s plug-and-play design enables deployment by simply downloading it directly from its GitHub repository onto a cluster of virtual machines. In our environment, each node in the cluster operates on a fresh installation of Ubuntu 22.04, configured with only the minimum requirements necessary for Hecatoncheir: (i) host set up with Boost Geometry 1.73, CMake 3.22, and C/C++17; (ii) MPICH installed on all nodes; and (iii) passwordless SSH configured between the nodes. Once downloaded, we build Hecatoncheir using the installation script and we execute the provided test suite to validate inter-node connectivity, communication, and the system’s deployment in the cluster.

Our cluster setup consists of 10 VMs on 5 host computers. Each VM runs on an Intel CPU i9 (4 cores), clocked at 3.70GHz and 12GB RAM. The virtual machines run Ubuntu 22.04 LTS and have C++17 and MPICH 4.0 installed.

We compare Hecatoncheir against Apache Sedona [52], set with 2 executor-cores per VM, 8GB of memory per executor, QuadTree partitioning with sampling enabled, RTree indexing per Spark partition and underlying Hadoop for data distribution.

4.2.2 Datasets

In our experiments, we used datasets from the Tiger 2015 and OpenStreetMap (OSM) collections of SpatialHadoop ([51]). For point data, we computed the centroids of dataset T2 to generate a new dataset T2P. Additionally, we randomly generated rectangular range queries QX on U.S. territory with varying selectivity $X = [0.01, 0.05, 0.1, 0.5, 1]$. A summary of the datasets is shown in Table 4.1.

Table 4.1: Summary of datasets that were used in our experiments.

Dataset	Data Type	Object Count	Description
T2	Polygon	2.3M	TIGER2015 U.S. Water Areas
T2P	Point	2.3M	TIGER2015 U.S. Water Areas Centroids
T8	Linestring	20M	TIGER2015 U.S. Roads
O3P	Point	114.7M	OSM Global Building Centroids
O5	Polygon	8.4M	OSM Global Lakes
O6	Polygon	10M	OSM Global Parks
QX	Rectangle	10K	U.S. $X\%$ selectivity random queries
NN	Point	100	Randomly sampled from T2P

4.2.3 Scalability

Hecatoncheir’s scalability stems from several factors. First, preparation tasks such as data distribution, indexing, and APRIL creation are distributed across all Workers, with minimal additional cost which is limited to a few extra inter-node communications and some overhead on the Host Worker for batch organization during partitioning. Additionally, query times are reduced through two-level parallelism, with operations executed concurrently by Workers and their threads.

Table 4.2 shows Hecatoncheir’s scalability for various query scenarios on an increasing number of Workers. For the two spatial intersection join scenarios, query evaluation times decrease as the number of Workers increases, demonstrating effective workload distribution. In the O5 $\bowtie$ O6 scenario the scaling converges around 8-10 nodes. In the distance join scenario, Hecatoncheir scales more gradually, primarily due to imbalances in the dataset distributions. While T2P contains points limited to the U.S. (lower 48 states), O3P is global, which can lead to uneven workload distribution across Workers, affecting scalability and performance. In the k NN scenario, performance does not scale, likely because the Host’s serial aggregation of results dominates the total runtime. In Section 4.2.5, we analyze these scenarios in greater detail, identifying bottlenecks and examining per-Worker workloads to provide deeper insights into Hecatoncheir’s scalability across different query types.

In batch range query evaluation, scalability can be influenced by several factors. First, batches of queries are broadcast to all Workers. This means that query win-

Table 4.2: Scaling of Hecatoncheir’s query evaluation time (seconds) for spatial join, distance join and k NN query scenarios.

Spatial Intersection Join					
Scenario/Nodes	2	4	6	8	10
O5⋈O6	14.68	8.12	7.41	5.83	6.75
O6⋈T8	3.78	2.63	2.12	2.16	1.95
Distance Join ($\epsilon = 0.001$)					
Scenario/Nodes	2	4	6	8	10
T2P⋈O3P	25.70	20.13	20.19	15.51	14.97
k NN ($k = 5$)					
Scenario/Nodes	2	4	6	8	10
NN on O3P	2.58	3.69	2.66	2.16	2.63

dows overlapping exclusively with partitions a Worker does not manage are quickly determined to yield zero results for that Worker, allowing it to move on to the remaining queries. However, with fewer Workers, each Worker is more likely to manage a partition relevant to a query, resulting in more queries being evaluated per Worker. This, in turn, increases the total batch evaluation time. Additionally, the selectivity of queries (i.e., the extent of the range windows) may overlap with multiple partitions in Hecatoncheir’s partitioning and indexing grids. This increases the number of partitions that need to be evaluated, which (especially in setups with fewer nodes) can reduce task parallelism and lead to more serialized evaluation. Consequently, the available pool of threads on each node must handle more partitions, further reducing per-node parallelism.

Table 4.3 illustrates Hecatoncheir’s scalability on batch range queries over the T2P dataset for varying selectivities and increasing node count. Notice that low-selectivity query windows do not benefit significantly from adding more Workers. This is due to two factors: first, the dataset is relatively small, so evaluating individual range queries is not very costly; second, as mentioned earlier, small range windows overlap fewer partitions per query, resulting in less total workload to parallelize. In contrast, for 1% selectivity range queries, Hecatoncheir demonstrates noticeable scaling as additional Workers are added, with performance converging around 4–6 Workers. On the much larger, global O3P dataset, which cannot fit into the memory of a single machine,

Table 4.3: Scaling of Hecatoncheir’s query evaluation time (seconds) for the QX batch range queries on the T2P and O3P datasets.

QX on T2P						
Selectivity $X\%$ /Nodes	1	2	4	6	8	10
0.01	0.12	0.10	0.15	0.14	0.16	0.21
0.1	0.41	0.24	0.18	0.20	0.20	0.25
1	2.95	1.52	0.78	0.84	0.64	0.65
QX on O3P						
Selectivity $X\%$ /Nodes	1	2	4	6	8	10
0.01	N/A	0.30	0.23	0.22	0.24	0.23
0.1	N/A	0.72	0.67	0.56	0.66	0.52
1	N/A	3.33	2.27	1.74	1.88	1.66

Hecatoncheir’s scalability becomes even more evident. Even at smaller selectivities, such as 0.1%, the vast number of objects to be processed highlights the advantages of Hecatoncheir’s internal parallelization and batch-processing optimizations that scale combined with the increasing number of Workers.

4.2.4 Performance

4.2.4.1 Range Queries

We compare the performance of Hecatoncheir and Sedona across various spatial query types. Since Sedona does not natively support batch range queries and to avoid the costly overhead of rebuilding indexes for each query, we implemented batch range query processing as a spatial join between the queries and the input dataset. Table 4.4 summarizes two batch range query scenarios for varying selectivity, on datasets T2P and O3P. In both cases Hecatoncheir outperforms Sedona by at least two orders of magnitude.

Hecatoncheir’s superior performance relies on several factors: first, Hecatoncheir parallelizes query processing by broadcasting range queries in batches, allowing Workers to evaluate them asynchronously and in parallel with each other, as making the most out of their available threads to parallelize the processing of each batch. Results are also returned in batches to the Host to minimize communication overhead

Table 4.4: Hecatoncheir and Apache Sedona comparison on Range Queries evaluation time (seconds), in a cluster with 10 nodes.

QX on T2P			QX on O3P		
Selectivity $X\%$	Sedona	Hecatoncheir	Selectivity $X\%$	Sedona	Hecatoncheir
0.01	18	0.16	0.01	70	0.23
0.05	29	0.16	0.05	92	0.37
0.1	61	0.19	0.1	97	0.55
0.5	85	0.36	0.5	187	1.09
1	210	0.51	1	203	1.59

Table 4.5: Hecatoncheir and Apache Sedona comparison on Spatial Intersection Join evaluation time (seconds), in a cluster with 10 nodes.

O5⋈O6			O6⋈T8		
	Sedona	Hecatoncheir		Sedona	Hecatoncheir
Time (s)	1378	6.74	Time (s)	299	1.98

and facilitate more flexible result aggregation.

4.2.4.2 Spatial Intersection Joins

As shown in Table 4.5, Apache Sedona required 1378 seconds to run the O5⋈O6 spatial intersection join on 10 nodes, whereas Hecatoncheir completed the task in just 6.74 seconds. Similarly, in the O6⋈T8 polygon–linestring join, Hecatoncheir again achieved a substantial performance advantage. This efficiency stems from Hecatoncheir’s high-performance two-layer index combined with APRIL spatial approximations, which enable advanced, efficient, and accurate filtering while significantly reducing refinement overhead for complex geometries. We also observed that Apache Sedona produced many more object pairs as results that do not actually intersect, indicating that its internal geometrical operations are less precise than the Boost Geometry library used by Hecatoncheir.

4.2.4.3 k NN Queries

Table 4.6 compares Hecatoncheir and Apache Sedona on k NN queries for various values of k . The performance of both frameworks is only slightly affected by increasing

Table 4.6: Hecatoncheir and Apache Sedona comparison on k NN query evaluation time (seconds), for increasing k , in a cluster with 10 nodes.

NN on T2P			NN on O3P		
k	Sedona	Hecatoncheir	k	Sedona	Hecatoncheir
5	154	0.71	5	N/A	2.38
50	159	0.72	50	N/A	2.44
500	156	0.79	500	N/A	2.71

k . We were unable to run the k NN queries on the O3P dataset in our cluster, as Apache Sedona repeatedly failed internally, even when the allocated executor memory was reduced significantly.

Both frameworks evaluate k NN queries in a similar way, processing points one by one without batch execution. Hecatoncheir, however, fully exploits available threads to parallelize partition checks and comparisons, achieving over two orders of magnitude faster performance than Apache Sedona in both scenarios.

4.2.4.4 Distance Joins

We ran the O3P $\bowtie$ T2P distance join scenario on both Apache Sedona and Hecatoncheir. As shown in Table 4.7, Hecatoncheir outperforms Apache Sedona in this scenario as well, though the margin is smaller compared to other query types. This is because Hecatoncheir’s current (and preliminary) distance join implementation consists of several sequential phases, with limited opportunities for parallelization beyond partitioning and distance evaluations.

In future work, we plan to optimize distance join evaluation to be more parallel and asynchronous across Workers, enabling faster overall performance.

4.2.5 Profiling

In this section, we present experimental results that profile Hecatoncheir’s memory usage and load balancing across Workers. Using the TAU system profiling tool [130], we break down and measure the execution time, memory footprint and number of MPI receive/send operations on a per-Worker basis.

In all experiments, we used 10 Worker nodes (ranks 1–10), with rank 0 denoting

Table 4.7: Hecatoncheir and Apache Sedona comparison on Distance Join evaluation time (seconds) for increasing distance value ϵ , in a cluster with 10 nodes.

O3P $\bowtie$ T2P		
ϵ	Sedona	Hecatoncheir
0.001	72	17
0.005	85	16
0.01	102	21
0.1	268	55

the Host process, which runs on the same machine as the Worker assigned rank 1. In the following plots, columns of the same color (i.e., representing the same metric) with comparable values indicate that the metric is balanced across Workers, a desirable property in distributed systems such as Hecatoncheir.

Figure 4.6 illustrates the O5 $\bowtie$ O6 spatial intersection join scenario. Both the peak memory footprints (Figure 4.6(a)) and query evaluation times (Figure 4.6(b)) per Worker fall within similar ranges, indicating that each Worker requires a comparable amount of memory to index its assigned data in-memory and spends a similar amount of time processing it, without significant variation. This behavior can be attributed to the O5 and O6 datasets, which span the entire globe and exhibit similar distributions. Furthermore, all Workers perform comparable MPI send and receive operations (Figure 4.6(c)) throughout the end-to-end workflow.

In contrast, Figure 4.7 profiles a distance join between the T2P and O3P datasets, which exhibit markedly different spatial distributions. T2P contains points exclusively within the contiguous United States (lo48), whereas O3P spans the entire globe. This discrepancy is reflected in the evaluation times: Workers 9 and 10 require substantially more time than Workers 5 and 7. We infer that Workers 9 and 10 are responsible for a larger portion of data located in the U.S., resulting in a higher number of comparisons. Conversely, the data assigned to Workers 5 and 7 likely resides outside the U.S., enabling them to quickly discard non-results due to the properties of the grid index. Notably, the peak memory footprints of all Workers remain within a similar range, reinforcing the conclusion that the differences in evaluation time stem from the topological characteristics of the data rather than disparities in data volume. Additionally, in the distance join, data exchange occurs between Workers, creating a

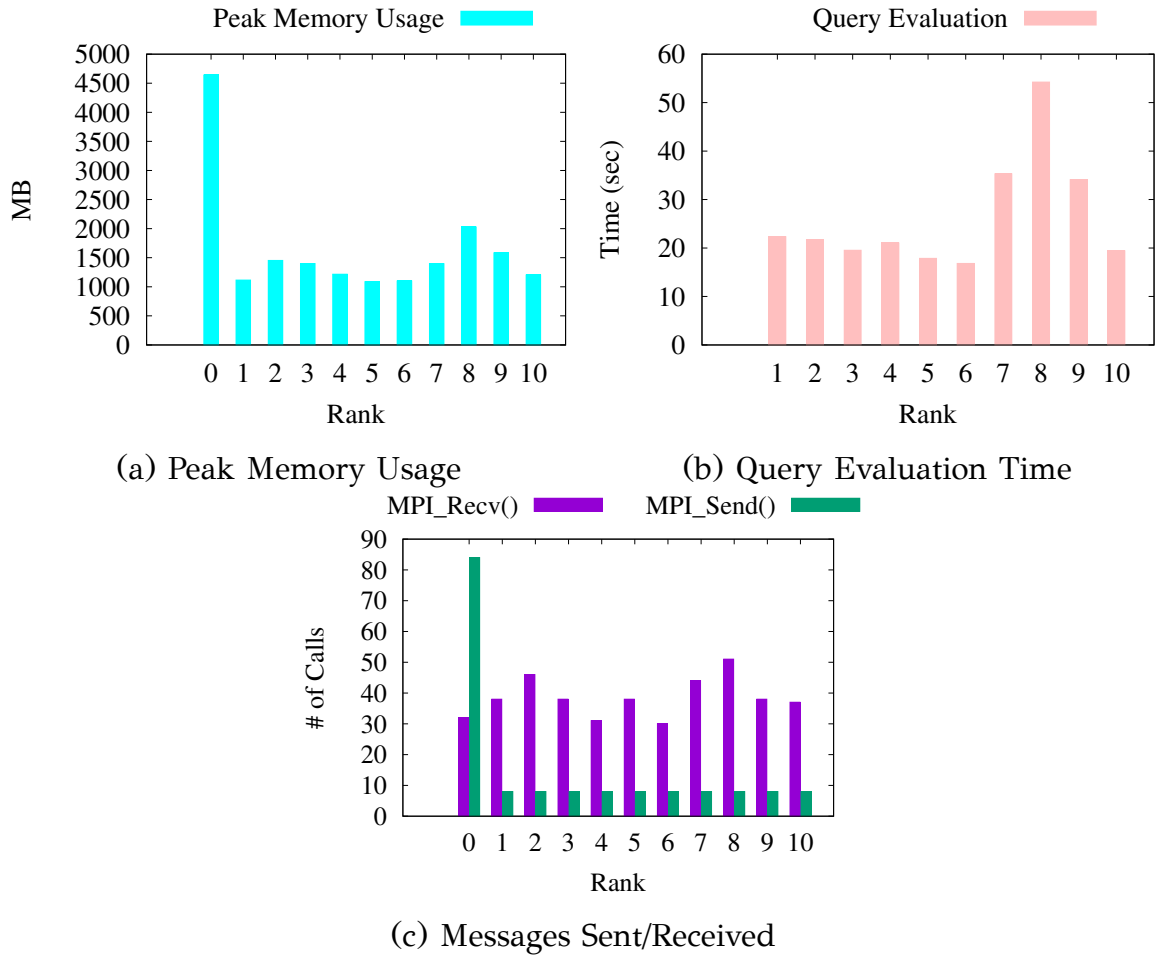


Figure 4.6: Per Worker (rank) profiling for the O5×O6 spatial intersection join, regarding count of messages sent/received (a), memory (b), and query execution time (c).

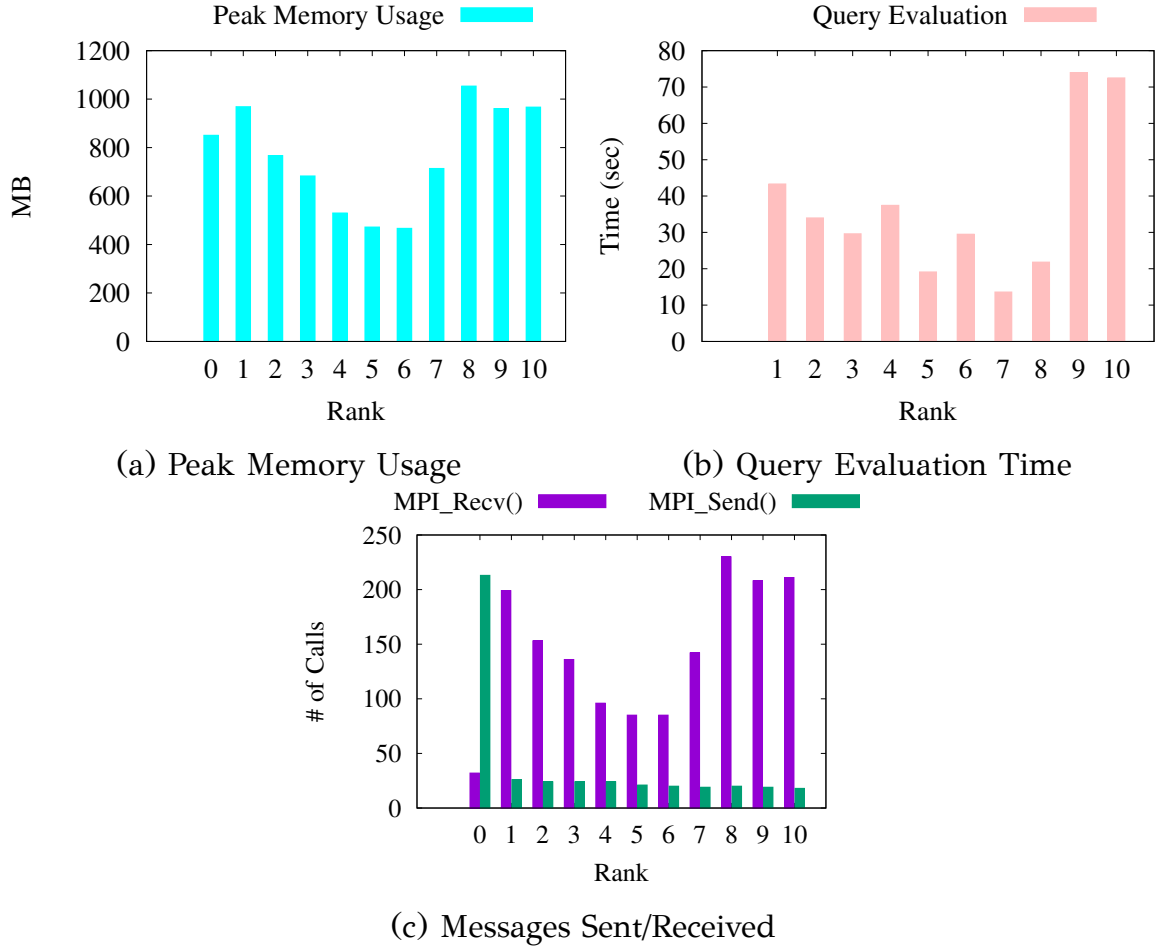


Figure 4.7: Per Worker (rank) profiling for the T2P⋈O3P distance join with $\epsilon = 0.001$, regarding count of messages sent/received (a), memory (b), and query execution time (c).

correlation between the number of MPI receive operations and the total evaluation time: Workers that receive more data from others must evaluate it, thereby increasing their overall processing time.

Figure 4.8 profiles Hecatoncheir in a batch range query scenario with high-selectivity queries (Q1) on the O3P dataset. As in Figure 4.7, the Q1 queries are restricted to the U.S., whereas O3P spans the globe. This discrepancy is again reflected in the total query evaluation time, which is significantly higher for Worker 5 than for Worker 3. However, since Workers evaluate batches of queries asynchronously and return results to the Host as soon as a batch is complete, there is minimal waiting among Workers, except toward the very end of the dataset evaluation. In this scenario, we measured the average query evaluation time per Worker to be 220.25 seconds. Meanwhile, the Host spent a total of 155.65 seconds gathering results for all

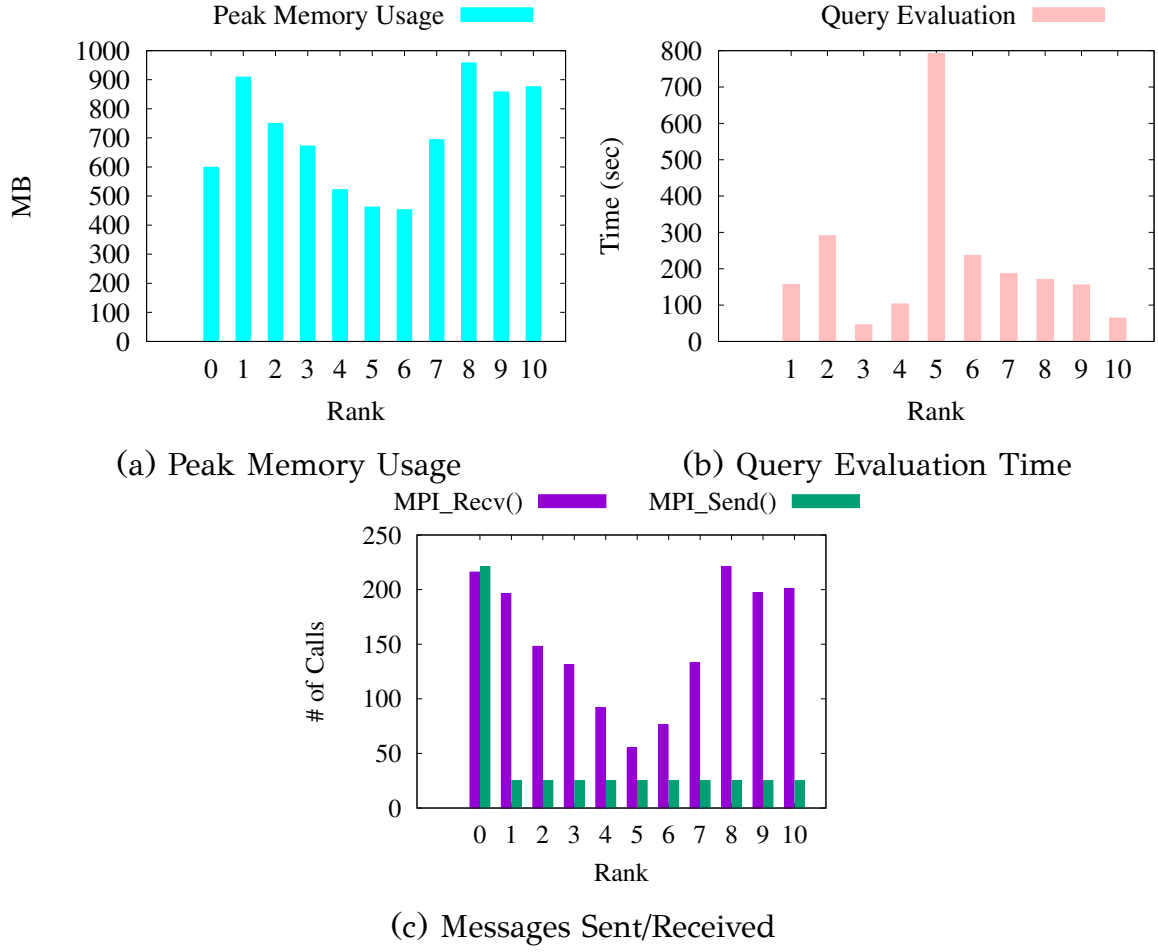


Figure 4.8: Per Worker (rank) profiling for the Q1 range queries on the O3P dataset, regarding count of messages sent/received (a), memory (b), and query execution time (c).

batches, meaning that on average, by the time all Workers finished their evaluations, the user experienced minimal delay for the Host to aggregate the results. Workers exhibit memory requirements of a similar order of magnitude, with slight variations arising from the O3P dataset’s global distribution. The data tends to cluster in high-density regions such as Europe, Asia, and North America, while remaining sparse across vast areas like the oceans.

Figures 4.9(a) and (b) illustrate the uneven workloads and memory requirements across Workers in the k NN scenario on the O3P dataset with $k = 5$. As in Figure 4.8, the memory requirements are uneven due to the clustered global distribution of the O3P dataset. In k NN queries, however, this imbalance has little impact, since all Workers compute a local set of k nearest neighbors for each query point, and the final results are aggregated by the Host. This behavior is evident in Figure 4.9(b),

where most Workers perform similarly across the entire query set, with only minor variations. Notably, Workers 6, 7, and 9 are the fastest, as their partitions (and neighboring partitions) contain relatively little data, reducing the number of points they must iterate through to identify the top- k neighbors. An interesting observation is that all Workers completed their respective k NN evaluations in 1.81 to 5.65 seconds, whereas the Host’s gathering and aggregation of the k NN results took about 7.2 seconds in total. This creates a bottleneck in the query evaluation pipeline: queries are processed sequentially, and even though some Workers finish early, they must wait for others to complete and for the Host to aggregate the results before moving on to the next query in the batch. In future work, we plan to adopt an asynchronous batch evaluation technique similar to the one used in Hecatoncheir’s range query evaluation. With respect to send/receive operations, all Workers issue the same number of `MPI_Send()` calls. The variation in `MPI_Recv()` calls arises from data partitioning, as some Workers receive more data than others.

4.3 Conclusions

In this chapter, we introduced Hecatoncheir, a prototype distributed spatial data management framework built with MPI and OpenMP. Hecatoncheir is designed for high performance, leveraging state-of-the-art indexes, spatial approximations, and query optimization techniques, while also being a self-contained C++ library that is easy to set up and use for fast, accurate spatial data management. Hecatoncheir abstracts its internals by offering a high-level, C++ header API and a web-based GUI for users to interact with it.

Through detailed experiments on real-world datasets and system profiling, we demonstrated Hecatoncheir’s superior performance and flexibility compared to the current state-of-the-art distributed framework, Apache Sedona.

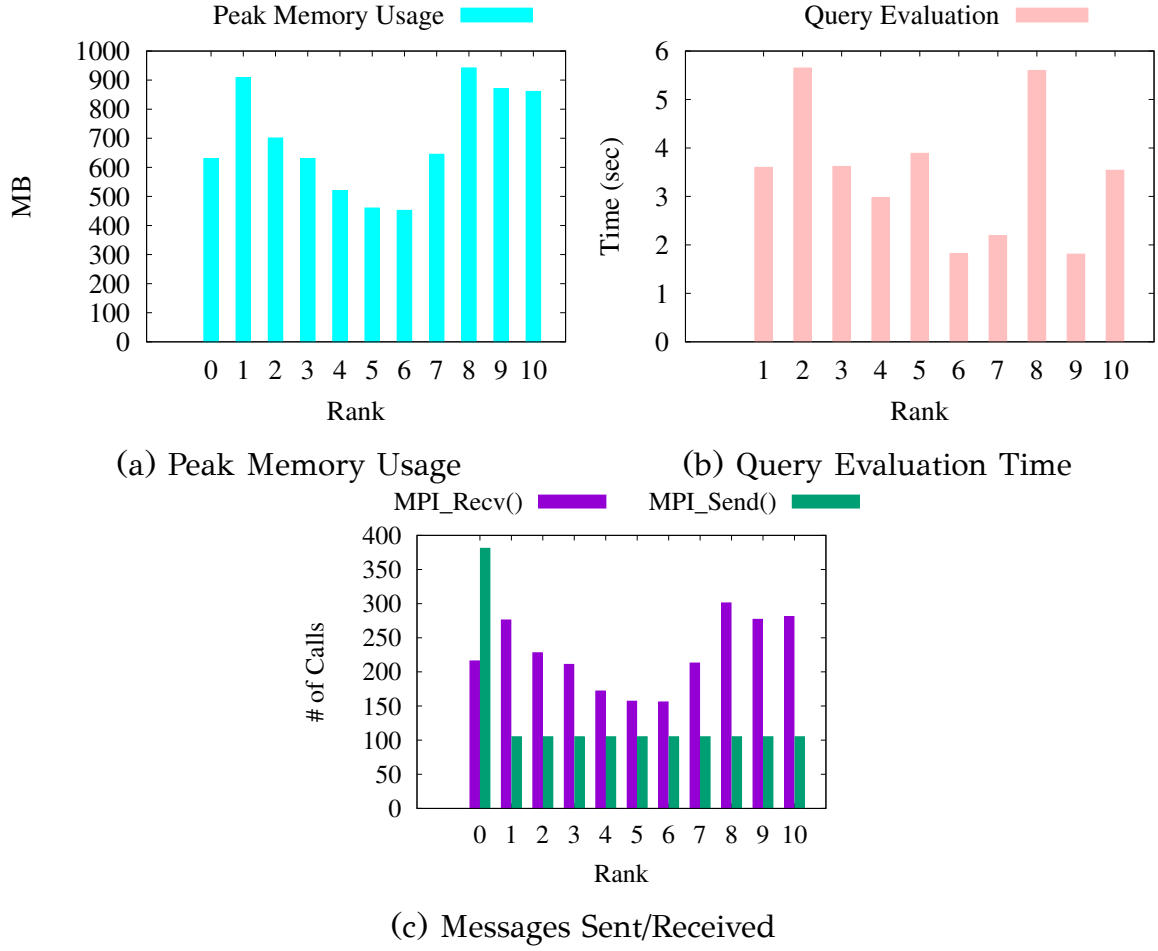


Figure 4.9: Per Worker (rank) profiling for the NN k NN queries on the O3P dataset with $k = 5$, regarding count of messages sent/received (a), memory (b), and query execution time (c).

CHAPTER 5

SPATIAL REASONING USING RETRIEVAL-AUGMENTED GENERATION

5.1 SpaRAGraph Overview

5.2 Pre-processing

5.3 Spatial Reasoning with SpaRAGraph

5.4 SRB - The Spatial Reasoning Benchmark

5.5 Experimental Evaluation

5.6 Conclusions

As discussed in Sections 1.3 and 2.4, LLMs are increasingly being integrated into GIS as AI assistants or agentic tools for extracting spatial reasoning knowledge from spatial data. Since spatial data is often highly domain-specific, it is natural to consider RAG as a method for enhancing LLM performance in these tasks. This, however, raises the question of whether LLMs can truly comprehend and utilize spatial knowledge expressed in text, and how scalable and efficient such an approach can be in real time.

In this chapter, we propose *SpaRAGraph*, a RAG-based approach that enhances LLMs’ spatial reasoning for domain-specific questions by generating relevant context from pre-processed topological relations in a spatial database. SpaRAGraph leverages

our proposed spatial approximation and processing techniques to compute these relations scalably over large spatial datasets, which are then provided to LLMs via RAG to improve performance on spatial reasoning tasks in real time.

Outline In Section 5.1 we present an overview of SpaRAGraph’s architecture, describing its components, usage and goal. Section 5.2 describes in detail SpaRAGraph’s pre-processing stage and how topological relations are extracted and structured to facilitate its RAG mechanism for efficient spatial reasoning context generation. In Section 5.3 we describe how SpaRAGraph uses the aforementioned pre-processed spatial relations at inference time through RAG. In Section 5.4 we propose a benchmark designed to evaluate the performance of RAG-based techniques on the improving spatial reasoning capabilities of LLMs. Section 5.5 includes our experiments that evaluate SpaRAGraph’s efficiency. Finally, in 5.6 we discuss our conclusions.

5.1 SpaRAGraph Overview

This section presents an overview of SpaRAGraph, describing its end-to-end pipeline and showcasing its usage and benefits through example scenarios.

Spatial Textual Context The geometry of a spatial entity is represented by a sequence of geographic coordinates (longitude, latitude). To compute spatial relations between entities from their raw representations, costly operations, such as line intersection detection, point-in-polygon tests (to detect containment of an object into another), and distance calculations (for proximity detection) must be applied [131]. Additionally, spatial, domain-specific knowledge is missing from foundation models, giving room for improvement via RAG. We hypothesize that if spatial knowledge is expressed comprehensively (via natural language) and concisely (lack of noise, redundancy) in textual form, then the LLM may be able to infer spatial relationships between objects. For example, consider the prompt: “*What is the relative location of A to C, if A is west of B and B is west of C?*” This question asks for the spatial relationship between entities A and C while providing intermediate context from which the relationship can be inferred. When we presented this prompt to Llama-3.1-8B-Instruct, it responded as follows: “*To find the relative location of A to C, let’s break it down: A is west of B, B is west of C. This means A is to the west of B, and B is to the west of C. So, A is to the west of C.*” The LLM successfully inferred the correct spatial relation

between A and C using the context provided with the question, purely over text. Without such context, however, the same model fails to infer the correct relationship between entities (Figure 5.4).

5.1.1 Pipeline

Figure 5.1 illustrates the full SpaRAGraph framework, which is divided into two main stages: pre-processing and inference. **During pre-processing** (performed once unless the underlying datasets change), all spatial datasets are processed through *SpaTex*. This module computes precise spatial relationships (both topological and directional) between entities by analyzing their geometries. The resulting relationships are encoded as RDF triplets and indexed into a global spatial graph. A relation composition matrix is also defined, capturing all possible transitions between spatial relations to support multi-hop reasoning across the graph.

At inference time, when a user poses a spatial question, SpaRAGraph uses named entity recognition (NER) via spaCy [132] to extract mentioned spatial entities. These are matched to graph nodes using FAISS-based similarity search. The shortest paths connecting matched entities are then computed, and the corresponding RDF triplets are retrieved. Using the relation composition matrix, these triplets are summarized into a coherent spatial context, which is appended to the user’s original query. This enriched prompt is finally passed to the LLM for response generation.

5.1.2 Usage

An example interaction with our framework is shown in Figure 5.2, demonstrating how it facilitates spatial reasoning and natural language dialogue by automatically generating the necessary contextual information, relieving the user of the burden to supply it manually. We focus on generation tasks involving spatial relations (e.g., addressing the following question: “*Is Dickinson County, Kansas east of Douglas County, Kansas?*”), implementing and testing our framework using spatial data. Nonetheless, our approach can be generalized to assist any RAG approach that involves complex relations between objects that can be supported by inference rules. Additionally, SpaRAGraph is compatible with existing RDF datasets, bypassing the need for spatial-to-RDF pre-processing, provided that the RDF data encodes semantically meaningful relations between entities. For effective traversal and inference, a composition matrix

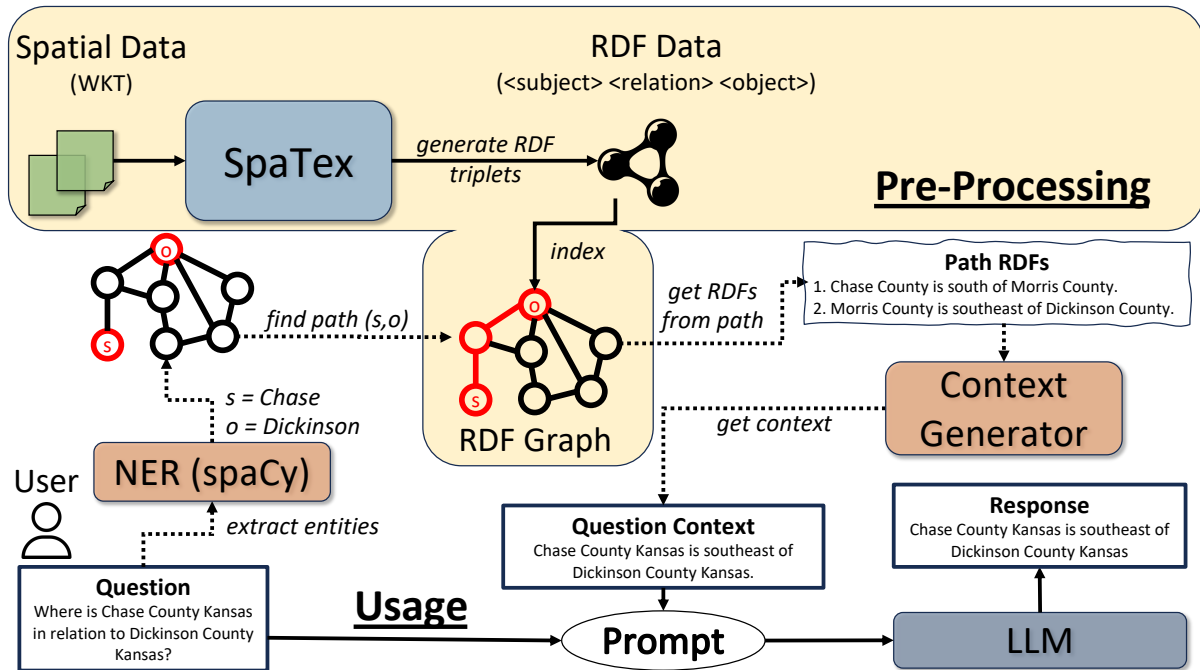


Figure 5.1: SpaRAGraph's overview divided into two stages: pre-processing, performed once and consisting of *SpaTex*'s spatial RDF generation and their indexing in a graph; and usage, which illustrates user interaction with SpaRAGraph, showing how context is generated in the back-end through named entity recognition and path search on the RDF graph, before being appended to the prompt and sent to the LLM.

- **Using SpaRAGraph** - Give a question (type 'exit' to quit): Is Dickinson County, Kansas east of Douglas County, Kansas?
- **Generated Context** - Dickinson County, Kansas is west of Douglas County, Kansas.
- **Prompt** - Question: Is Dickinson County, Kansas east of Douglas County, Kansas?
Context: Dickinson County, Kansas is west of Douglas County, Kansas.
- **Response** - No, Dickinson County, Kansas is to the west of Douglas County, Kansas.

Figure 5.2: Example interaction with SpaRAGraph when given a spatial question. The context is automatically generated and appended to the prompt, to guide the model's response.

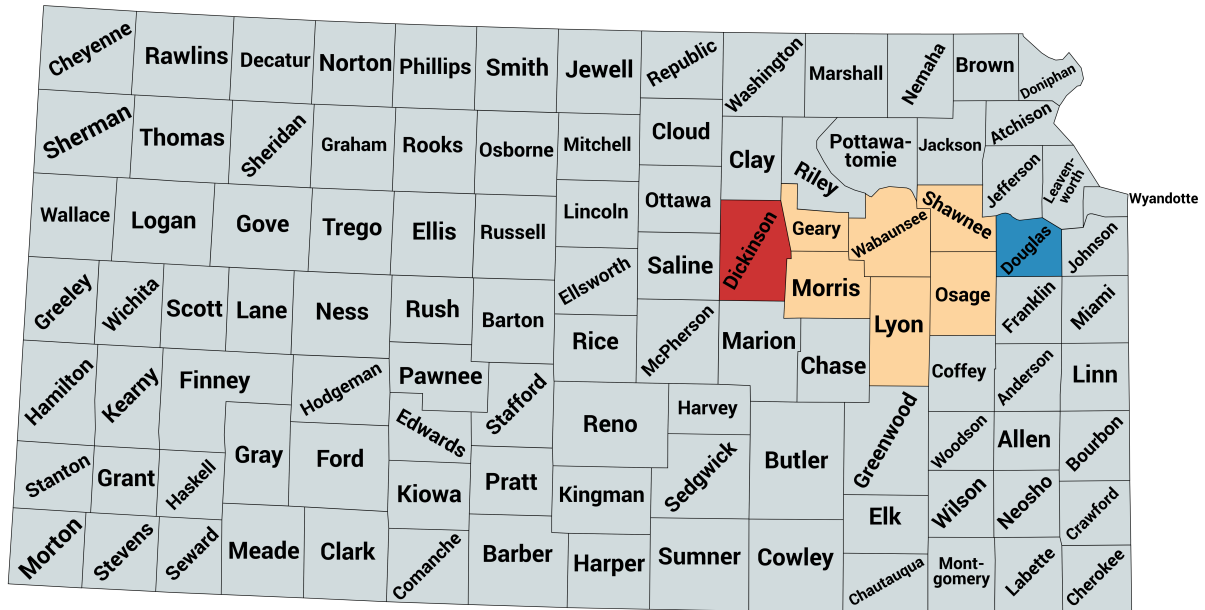


Figure 5.3: Map of all counties in Kansas, illustrating possible shortest paths (yellow) between Dickinson County (red) and Douglas County (blue). These paths enable inference of the relative spatial relation between the two entities through intermediate connections. The map was created on <https://www.mapchart.net>

or algebra must be defined over the relation set, enabling the combination of any pair of relations to compute a composed relation between any two connected entities in the graph.

Figure 5.3 presents the complete map of Kansas counties, with the possible shortest paths highlighted through which the spatial relation between Dickinson County and Douglas County was inferred using the RDF graph. Assuming that each pair of neighboring counties has their spatial relation explicitly encoded in RDF, the relation between the two counties can be derived compositionally. For instance, Dickinson County is west of Geary County, Geary County is west of Wabaunsee County, and so on, allowing us to infer that Dickinson County is west of Douglas County by chaining these intermediate spatial relations step by step. In the example of Figure 5.2, SpaRAGraph helps Llama-3.1-8B-Instruct respond correctly to a question regarding the relative topology of two counties in Kansas, by enriching the original prompt with the necessary context for an accurate response. With the context-enriched prompt, the model is able to answer the question accurately.

Without the generated context, the model may either be unable to respond or hallucinate on the answer based on the general knowledge it might possess. In Figure

USER
<i>What is the relative location of Dickinson County Kansas to Douglas County Kansas?</i>
LLM
<i>To find the relative location of Dickinson County, Kansas to Douglas County, Kansas, I'll need to know their respective locations.</i>
<i>Dickinson County, Kansas is located in the northwestern part of the state, while Douglas County, Kansas is located in the northeastern part of the state.</i>
<i>Dickinson County is generally east of the 96th meridian, while Douglas County is generally east of the 96th meridian.</i>
<i>Dickinson County is north of Sumner County and south of Cloud County.</i>
<i>Douglas County is north of Miami County and south of Jefferson County.</i>
<i>Given this information, Douglas County, Kansas is north of Dickinson County, Kansas.</i>

Figure 5.4: Llama-3.1-8B-Instruct response to an example spatial question.

5.4, for example, we prompted Meta’s Llama-3.1-8B-Instruct model with the question: “*What is the relative location of Dickinson County to Douglas County in Kansas?*” Based on its response, it becomes evident that the model possesses a general understanding of the locations of the two Counties within the State of Kansas, but it is ultimately unable to infer their relative spatial positioning and hallucinates, responding incorrectly. This limitation highlights the model’s deficiency in spatial reasoning based solely on textual information. On the other hand, Figure 5.5 shows the model’s response to the same question **when employed with SpaRAGraph**. The context generated by SpaRAGraph was deterministically constructed and explicitly conveyed the relative locations of the two counties to the model, thereby guiding and informing its response internally.

5.2 Pre-processing

SpaRAGraph relies on the pre-processing of raw spatial data to generate synthetic RDF data that captures key relationships between entities. By explicitly storing a subset of spatial relations from the dataset, SpaRAGraph can infer relations that are not explicitly stored and enriches model responses with related context. The pre-

USER
What is the relative location of Dickinson County Kansas to Douglas County Kansas?
LLM + SpaRAGraph
Since Dickinson County Kansas is west of Douglas County Kansas, its relative location to Douglas County Kansas is to the west.

Figure 5.5: Llama-3.1-8B-Instruct response to an example spatial question when employed with SpaRAGraph.

processing stage consists of two components. First, the *SpaTex* component processes all input spatial data and generates RDF triplets that capture meaningful spatial relations among entities. Second, these RDF triplets are organized into a bidirectional graph index, forming a per-relation topological structure over all unique entities. This enables efficient computation of relationships between entities by traversing the graph through intermediate nodes.

5.2.1 *SpaTex*: Spatial Relation RDF Generator

Spatial knowledge may contain various different aspects and metrics, such as the distance between entities, their topological relationships (e.g. adjacent, intersect) and the cardinal direction of an entity in relation to another one (e.g. north, southwest). We refer to any type of relation between two geographical entities as a *spatial relation*. Note that determining the cardinal direction between two non-point, arbitrary polygons (such as countries or lakes) can be inherently subjective. Different parts of one object may lie north, east, or northeast of parts of the other. To produce a definitive spatial relation, we compute the centroids of each polygon and determine the cardinal direction based on the relative position of these centroids. While this approach simplifies the spatial representation, it introduces some distortion in the resulting RDF triplets, rendering them approximate. Consequently, any inference derived from these relations should also be considered approximate.

To extract these spatial relations and generate RDF triplets that describe them comprehensively and concisely, we introduce *SpaTex*, a rule-based spatial-to-RDF data generator that takes as input spatial data collections in raw format (WKT, CSV, etc.). An overview of *SpaTex* is shown in Figure 5.6. The output is a collection of RDF triplets that encapsulate the spatial relations between (nearby) pairs of objects.

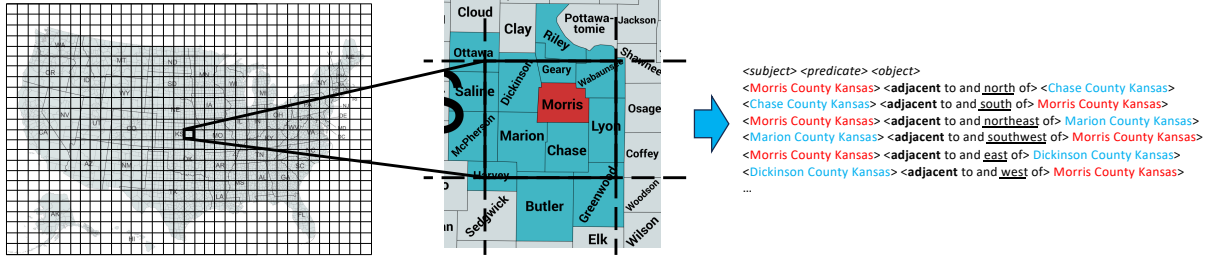


Figure 5.6: The Spatial relation identification process by *SpaTex* that uses a global grid to group nearby entities and compute their spatial relations, outputting them in RDF format.

We aim for the RDF triplets to be as concise and simple as possible, minimizing redundancy by ensuring that each subject and object consists of a single entity name, and that each predicate captures only the spatial relation, free of any descriptive or “flavor” text. However, predicates must still be concise and semantically informative enough for an LLM to effectively infer implicit spatial relationships not explicitly encoded in the data.

Generating spatial RDF triplets that capture nearby entities’ spatial relations is computationally challenging. The total number of pairwise spatial relations between entities on a map is quadratic, making their generation and encoding a challenging task. In *SpaRAGraph*, we address this scalability challenge by i) dividing the map into numerous local partitions, ii) computing non-trivial spatial relations between all pairs of entities within each local region, and iii) structuring the computed relations in a clear and comprehensive RDF format to enhance the model’s ability to infer non-local relations.

The partitioning approach employed by *SpaTex* has two advantages. First, we avoid computing an excessive (and redundant) number of spatial relations, which can be inferred; for two entities (e.g., Counties) in different partitions, their relation should be disjoint and their relative cardinal direction can be inferred by the cardinal directions of entities that enclose them (e.g., States). Second, each partition is processed independently and in parallel, scaling up the relation generation process. The input spatial data must be accompanied by metadata that provides a human-readable name for each entity, enabling reference in natural language. RDF data often uses URIs, which may or may not include meaningful names. When the URIs already contain readable names, *SpaTex* retains them as node labels in the graph. However, if the URIs consist of non-descriptive identifiers or links, *SpaTex* replaces them with

custom URIs derived from the provided metadata (names) to ensure compatibility with natural language processing.

For the detection of topological relations, we use the standard Dimensionally Extended 9-Intersection model (DE-9IM) [40]. DE-9IM defines a 3×3 matrix where the rows and columns represent two objects' interior, boundary and exterior areas. Note that, by the definition of DE-9IM, spatial relations are asymmetric. This means that any pair of entities can have at most one spatial relation in a given direction, and at most its inverse in the opposite direction. In other words, a single pair of entities cannot simultaneously be related by multiple distinct spatial predicates. The combination of values in the matrix defines the exact topological relationship for two objects. Moreover, *SpaTex* calculates the cardinal direction between nearby objects in relation to one another, as well as their in-between distance and their common area (if any) in square kilometers. For two input spatial datasets R and S , *SpaTex* performs a *spatial join* $R \bowtie S$ between them, an operation that identifies all pairs of objects $\langle (r, s) | r \in R, s \in S \rangle$ that intersect with each other. For each dataset, a *self-join* is performed ($R \bowtie R$ and $S \bowtie S$), to identify relations between objects in the same dataset as well.

The vast majority of object pairs in real-world spatial datasets are *disjoint* [29], so we only detect and generate non-disjoint topological relations, as disjointness can be implied. This saves us both the effort and the overhead of encoding and retrieving disjoint relations. In general, spatial relations between objects that are disjoint and far from each other can be inferred by LLMs and do not need to be explicitly defined in the context. For example, describing two entities as adjacent implies that their borders touch and thus, LLMs can infer that since they touch, they are not disjoint with each other.

SpaRAGraph takes advantage of spatial reasoning as much as possible to reduce the volume of the generated relations by *SpaTex*. To this end, we partition the data space using a uniform grid and assign each spatial entity to the partitions (i.e., tiles) that it spatially overlaps. *SpaTex* then performs a partition-to-partition spatial join [49] for each cell; hence, we only compute and generate the spatial relations between objects of the same tile. For any pair of objects in a partition, we first compare their Minimum Bounding Rectangle ($MBR(r)$). If the MBRs do not intersect, then we only compute the relative cardinal direction between them (e.g., north of); otherwise, we compute the DE-9IM matrix. For overlapping objects, we only generate the topological

relation (e.g., overlaps, inside, covers); if the objects are adjacent, we also compute their cardinal direction relation.

5.2.2 Graph-based Topology Index

We index the spatial RDF triplets generated by *SpaTex* using a directed graph $G = (V, E)$, where each node $v \in V$ represents an entity from the original data and each edge $e = (v_1, v_2) \in E$ corresponds to a spatial relation (predicate) from v_1 (subject) to v_2 (object).

For each pair of entities, *SpaTex* generates bi-directional RDF triplets. For example, both $\langle \text{The State of Kansas} \rangle \langle \text{contains} \rangle \langle \text{Morris County Kansas} \rangle$ and $\langle \text{Morris County Kansas} \rangle \langle \text{inside} \rangle \langle \text{The State of Kansas} \rangle$ are created. This results in two edges between the nodes $\langle \text{Morris County Kansas} \rangle$ and $\langle \text{The State of Kansas} \rangle$, one labeled $\langle \text{contains} \rangle$, the other $\langle \text{inside} \rangle$. This bi-directionality supports a greater number of possible paths during graph indexing, resulting in more flexible graph traversal. An illustration of an example graph index for a set of RDF triplets is shown in Figure 5.7. Recall that *SpaTex* computes spatial relations only between entities that fall within the same grid cell.

The total number of nodes $|V|$ in the graph corresponds to the number of unique entities in the original input data. The number of edges $|E|$, however, depends on the granularity of the grid used by *SpaTex* when computing spatial relations. A fine-grained grid results in fewer RDF triplets, as each object is compared against a smaller set of nearby neighbors. In contrast, a coarse-grained grid leads to a significantly higher number of relations, since each entity is compared with more entities within the same larger cell. Grid granularity is directly correlated with the size of the objects (in terms of area covered). We found that relatively coarse-grained grids (e.g., 1,000 cells per dimension) are sufficient for larger entities, such as U.S. Counties and States. In contrast, smaller entities (such as Zipcodes) benefit from finer grids (e.g., 10,000 cells per dimension), which produce a meaningful number of spatial relations without generating an excessive number of RDF triplets.

We hypothesize that a small number of spatial RDF triples is sufficient to help models infer missing relations, allowing them to answer questions about entities whose relationships are not explicitly stored by reasoning over the available information. The graph index represents the final step of the pre-processing stage and must

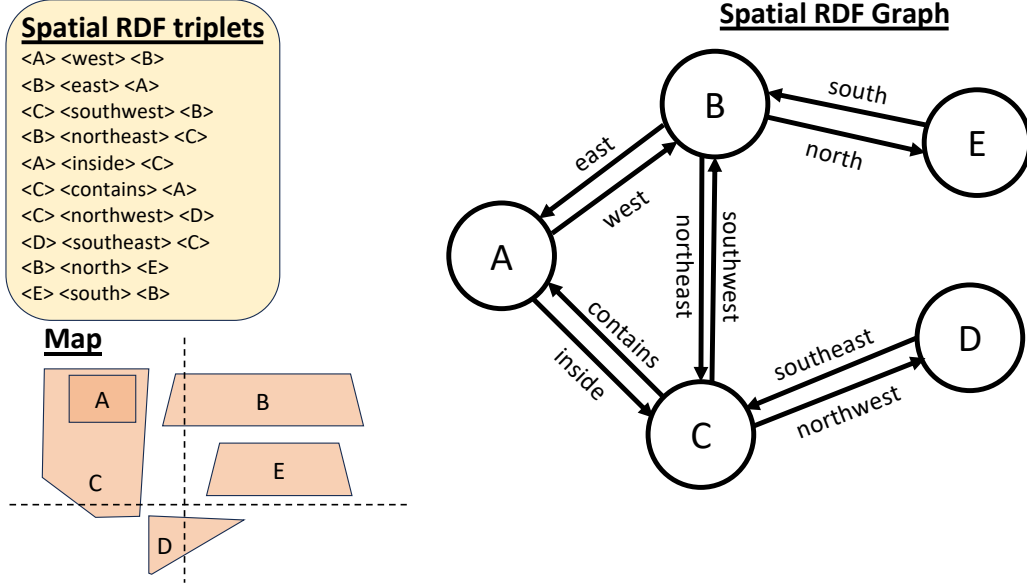


Figure 5.7: An example graph constructed from a set of spatial RDF triplets. The map illustrates the spatial arrangement of objects in the data space, with dotted lines indicating the grid cells defined by *SpaTex* and the resulting bi-directional index built over them.

be loaded and ready at inference time. Since the spatial relationships between entities typically remain static, there is usually no need to repeat the pre-processing pipeline (from *SpaTex*'s RDF generation to graph indexing) more than once.

5.2.3 Spatial Relation Composition matrix

Building on the foundations established by DE-9IM [40, 133], RCC8 [134], and Cardinal Direction Composition [135], we define a simplified composition matrix for pairs of spatial relations (Table 5.1). The matrix sets the rules for composing an overall spatial relation between two entities A and B , based on their relation with an intermediate entity I . Specifically, for any two RDF triplets $\langle A \rangle \langle p_1 \rangle \langle I \rangle$ and $\langle I \rangle \langle p_2 \rangle \langle B \rangle$, the matrix defines the composition of the two predicates p_1, p_2 as $p_1 \rightarrow p_2 = p_3$, forming a new relation p_3 such that the RDF triplet $\langle A \rangle \langle p_3 \rangle \langle B \rangle$ describes accurately the spatial relation between entities A and B .

Topological relations such as *inside*, *contains*, *intersects*, and *meets* (adjacency) often become redundant once combined with cardinal directions. For example, for two example RDF triplets $\langle o_1 \rangle \langle inside \rangle \langle o_2 \rangle$ and $\langle o_2 \rangle \langle south \rangle \langle o_3 \rangle$, then o_1 inherits that directional relation from o_2 to o_3 , generating $\langle o_1 \rangle \langle south \rangle \langle o_3 \rangle$. The same goes

Table 5.1: Relation composition matrix with a right arrow ($\rightarrow$) denoting left-to-right composition.

$\rightarrow$	N	NE	E	SE	S	SW	W	NW	inside	contains	intersects
N	N	NE	NE	E	N	W	NW	NW	N	N	N
NE	NE	NE	E	E	E	N	N	N	NE	NE	NE
E	NE	E	E	SE	SE	SE	N	N	E	E	E
SE	E	E	SE	SE	S	S	E	E	SE	SE	SE
S	S	E	SE	S	S	S	SW	W	S	S	S
SW	W	S	S	S	SW	SW	W	W	SW	SW	SW
W	NW	N	N	E	SW	W	W	NW	W	W	W
NW	NW	N	N	E	W	W	NW	NW	NW	NW	NW
inside	N	NE	E	SE	S	SW	W	NW	inside	???	???
contains	N	NE	E	SE	S	SW	W	NW	???	contains	???
intersects	N	NE	E	SE	S	SW	W	NW	???	???	???

for the *contains* relation. When not combined with a cardinal direction, the intersection/containment relations usually carry on, as long as they remain the same. For example, if o_1 is inside of o_2 and o_2 is inside of o_3 , then o_1 is inside of o_3 as well. In our case, through *SpaTex*'s global grid and per-cell processing of neighboring entities, all non-disjoint topological relations will be generated and explicitly stored, meaning that they won't have to be inferred. Hence, SpaRAGraph simplifies paths by discarding topological relations when they co-occur with directional ones, preserving only the latter.

On the other hand, cross-topological combinations are much harder to deduce accurately. For example, for two RDF triplets $\langle o_1 \rangle \langle \textit{inside} \rangle \langle o_2 \rangle$ and $\langle o_2 \rangle \langle \textit{contains} \rangle \langle o_3 \rangle$, the combined relation $\textit{inside} \rightarrow \textit{contains} = p_c$ does not, by itself, convey meaningful information about the relation between o_1 and o_3 . We can disregard cases where $p_c = \textit{inside}$, as *SpaTex* would have already computed and explicitly stored the RDF triplet $\langle o_1 \rangle \langle \textit{inside} \rangle \langle o_3 \rangle$. The same applies to the *contains* relation. Additionally, this composition indicates that both o_1 and o_3 are inside the same, larger entity o_2 . For their explicit relation not to exist in the graph, they have to be far away enough so that they do not fall inside the same cell of *SpaTex*'s grid as well. Aside from this, it remains unclear what the correct relation p_c is, based solely on the given pair of RDF triples.

Another special case is the *intersects* relation, which does not convey much useful information in combination with other topological relations. For example, the composed relation between two RDF triplets $\langle o_1 \rangle \langle intersects \rangle \langle o_2 \rangle$ and $\langle o_2 \rangle \langle intersects \rangle \langle o_3 \rangle$ can not be *intersects*, *contains* or *inside*, otherwise it would have been explicitly generated by *SpaTex*. On the other hand, for such case to appear, o_2 must have large extent, so that o_1 and o_3 intersect with it inside different cells in *SpaTex*’s grid, otherwise their relation would have been stored explicitly. In such cases, entities o_1 and o_3 may only be related by a cardinal direction, which cannot be deduced based solely on this pair of RDF triplets.

These rare, but possible, edge cases are labeled with three question marks (“???”) in the composition matrix, as they require specialized handling by SpaRAGraph. We elaborate on how such cases are managed in Section 5.3.2. Moreover, as discussed in Section 5.2.1, defining the relative direction between two entities using a single cardinal direction term is often only partly accurate and inherently approximate. This approximation carries over into the estimations performed during the compositions shown in Table 5.1.

5.3 Spatial Reasoning with SpaRAGraph

In this section, we detail how SpaRAGraph utilizes the graph index constructed over the spatial RDF data to generate relevant context for a given spatial question during inference. The process consists of two main stages: First, SpaRAGraph identifies the spatial entities referenced in the question and traverses the RDF graph to determine their relative spatial relation. Then, it synthesizes this information into a coherent context, which is appended to the original question and provided to the language model to support accurate and informed reasoning.

5.3.1 Graph Traversal

Named Entity Recognition When a spatial question is posed, the first step is to identify the entities it references. We use spaCy [132] for this NER task. spaCy must be configured to recognize the types of entities present in the RDF data. For example, in our experimental evaluation, we use data from States, Counties, and Zipcodes in the U.S., so we customized spaCy’s rules to distinguish between these entity types

and accurately identify them. Note that the extracted entities are returned in the order they appear in the user’s prompt. More advanced recognition rules can be defined to further guide SpaRAGraph’s graph traversal and context generation, such as identifying specific relations to target in the graph or retrieving multiple paths between two entities. However, in this initial version of SpaRAGraph, we simply use spaCy to extract all referenced entities.

Entity-Node Matching Once the referenced entities are extracted, they must be matched to the corresponding nodes in the RDF graph. However, node names are stored as URIs, which may not match the way users refer to the same entities. For instance, a user might refer to “The State of Kansas” as “Kansas State”, simply “Kansas”, or by its abbreviation, “KS.” To bridge this gap, we perform entity-to-node matching via similarity search. All node labels are indexed using FAISS [136] to enable fast approximate similarity search. Each extracted entity is then queried against this index, and the most similar node label is returned as its match.

Path Search The identified entities are then paired into $(start, end)$ tuples, defining the list of paths to be searched in the RDF graph. A path p in a graph $G = (V, E)$ is defined as $p = (v_0, v_1, \dots, v_k)$ such that all edges (i.e. pairs of vertices) $v_i, v_{i+1} \in E$ for all $i = 0, 1, \dots, k - 1$. Note that, due to the way *SpaTex* computes spatial relations, a given pair of nodes can have at most one edge in each direction between them. Depending on the task, multiple paths —possibly in both directions— may need to be considered. To determine the spatial relation between two entities, a single path may suffice. However, when querying the relation of one entity with multiple others, it is necessary to traverse additional paths to comprehensively capture all relevant relations. For a given pair $(subject, object)$, SpaRAGraph first attempts to find the shortest path (the one with the fewest edges) from *subject* to *object*. We use Breadth-First Search (BFS) because all edges in the graph are considered to have equal weight, making it effectively an unweighted graph. BFS is optimal for such cases, as it guarantees the discovery of the shortest path in terms of the number of edges (i.e., the minimal number of intermediate RDF triples) from the *subject* to the *object*. If no path is found, it then searches in the reverse direction, from *object* to *subject*. If no path exists in either direction, no contextual information is generated for that entity pair. For questions involving multiple entities, paths for all possible $(start, end)$ combinations may need to be computed, depending on the task. In this version of SpaRAGraph, the path search strategy is defined by the task type. In our

experimental evaluation, we test SpaRAGraph on binary, multiclass, and multilabel classification tasks. For binary classification, we extract a single (*subject, object*) pair based on the order of appearance in the question. If no path is found from subject to object, we attempt the reverse direction, from object to subject. We follow the same strategy for multiclass classification, where each question involves two entities and a set of relation options, only one of which is correct. Again, we search for a path from subject to object and fall back to the reverse if necessary. In multilabel classification, each question involves a single subject entity and multiple candidate object entities. The task is to identify all (if any) object entities that satisfy the specified relation in the question. In this case, we compute paths from the subject to each candidate object. If no path is found in that direction, we then check the reverse direction on a per-pair basis.

5.3.2 Context Generation

At this stage, SpaRAGraph has identified a list of paths between the entities mentioned in the question. For a path p from the subject to the object, each hop in the path corresponds to a spatial RDF triple. This creates a natural, sequential order in which to process the triples (from subject to object) in order to determine their overall spatial relationship. The next step is to generate context based on the identified path(s) and use it to augment the user’s question, providing the LLM with relevant spatial information to guide its response so that it answers correctly.

We semantically evaluate a path through deterministic, rule-based means, following the relation composition matrix (Table 5.1). Figures 5.8 and 5.9 show two end-to-end examples of how graph traversal and context generation is done for two spatial questions on the example data and graph of Figure 5.7. In Figure 5.8, the spatial relation between entities A and D is not explicitly stored as an RDF, so the path from A to D is longer than one hop. Combining the *inside* and *northwest* relations that A and D have with intermediate entity C respectively, leads to the conclusion that A is northwest of D (as the ‘inside’ topological relation is discarded when combined with a directional one). We can verify from the map that entity A is indeed northwest of entity D, confirming that the generated context is accurate. This context is then provided alongside the original spatial question to the LLM, guiding its response with factually grounded information about the referenced entities. In the example of

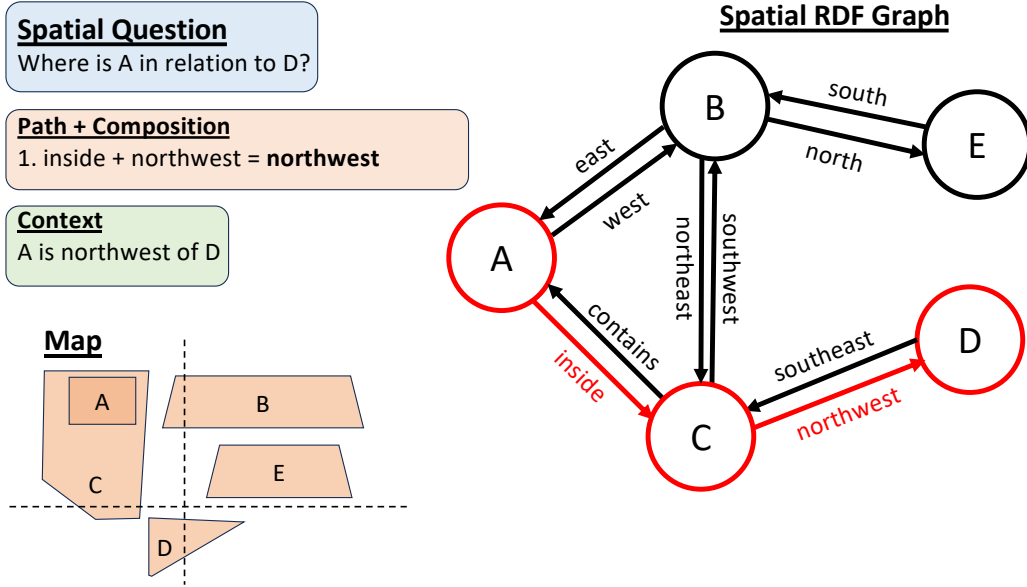


Figure 5.8: Path traversal and context generation for an example spatial question based on the data in Figure 5.7. SpaRAGraph identifies a 2-hop path, performs one directional composition, and generates context that accurately captures the spatial relation between the entities based on the map.

Figure 5.9, the path from entity D to entity E traverses through entities C and B, composing each spatial relation step-by-step and ultimately concluding that entity D is south of entity E. This result is approximately correct, offering the LLM a useful hint about D’s location relative to E. However, the more precise relation would be that D is southwest of E. This detail cannot be inferred from the available data without directly accessing the entities’ geometries.

Furthermore, due to the structure of our spatial RDFs and their indexing in the graph, adjacency relations (*meets*) typically become obsolete beyond the first hop in the path. This is because all adjacency relations are explicitly captured in the graph, as *SpaTex* groups nearby entities into grid cells and computes relations among neighboring entities. Hence, the shortest path in the graph between two adjacent entities will be a single hop, namely, the RDF that describes their adjacency and relative direction to one another.

We handle composition edge cases (marked as “???” in Table 5.1) as follows: if the shortest path between two nodes consists solely of undetermined compositions, SpaRAGraph iteratively searches for the next shortest path, continuing this process until it finds a path that includes at least one valid composition (up to a threshold). The generated context then includes both the summarized relation between the start

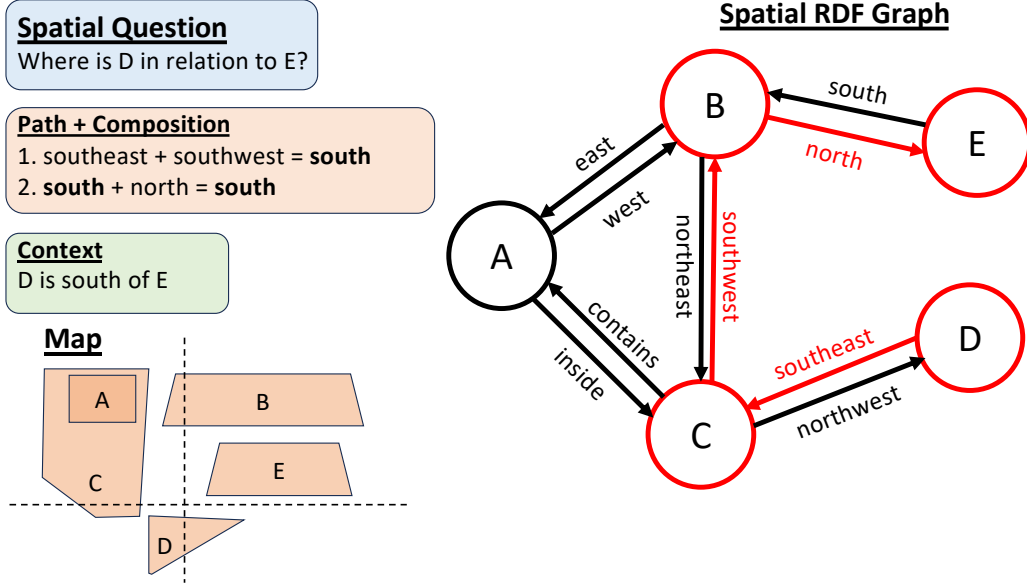


Figure 5.9: Path traversal and context generation for an example spatial question based on the data in Figure 5.7. SpaRAGraph identifies a 3-hop path, performs two directional compositions, and generates context that is approximately correct for entities D and E. in reality, D lies southwest of E based on the map.

and end nodes, as determined by the top k shortest path, and all RDF triplets from the top $k - 1$ paths as additional context. This approach ensures that the LLM receives as much relevant information as possible, even in the presence of incomplete or ambiguous composition data.

Based on how cardinal direction relations are computed by *SpaTex* and the inherently approximate composition matrix, our External Reasoning approach is by extension approximate as well; however, it provides a fast and efficient way to semantically compose spatial relations over text, enabling the generation of contextual information that can assist an LLM in answering spatial questions more accurately.

5.4 SRB - The Spatial Reasoning Benchmark

Evaluating the generation performance of a retrieval-augmented method such as SpaRAGraph on real-world spatial data requires two components: (i) datasets that are loaded and query-ready in an external database (in our case, RDF triplets that have been generated and indexed in the graph), and (ii) a set of question-answer pairs that either directly relate to the data in the database or are at least relevant to

it. Existing spatial reasoning benchmarks, such as StepGame [109], do not involve real-world, globally positioned spatial entities. As a result, they are not well-suited for evaluating RAG approaches, unless a small, per-question index is artificially constructed. However, such setups fall short of meaningfully testing a method’s scalability and retrieval capabilities.

To thoroughly evaluate SpaRAGraph, we introduce three datasets comprising natural language spatial questions based on real-world geographic divisions in the United States, specifically States, Counties, and ZIP Codes (hereafter referred to as Zipcodes). These questions evaluate topological relationships to benchmark methods that enhance spatial inference in large language models. The three datasets test binary, multiclass, and multilabel classification tasks, respectively, using natural language texts containing spatial knowledge.

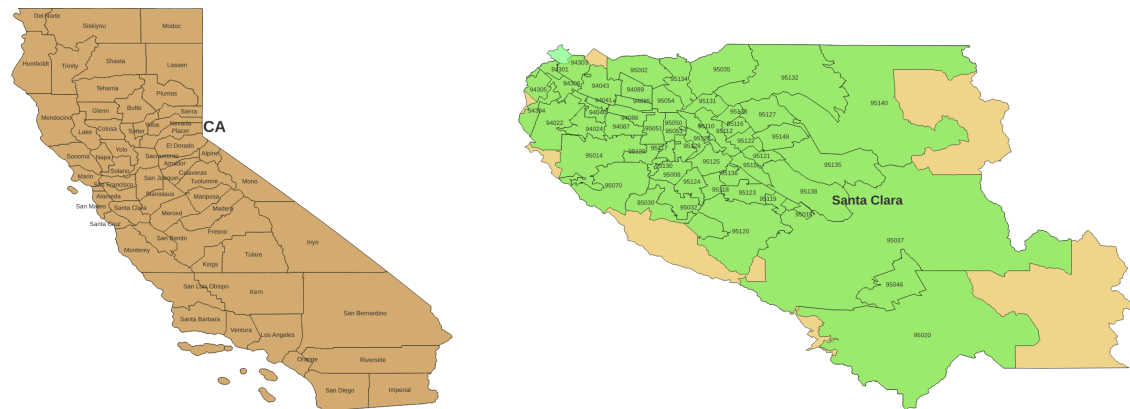
Our generated benchmarking datasets comprise semantically meaningful questions (and answers), which are produced from raw spatial data. Specifically, the questions focus on the relative locations of geographic divisions within the United States, such as states and counties. Each dataset contains 1,000 questions of the same type (Yes/No, Single-Choice, or Multi-Choice) and includes the correct answers for training and evaluation purposes. Table 5.2 summarizes these question types along with an example for each case.

5.4.1 Source Data

We used the TIGER 2015 real-world datasets [51] for U.S. States (50 entities), Counties (3,225 entities), and Zipcodes (33,144 entities) as the foundation for generating our spatial questions. Figure 5.10 presents example entities from these datasets to illustrate the relative scale of the different geographic divisions. All entities have non-overlapping areas within the same type (e.g., State borders do not intersect with other States), as these geographic divisions are defined by official geopolitical boundaries established by governmental authorities. In the United States, such boundaries are determined and maintained by entities like the U.S. Census Bureau and state legislatures, ensuring that units like Zipcodes, Counties, and States are usually hierarchically ordered and divisions of the same type are mutually exclusive in area.

Table 5.2: Overview of the different question types in the dataset, each illustrated with an example.

Type	Example Question	Example Answer(s)
Yes/No	Is Zipcode 62546 northeast of Zipcode 62560?	yes
Radio	Select exactly one option (a-e) that best describes the relationship of Douglas County Washington in relation to Zipcode 98848 in terms of geography. Options: a. contains b. adjacent to and northwest of c. adjacent to and north of d. southeast of e. none of the above	e
Checkbox	Select all options that are adjacent to and west of The State of Illinois? You may choose one or more options. Options: a. Zipcode 47993 b. Zipcode 47991 c. Zipcode 63782 d. Zipcode 62324 e. None of the above	a,b



(a) The State of California and its Counties (b) Santa Clara County, CA and its Zipcodes

Figure 5.10: Illustrative example showing the State of California and its Counties (a), with a focus on the Zipcodes within Santa Clara County, CA (b), to highlight the relative scale of the geographic divisions in our source data. (Visualized on QGIS [2].)

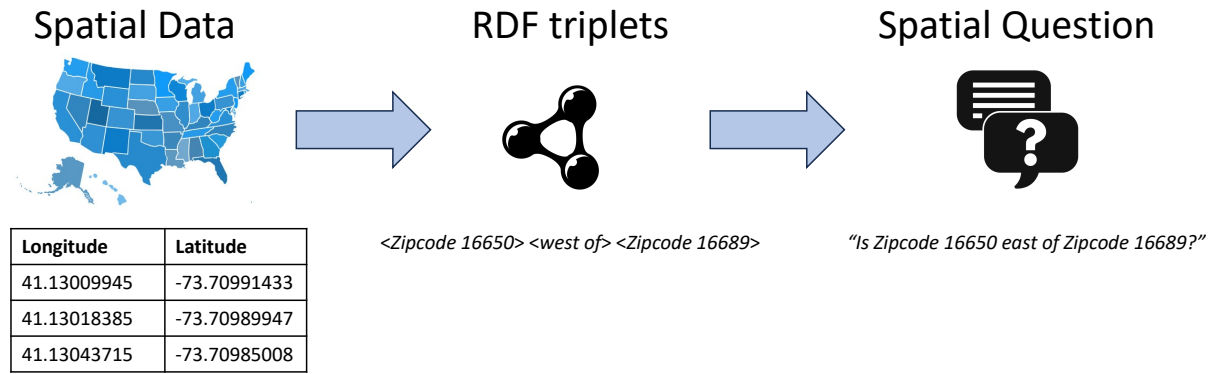


Figure 5.11: Overview of our pipeline for generating spatial questions from raw geographic data. The process involves calculating spatial relationships and transforming them into RDF triplets using *SpaTex*. Then, structured yes/no, single-choice, and multiple-choice questions are randomly generated from the set of RDF triplets.

5.4.2 Construction

To generate semantically meaningful questions grounded in the data stored in our database, we leverage the same set of RDF triples produced by *SpaTex* during SpaRA-Graph’s pre-processing phase. From these triplets, we automatically generate randomized natural language questions tailored to binary, multiclass, and multilabel classification tasks, all framed as question–answer (QA) pairs. An overview of this process is shown in Figure 5.11. The first step in the pipeline (transforming spatial data into RDF triplets) is performed by *SpaTex*.

5.4.2.1 Yes/No - Binary Classification

Figure 5.12 illustrates the generation process of questions with Yes/No answers. For a randomly selected triplet out of the available ones (e.g., the leftmost box in Figure 5.12), we assign a Yes/No answer with 50-50 chance. For questions with a positive answer, we simply transform the RDF triplet in the form of question. For example, the triplet `<Santa Clara County California> <south> <Alameda County, California>` becomes the question: *Is Santa Clara County, California south of Alameda County, California?* For generating questions with a negative answer, we retain the same entities but alter the relation to its semantic opposite. In the case of cardinal directions, we use the inverse (e.g., ‘north’ becomes ‘south’, ‘southwest’ becomes ‘northeast’). For topological relations, we replace each with its logical counterpart. For example, ‘contains’ becomes ‘is inside of’ and vice versa. This approach ensures that the resulting

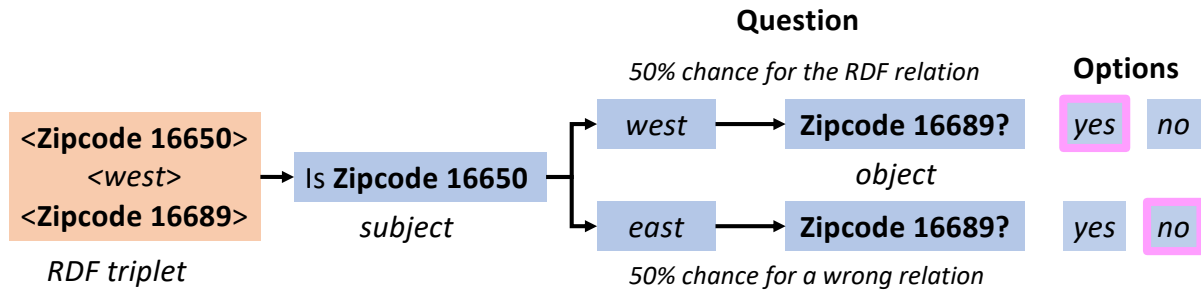


Figure 5.12: Example of Yes/No random question generation from a randomly sampled RDF triplet. The correct answer is indicated with a bolded boundary.

question is definitely false while preserving grammatical and semantic coherence. The phrasing of the questions is rule-based, ensuring that all questions follow a consistent syntactic structure.

5.4.2.2 Radio - Multiclass Classification

To generate multiple-choice questions, we begin by randomly sampling a triplet from the available dataset. Each question includes five answer options, labeled (a) through (e), with only one being correct based on the sampled triplet. Option (e) is always reserved for ‘none of the above’, allowing for cases where the correct answer is intentionally excluded from the listed options.

We introduce such cases with a 20% probability. In these instances, four incorrect relations are randomly selected with equal probability from the available pool (excluding the correct one), shuffled, and assigned to options (a) through (d), making (e) the correct answer. In the remaining 80% of the cases, where the correct relation is included, we randomly select three incorrect relations (excluding the correct one), shuffle them along with the correct relation, and assign them to options (a) through (d), with (e) remaining as ‘none of the above’. This approach ensures that the correct answer may appear in any of the five options.

The phrasing of Radio questions is always in the following format: *Select exactly one option (a-e) that best describes the relationship of <subject> in relation to <object> in terms of geography. Options: a. <relation₁> b. <relation₂> c. <relation₃> d. <relation₄> e. none of the above.* In this format, the instruction is embedded within the question itself, while the answer choices are presented separately, with exactly one correct option among (a) through (e).

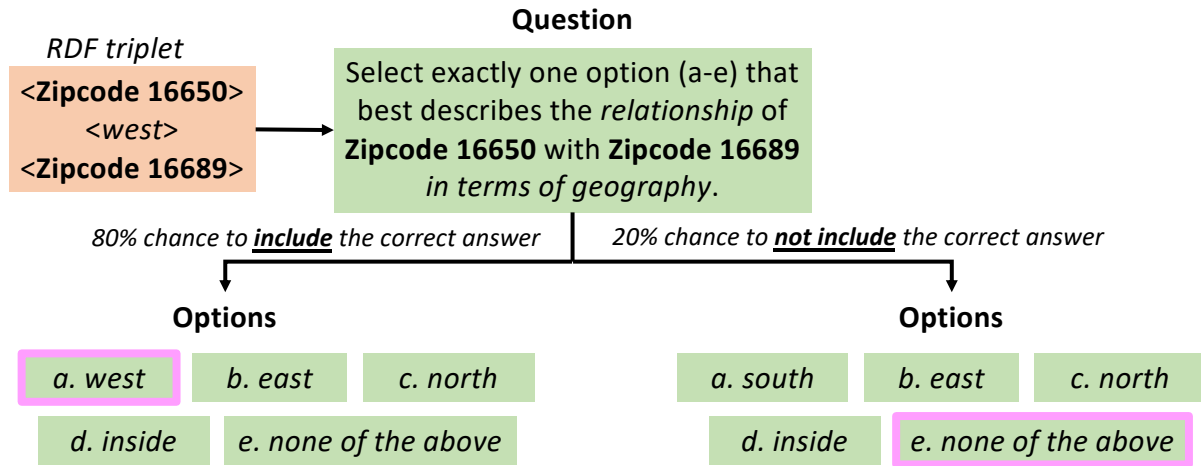


Figure 5.13: Radio (single-choice) random question generation from a randomly selected RDF triplet. The correct answer is indicated with a bolded boundary.

5.4.2.3 Checkbox - Multilabel Classification

For checkbox queries, we first construct a relation map for each subject entity before sampling a random triplet. Specifically, given an RDF triplet <subject> <predicate> <object>, we identify all objects in the dataset that share the same relation *predicate* with the *subject*. This results in a grouped set of related entities per relation, making it easier to identify and use them as multi-label answer options.

Next, we sample a random triplet and use the relation map to select a random relation for the triplet’s subject, along with all objects that share that relation with the subject. This set forms the pool of correct answer options for the query. To construct the incorrect options, we gather all other objects that are related to the same subject via a different relation. These objects serve as distractors, as they do not satisfy the target relation specified in the sampled triplet.

We then randomly determine the number of correct answers c to include in the question, selecting between 0 and 4 options from the pool of correct entities. If the pool contains fewer than c entities, c is reduced incrementally until sampling is possible. The pool is guaranteed to contain at least one entity; the object of the RDF triple that initially introduced the target relation into the pool.

The remaining options are filled with randomly chosen distractors from the incorrect set, and all answer choices are shuffled. Option (e) is always ‘none of the above’, enabling the inclusion of questions with no correct answer. For example, Figure 5.14 illustrates a case in which an entity (Zipcode 16650) appears as the subject

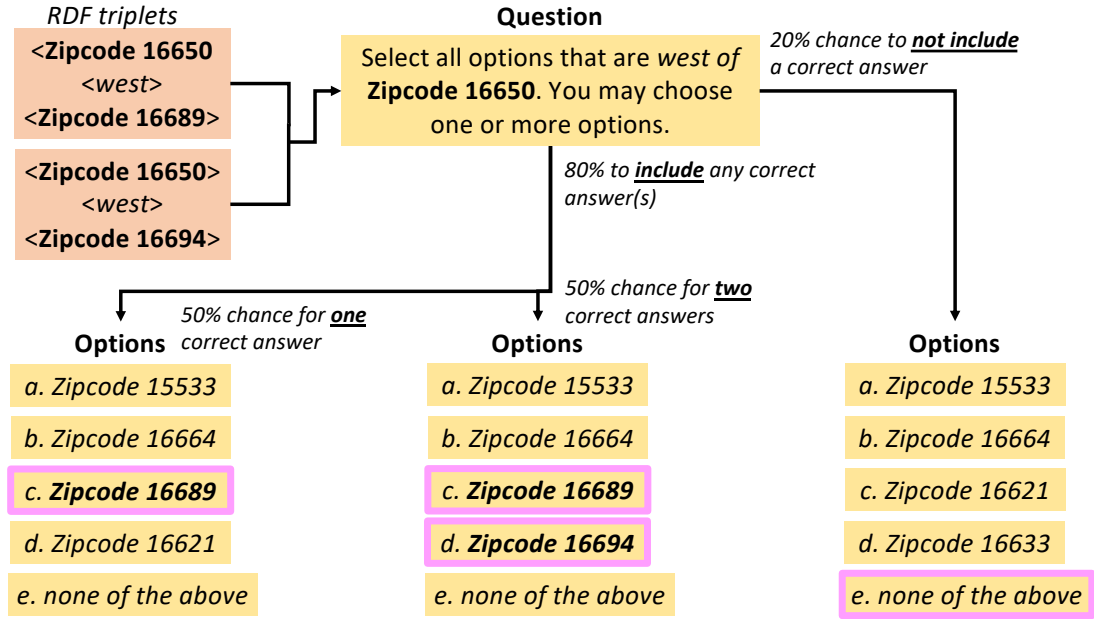


Figure 5.14: Checkbox (multiple-choice) random question generation from all triplets regarding a randomly selected entity (Zipcode 16650). The correct answers are shown with a boundary.

in two triplets involving the *west* relation. Based on these triplets, a checkbox query is randomly generated to include zero, one, or both of the correct objects among the answer options.

5.5 Experimental Evaluation

Question Answering (QA) To assess the performance of SpaRAGraph, we evaluate it on binary, multiclass, and multilabel classification tasks using SRB, introduced in Section 5.4. For each model, we run every question three times and retain the most representative response, as smaller models often produce varied outputs across runs. Evaluating the spatial inference capabilities of SpaRAGraph on open-ended, complex spatial questions is inherently challenging, as it would require textual ground truth annotations and expert evaluators to assess the correctness of generated responses. To date, no evaluation framework has been established with proven effectiveness for open-ended spatial reasoning tasks. Additionally, the StepGame benchmark [108] is comprised of randomly generated, locally-scoped synthetic entities, which prevents the use of a global index (such as our RDF graph) for capturing spatial relations at a broader, global level. Moreover, StepGame focuses solely on multilabel classification.

Table 5.3: Breakdown of question datasets by type of entities referenced in questions.

	States only	Counties only	Zipcodes only	State-County	State-Zipcode	Zipcode-County
Yes/No	N/A	3.5%	56.2%	0.8%	9.6%	28.0%
Radio	0.1%	5.7%	55.0%	1.3%	9.8%	27.1%
Checkbox	12.4%	35.6%	50.5%	N/A	N/A	N/A

Hence, SRB provides a practical foundation for preliminary yet detailed evaluation, focusing on structured but non-trivial tasks such as binary, multiclass, and multilabel classification over real-world ontologies.

Subject Entities States in the U.S. cover much larger areas than Counties or Zipcodes, and their names are more widely recognized beyond local contexts. Consequently, they are more likely to be referenced in bibliographic sources, online content, or other types of text. Such widely available geographic information increases the likelihood that topological relationships between states appeared in the documents used to train certain foundation models. Table 5.3 breaks down the datasets by the types of entities referenced in the questions. Yes/No and Radio questions always involve a relation between two entities (which may be of the same or different types), whereas Checkbox questions refer to a single entity (see Figures 5.12-5.14). Additionally, approximately half of the questions in each dataset pertain to Zipcodes, which is due to the uniform sampling from the source data to generate the questions, which comprise a significantly higher number of Zipcodes (33,144) compared to Counties (3,225) or States (50).

Embeddings & Indexing We implement SpaRAGraph in Python 3.9. All RDF-related operations such as reading, writing, indexing the graph and path finding, are handled using the RDFLib library. To embed both the RDF graph’s node labels and the extracted entities from user questions, we employ Sentence Transformers [137], specifically the all-MiniLM-L6-v2 model. For approximate k-nearest neighbors (akNN) search over these embeddings, we use FAISS [136] for efficient indexing and similarity-based querying.

Models & Setup In our experiments, we test the models shown in Table 5.4 employed with and without SpaRAGraph. All our experiments were conducted single-threaded on a machine with Ubuntu 20.04 OS, 64GB of memory and an Intel Core i9 3.60GHz processor. The models were loaded in an NVIDIA GeForce RTX 3060 with 12GB of memory.

Table 5.4: The models that we tested on SpaRAGraph.

Model	# of Parameters	Quantization
meta-llama/Llama3.1-8B-Instruct [138]	8B	4 bits
mistralai/Mistral-7B-Instruct [139]	7B	4 bits
Qwen/Qwen2.5-7B-Instruct [140]	7B	8 bits

To ensure statistical reliability, all generation experiments were conducted three times, with the reported scores representing the average across these runs. The total execution times for running each dataset once with and without SpaRAGraph is shown in Table 5.5. Notably, the task complexity affects the total execution time, as simpler tasks such as Yes/No questions are processed more quickly, while more complex formats like Radio and Checkbox questions require longer inference times. Using SpaRAGraph introduces a time overhead of approximately 6% to 29%. This increase is partly due to the RAG process (which includes the NER, node matching, graph traversal, and context generation) and partly due to longer generation times caused by the expanded prompt size.

Most individual components in SpaRAGraph can be independently optimized, further enhancing overall performance thanks to its highly modular architecture. For instance, incorporating state-of-the-art graph indexes and path traversal algorithms can significantly reduce the time required for context generation. Moreover, both accuracy and efficiency can be improved by integrating more advanced NER or similarity search techniques. Ultimately, the primary bottleneck in the pipeline tends to be the model’s response generation, rather than any of SpaRAGraph’s internal modules.

5.5.1 SpaRAGraph Generation Evaluation

To evaluate model performance across the three classification tasks, we apply task-appropriate metrics. For binary (Yes/No) questions, we report Precision (P), Recall (R), and F1 score. For multiclass (Radio) questions, where only one option is correct, we use macro-averaged Precision, Recall, and F1 score to account for class imbalance. For multilabel (Checkbox) questions, we adopt a sample-based evaluation: each question is treated as a separate sample, and Precision, Recall, and F1 score are computed by comparing the predicted set of labels to the ground truth for that sample. These metrics are then averaged across all samples, allowing us to assess how accurately

Table 5.5: Comparison of models’ average execution time after three runs, in seconds $\pm$ the standard error of the mean rounded up, on the Spatial Reasoning Benchmark, with and without SpaRAGraph in zero-shot setting.

Model	Yes/No (✓/✗)	Radio (●)	Checkbox [x]
Llama-3.1-8B-Instruct	248 $\pm$ 2	286 $\pm$ 4	434 $\pm$ 4
+ SpaRAGraph	278$\pm$2	328$\pm$2	473$\pm$1
Mistral-7B-Instruct	315 $\pm$ 1	360 $\pm$ 1	509 $\pm$ 1
+ SpaRAGraph	334$\pm$2	429$\pm$1	655$\pm$1
Qwen2.5-7B-Instruct	316 $\pm$ 1	328 $\pm$ 4	461 $\pm$ 2
+ SpaRAGraph	351$\pm$1	381$\pm$3	544$\pm$5

the model predicts multiple relevant labels per question, while also capturing partial correctness. Model generation temperature was set to 0.7 across all runs.

5.5.1.1 Effectiveness

We experimentally demonstrate the effectiveness of SpaRAGraph on all models by comparing its response accuracy across all three tasks to the accuracy of the same models when used out-of-the-box, in a zero-shot setting. Table 5.6 summarizes the performance of all evaluated models across the three tasks, highlighting the significant improvements achieved when augmented with SpaRAGraph.

In particular, all models exhibit the most substantial gains on the Yes/No (binary) classification task achieving near-perfect F1 scores compared to their standalone performance, which ranges from 35% to 61%. In Radio questions, which pose a greater challenge for LLMs than binary (Yes/No) queries due to their multi-class, single-answer nature, the improvements introduced by SpaRAGraph are great for both Llama-3.1-8B and Qwen2.5-7B. However, Mistral-7B continues to struggle even when assisted by SpaRAGraph, suggesting that the increased complexity of selecting a single correct option among several may exceed its reasoning capabilities, despite the additional context. A similar trend is observed in Checkbox questions, where

Table 5.6: Performance comparison of models on the Spatial Reasoning Benchmark, with and without SpaRAGraph in zero-shot setting. The scores (%) are the averages across three different response generation cycles and the standard error of the mean across all scores does not exceed 1%.

Model	Yes/No (✓/✗)			Radio (●)			Checkbox [x]		
	P	R	F1	macro-P	macro-R	macro-F1	sample-P	sample-R	sample-F1
Llama-3.1-8B-Instruct	50	80	61	23	21	17	24	50	30
+ SpaRAGraph	94	98	96	73	70	70	44	68	51
Mistral-7B-Instruct	52	50	51	23	21	14	24	45	29
+ SpaRAGraph	93	98	96	57	34	31	30	58	37
Qwen2.5-7B-Instruct	83	22	35	27	27	26	34	50	39
+ SpaRAGraph	100	97	98	83	82	76	72	85	76

the improvements from SpaRAGraph are somewhat smaller but present. Overall, Qwen2.5-7B consistently outperforms the other models and has the highest performance gains when employed with SpaRAGraph, highlighting the effectiveness of the generated context in guiding its responses more accurately.

5.5.1.2 Zero- vs Few-shot

We evaluated SpaRAGraph under zero-shot, one-shot, two-shot, and three-shot settings using prompts with static examples (shots). Due to the nature of our tasks and the inherent structure of spatial relations, these examples do not function as context in the conventional sense, meaning that they lack factual information about the specific entities involved in the test queries. Instead, the examples are intended to help models infer relational patterns, such as recognizing inverse spatial relationships. For instance, a static example might be: “*Entity A is south of Entity B. Where is B relative to A? // B is north of A.*” Figure 5.15 illustrates the limited benefits (that worsen as the number of examples increases) which few-shot learning brings when combined with SpaRAGraph. While static examples can occasionally guide the models toward generating correct responses, they generally lack the contextual grounding that SpaRAGraph provides. In many cases, these examples confuse the model more

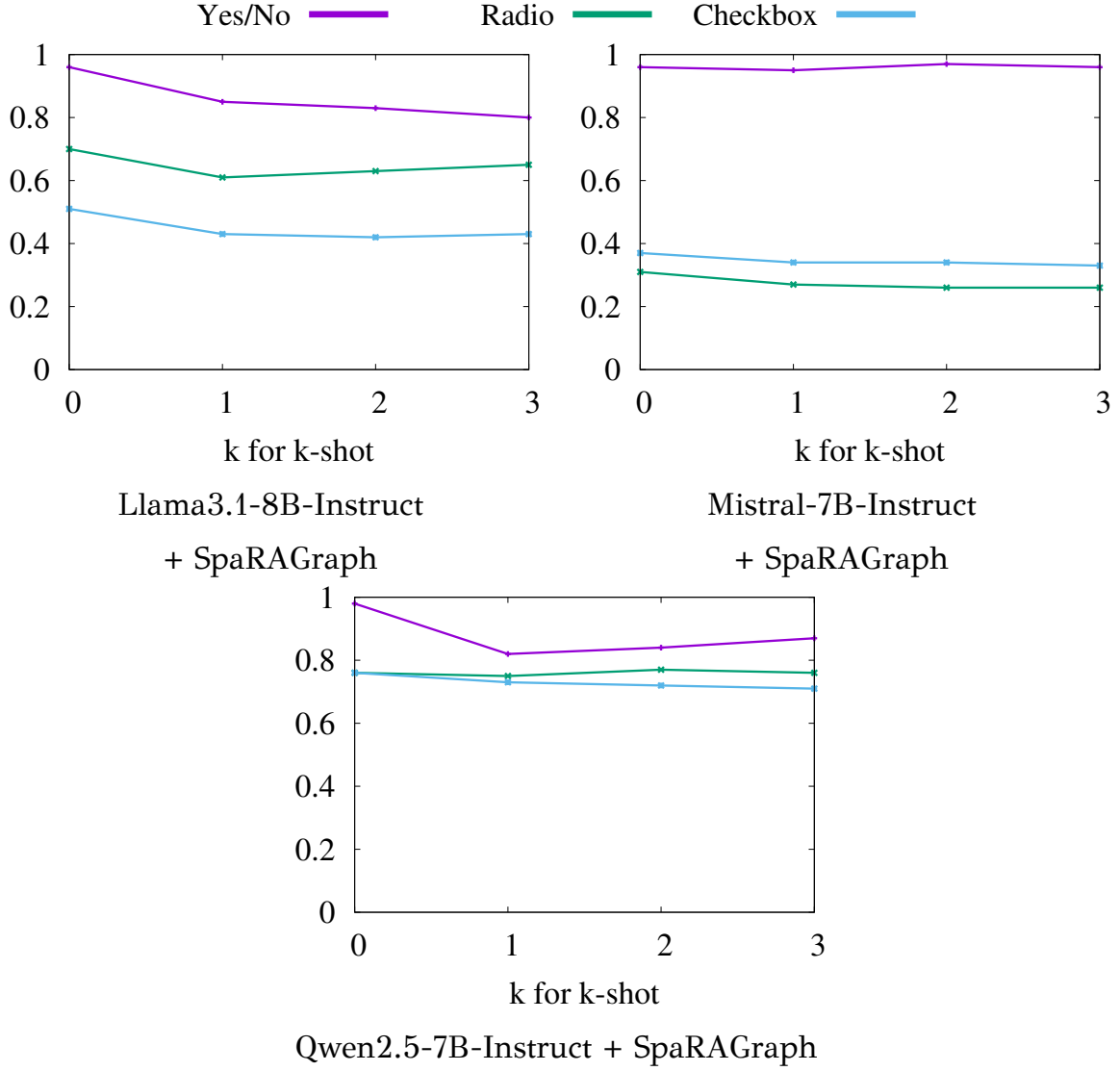


Figure 5.15: F1, macro-F1 and sample-F1 scores (Y axis) of each model for Yes/No, Radio and Checkbox questions, respectively, in k -shot settings ($k \in [0, 3]$).

than they help. We omit results from few-shot learning with randomly selected examples, as these are almost sure to introduce irrelevant information to each question, further impairing model performance.

In general, SpaRAGraph is specifically designed to be effective with out-of-the-box, untuned models operating in a zero-shot setting. Its structured guidance and contextual grounding are sufficient to elicit accurate and relevant responses from the model without the need for additional examples. As such, few-shot learning becomes unnecessary, as SpaRAGraph already provides the essential task understanding and relational context that examples would otherwise aim to convey. However, as spatial

questions grow more complex, few-shot learning may offer benefits by guiding the model on how to reason with the generated context.

5.5.2 Retrieval Accuracy

SpaRAGraph’s retrieval accuracy primarily depends on two factors: a) the effectiveness of NER, i.e., extracting all relevant entities from a prompt, and b) the accuracy of matching these entities to their corresponding nodes in the graph via similarity search. For NER, we use spaCy, which supports custom rule-based entity extraction tailored to the specific characteristics of the data. For similarity search, we leverage FAISS, a library known for its high-performance and accurate vector similarity search, enabling fast and precise entity-node matching in the graph. The tasks of accurately extracting entities from text and identifying the most similar nodes in the graph are orthogonal to SpaRAGraph’s core functions of path traversal and context generation. Therefore, state-of-the-art solutions (such as spaCy for NER or FAISS for similarity search) can be integrated to minimize retrieval inaccuracies without interfering with the framework’s primary operations. This compartmentalization allows for easier tuning of the various components in SpaRAGraph.

To thoroughly evaluate SpaRAGraph on retrieval accuracy, we employ three key metrics:

- Mean Reciprocal Rank (MRR) – This metric assesses the rank of the ground truth (i.e., the correct entities) within the list of retrieved entities. It is calculated as the reciprocal of the rank of the first correct entity, averaged across all questions.
- Full Match Rate (FMR) – This measures the percentage of prompts for which all related entities were correctly retrieved, with no omissions.
- Precision – This reflects the proportion of retrieved entities that are correct, indicating the accuracy of the results with respect to the entities relevant to each prompt.

Table 5.7 presents the performance of SpaRAGraph in accurately retrieving the target entities used for graph traversal across each dataset in the SRB. SpaRAGraph achieves near-perfect scores in MRR, FMR, and Precision across all datasets, demonstrating robust entity matching performance, even in scenarios where entities have

Dataset	MRR	FMR	Precision
Yes/No	0.99	0.97	0.99
Radio	1.00	0.98	0.99
Checkbox	1.00	0.97	0.99

Table 5.7: SpaRAGraph’s retrieval accuracy expressed through its MRR, FMR (percentage of prompts that had all their entities retrieved correctly) and Precision (percentage of all retrieved entities that are correct for their respective prompts). Metrics are shown per individual dataset, as Yes/No and Radio questions always regard two entities each, whilst SRB’s Checkbox questions always pertain five entities.

highly similar names (e.g., Zipcode 00951 vs. Zipcode 00952). The very few retrieval errors observed typically involve such edge cases. For instance, in questions involving counties in West Virginia, the entity recognizer (spaCy) may include the state name alongside county names, resulting in additional graph paths being explored and introducing noise into the generated context. Even these few issues, however, could be easily mitigated through refinements, e.g., calibrating spaCy or adopting more standardized naming conventions for RDF subjects and graph nodes.

Optimizing NER and entity-to-node similarity search is beyond the scope of this work. Nonetheless, existing state-of-the-art solutions for both tasks are readily compatible with SpaRAGraph and can be integrated to further enhance its performance.

5.6 Conclusions

This chapter introduced SpaRAGraph, a novel approach for enhancing the ability of LLMs to answer spatial queries using RAG. Our method is applicable to a wide range of spatial data, including existing RDF datasets, as long as their predicate transitions can be represented using a transition matrix or a suitable algebraic framework. Our experimental analysis shows that SpaRAGraph improves the response generation of three LLMs for spatial questions by 35-63 points (47 average) in binary, 17-53 points (40 average) in multiclass and 8-37 points (23 average) in multilabel classification tasks.

We also introduced the first spatial reasoning datasets, along with a benchmark designed to evaluate model performance on binary, multiclass, and multilabel clas-

sification tasks involving spatial questions. Our datasets are grounded in accurate, real-world topological relationships among States, Counties, and Zipcodes in the U.S., enabling robust evaluation of spatial reasoning capabilities in large language models in both RAG and non-RAG scenarios.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Summary of Contributions

6.2 Directions for Future Work

In conclusion, we present a summary of this dissertation’s most significant contributions and outline potential directions for future work.

6.1 Summary of Contributions

In this dissertation, we studied the challenges of managing complex, non-point data types such as high-complexity polygons. The goal was to design new techniques of indexing, approximating, evaluating and storing such geometries to improve spatial query efficiency and performance. At the same time, we focused on the scalability of our approaches, both vertically and horizontally in distributed spatial data management environments, as well as on a per-object basis by reducing processing costs as shapes grew more complex. Additionally, we explored how these techniques can be applied both in traditional spatial DBMS scenarios and in RAG mechanisms to enhance LLMs’ spatial reasoning capabilities.

Spatial Joins with Complex Geometries In the first part of this thesis, we focused on spatial joins between complex geometries, which are inherently computationally expensive. To address this, we proposed a new spatial approximation technique, along with its extended version (Raster Intervals and APRIL), that can be used during query

processing to (i) avoid direct computation on geometries, (ii) detect results and non-results early, and (iii) minimize memory requirements, enabling their practical use in real-world spatial DBMS. Additionally, we proved how these approximations can support multiple topological predicates in spatial joins, as well as specialized cross-type queries such as polygon–linestring joins.

Scalable Spatial Data Management In the second part of this thesis, we designed a novel distributed spatial data management framework that integrates our proposed techniques. We studied the challenges of achieving scalable spatial data management, focusing on minimizing inter-node communication and data distribution costs, while ensuring that our single-node strategies can be parallelized to fully exploit each machine’s resources (threads, cores, etc.). Our system supports all major spatial data and query types and outperforms similar state-of-the-art frameworks in both performance, memory requirements and usability.

Spatial Reasoning using RAG In the last part of this thesis, we explored how topological relations combined with RAG can be used to improve and guide an LLM’s spatial reasoning on domain-specific knowledge at inference time. We combined modern techniques (NER, graph indexing, vector similarity search, etc.) with our spatial data management methods to generate text that captures meaningful spatial relationships between entities and to automatically provide context relevant to a user’s spatial query to the LLM, thereby guiding the model’s response with factual data in a form it can readily comprehend.

In summary, this dissertation demonstrates how spatial approximations can make spatial data management scalable, with proven potential for integration into real-world spatial DBMS to enhance performance. We showed that efficient and accurate detection of topological relationships in spatial data has broad applicability, and we presented methods for integrating such techniques into distributed environments, as well as coupling them with non-spatial indexes (graphs, vectors) for LLM-specific use cases. Together, these contributions underscore the growing importance of modern, scalable spatial data management in meeting ever-increasing demands.

6.2 Directions for Future Work

In the future, we will explore the effectiveness of approximate join evaluation according with the recent trend [25]. Additionally, we plan to investigate further the

problem of interval join order optimization and explore the effectiveness of APRIL for 3D objects (e.g., polytopes). We also aim to investigate integrating APRIL into a big distributed spatial database management system, such as Apache Sedona, as well as an open-source spatial database system, such as PostGIS and leverage GPUs for parallelizing both the construction and the evaluation of APRIL.

Regarding topological relations, we plan to integrate our method into existing link discovery frameworks such as Silk [141] to enable faster user-defined spatial link generation, contributing to the expansion of geospatially linked data available online. Additionally, implementing our approach in open-source spatial DBMSs (e.g., PostGIS [43]) could enhance the performance of topological relation queries, and, in turn the usability of systems and applications that handle complex spatial data and, in turn, improve the usability of systems and applications that handle complex spatial data.

For Hecatoncheir to be considered a full-scale spatial data management system, it must support (i) cloud deployment, (ii) fault tolerance, and (iii) security. These represent the most important directions for future work on the system. While some of these challenges could be addressed using robust, publicly available tools for resource management, allocation, and load balancing, the overarching goal of Hecatoncheir is to remain a self-contained framework with minimal installation requirements and no external dependencies. Therefore, it requires specialized, dedicated mechanisms to provide these features while remaining lightweight and independent.

We also plan to integrate a wider range of indexing techniques, particularly for data assignment to nodes. For example, QuadTrees and Adaptive Grids can help better balance non-uniform data distributions across a finite number of nodes. Furthermore, certain query processing mechanisms in Hecatoncheir such as k NN and distance join workflows, can be further parallelized and made more asynchronous between Workers and the Host, enabling faster performance and reduced downtime.

In addition, we aim to introduce more user-oriented features, including broader support for cross-type query combinations, user-defined functions (UDFs), and SQL support. These enhancements will allow users to define and execute more complex queries, moving Hecatoncheir closer to a fully featured spatial DBMS.

Finally, the ultimate goal of our work on SpaRAGraph in LLMs is to evaluate models on open-ended spatial reasoning tasks, allowing evaluators to assess model predictions against textual ground truths while examining the evaluators' own effectiveness and accuracy. We also plan to expand our datasets to include global

coverage and incorporate additional entity types such as lakes, parks and rivers. This will increase both the diversity and complexity of spatial relationships, enabling more rigorous evaluation of spatial inference across varied data types.

BIBLIOGRAPHY

- [1] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger, “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*, R. T. Snodgrass and M. Winslett, Eds. ACM Press, 1994, pp. 197–208. [Online]. Available: <https://doi.org/10.1145/191839.191880>
- [2] “Qgis,” 2024, www.qgis.org.
- [3] R. Ramakrishnan and J. Gehrke, *Database management systems (3. ed.)*. McGraw-Hill, 2003.
- [4] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company, 2020. [Online]. Available: <https://www.db-book.com/>
- [5] W. H. Inmon, “What is a data warehouse?” *Prism Tech Topic*, vol. 1, no. 1, pp. 1–5, 1995.
- [6] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *SIGMOD’84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984*, B. Yormark, Ed. ACM Press, 1984, pp. 47–57. [Online]. Available: <https://doi.org/10.1145/602259.602266>
- [7] D. Tsitsigkos, K. Lampropoulos, P. Bouros, N. Mamoulis, and M. Terrovitis, “A two-layer partitioning for non-point spatial data,” in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 1787–1798. [Online]. Available: <https://doi.org/10.1109/ICDE51399.2021.00157>
- [8] M. P. Forum, “Mpi: A message-passing interface standard,” 1994.

- [9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.
- [10] E. H. Jacox and H. Samet, "Spatial join techniques," *ACM Trans. Database Syst.*, vol. 32, no. 1, p. 7, 2007. [Online]. Available: <https://doi.org/10.1145/1206049.1206056>
- [11] G. Papadakis, G. M. Mandilaras, N. Mamoulis, and M. Koubarakis, "Progressive, holistic geospatial interlinking," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, J. Leskovec, M. Grobeldnik, M. Najork, J. Tang, and L. Zia, Eds. ACM / IW3C2, 2021, pp. 833–844. [Online]. Available: <https://doi.org/10.1145/3442381.3449850>
- [12] K. Theocharidis, J. Liagouris, N. Mamoulis, P. Bouros, and M. Terrovitis, "SRX: efficient management of spatial RDF data," *VLDB J.*, vol. 28, no. 5, pp. 703–733, 2019. [Online]. Available: <https://doi.org/10.1007/s00778-019-00554-z>
- [13] M. Shinya and M. Forgue, "Interference detection through rasterization," *Comput. Animat. Virtual Worlds*, vol. 2, no. 4, pp. 132–134, 1991. [Online]. Available: <https://doi.org/10.1002/vis.4340020408>
- [14] S. Nobari, F. Tauheed, T. Heinis, P. Karras, S. Bressan, and A. Ailamaki, "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*, K. A. Ross, D. Srivastava, and D. Papadias, Eds. ACM, 2013, pp. 701–712. [Online]. Available: <https://doi.org/10.1145/2463676.2463700>
- [15] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtrees: 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*, J. Fujii, Ed. ACM, 1996, pp. 171–180. [Online]. Available: <https://doi.org/10.1145/237170.237244>
- [16] H. Doraiswamy and J. Freire, "A gpu-friendly geometric data model and algebra for spatial queries," in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland,*

- OR, USA], June 14-19, 2020, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds. ACM, 2020, pp. 1875–1885. [Online]. Available: <https://doi.org/10.1145/3318464.3389774>
- [17] —, “SPADE: gpu-powered spatial database engine for commodity hardware,” in *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2022, pp. 2669–2681. [Online]. Available: <https://doi.org/10.1109/ICDE53745.2022.00245>
- [18] A. Kipf, H. Lang, V. Pandey, R. A. Persa, C. Anneser, E. T. Zacharatou, H. Doraiswamy, P. A. Boncz, T. Neumann, and A. Kemper, “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*, A. Bonifati, Y. Zhou, M. A. V. Salles, A. Böhm, D. Olteanu, G. H. L. Fletcher, A. Khan, and B. Yang, Eds. OpenProceedings.org, 2020, pp. 347–358. [Online]. Available: <https://doi.org/10.5441/002/edbt.2020.31>
- [19] C. Winter, A. Kipf, C. Anneser, E. T. Zacharatou, T. Neumann, and A. Kemper, “Geoblocks: A query-cache accelerated data structure for spatial aggregation over polygons,” in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthos, and F. Guerra, Eds. OpenProceedings.org, 2021, pp. 169–180. [Online]. Available: <https://doi.org/10.5441/002/edbt.2021.16>
- [20] S. Nobari, Q. Qu, and C. S. Jensen, “In-memory spatial join: The data matters!” in *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, V. Markl, S. Orlando, B. Mitschang, P. Andritsos, K. Sattler, and S. Breß, Eds. OpenProceedings.org, 2017, pp. 462–465. [Online]. Available: <https://doi.org/10.5441/002/edbt.2017.45>
- [21] V. Pandey, A. Kipf, T. Neumann, and A. Kemper, “How good are modern spatial analytics systems?” *Proc. VLDB Endow.*, vol. 11, no. 11, pp. 1661–1673, 2018. [Online]. Available: <http://www.vldb.org/pvldb/vol11/p1661-pandey.pdf>

- [22] D. Sidlauskas, S. Chester, E. T. Zacharatou, and A. Ailamaki, “Improving spatial data processing by clipping minimum bounding boxes,” in *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 2018, pp. 425–436. [Online]. Available: <https://doi.org/10.1109/ICDE.2018.00046>
- [23] D. Teng, F. Baig, Q. Sun, J. Kong, and F. Wang, “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*. IEEE, 2021, pp. 99–108. [Online]. Available: <https://doi.org/10.1109/MDM52706.2021.00024>
- [24] E. T. Zacharatou, H. Doraiswamy, A. Ailamaki, C. T. Silva, and J. Freire, “GPU rasterization for real-time spatial aggregation over arbitrary polygons,” *Proc. VLDB Endow.*, vol. 11, no. 3, pp. 352–365, 2017. [Online]. Available: <http://www.vldb.org/pvldb/vol11/p352-zacharatou.pdf>
- [25] E. T. Zacharatou, A. Kipf, I. Sabek, V. Pandey, H. Doraiswamy, and V. Markl, “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, 2021. [Online]. Available: http://cidrdb.org/cidr2021/papers/cidr2021_paper19.pdf
- [26] T. Brinkhoff, H. Kriegel, and B. Seeger, “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*, P. Buneman and S. Jajodia, Eds. ACM Press, 1993, pp. 237–246. [Online]. Available: <https://doi.org/10.1145/170035.170075>
- [27] D. Tsitsigkos, P. Bouros, N. Mamoulis, and M. Terrovitis, “Parallel in-memory evaluation of spatial joins,” in *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2019, Chicago, IL, USA, November 5-8, 2019*, F. B. Kashani, G. Trajcevski, R. H. Güting, L. Kulik, and S. D. Newsam, Eds. ACM, 2019, pp. 516–519. [Online]. Available: <https://doi.org/10.1145/3347146.3359343>

- [28] G. Zimbrao and J. M. de Souza, “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*, A. Gupta, O. Shmueli, and J. Widom, Eds. Morgan Kaufmann, 1998, pp. 558–569. [Online]. Available: <http://www.vldb.org/conf/1998/p558.pdf>
- [29] T. Georgiadis and N. Mamoulis, “Raster intervals: An approximation technique for polygon intersection joins,” *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 36:1–36:18, 2023. [Online]. Available: <https://doi.org/10.1145/3588716>
- [30] M. N. DeMers, *Fundamentals of geographic information systems (4. ed.)*. Wiley, 2008.
- [31] I. H. Kim, C. Feng, and Y. Wang, “A simplified linear feature matching method using decision tree analysis, weighted linear directional mean, and topological relationships,” *Int. J. Geogr. Inf. Sci.*, vol. 31, no. 5, pp. 1042–1060, 2017. [Online]. Available: <https://doi.org/10.1080/13658816.2016.1267736>
- [32] European Environment Agency, “Homepage of the european environment agency,” 2024. [Online]. Available: <https://www.eea.europa.eu/>
- [33] N. Mamoulis, *Spatial data management*. Morgan & Claypool Publishers, 2012, vol. 21.
- [34] X. Lin, Q. Liu, Y. Yuan, and X. Zhou, “Multiscale histograms: Summarizing topological relations in large spatial datasets,” in *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003*, J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, Eds. Morgan Kaufmann, 2003, pp. 814–825. [Online]. Available: <http://www.vldb.org/conf/2003/papers/S24P03.pdf>
- [35] M. A. Sherif, K. Drefßler, P. Smeros, and A. N. Ngomo, “Radon - rapid discovery of topological relations,” in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, S. Singh and S. Markovitch, Eds. AAAI Press, 2017, pp. 175–181. [Online]. Available: <https://doi.org/10.1609/aaai.v31i1.10478>

- [36] Y. Singh, C. M. Farrelly, Q. A. Hathaway, T. Leiner, J. Jagtap, G. E. Carlsson, and B. J. Erickson, "Topological data analysis in medical imaging: current state of the art," *Insights into Imaging*, vol. 14, no. 1, p. 58, 2023.
- [37] A. P. Sistla, C. T. Yu, and R. Haddad, "Reasoning about spatial relationships in picture retrieval systems," in *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Morgan Kaufmann, 1994, pp. 570–581. [Online]. Available: <http://www.vldb.org/conf/1994/P570.PDF>
- [38] D. Papadias, N. Mamoulis, and D. Meretakis, "Image similarity retrieval by spatial constraints," in *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management, Bethesda, Maryland, USA, November 3-7, 1998*, G. Gardarin, J. C. French, N. Pissinou, K. Makki, and L. Bouganim, Eds. ACM, 1998, pp. 289–296. [Online]. Available: <https://doi.org/10.1145/288627.288669>
- [39] L. Guo, J. Shanmugasundaram, and G. Yona, "Topology search over biological databases," in *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, R. Chirkova, A. Dogac, M. T. Özsu, and T. K. Sellis, Eds. IEEE Computer Society, 2007, pp. 556–565. [Online]. Available: <https://doi.org/10.1109/ICDE.2007.367901>
- [40] E. Clementini, P. D. Felice, and P. van Oosterom, "A small set of formal topological relationships suitable for end-user interaction," in *Advances in Spatial Databases, Third International Symposium, SSD'93, Singapore, June 23-25, 1993, Proceedings*, ser. Lecture Notes in Computer Science, D. J. Abel and B. C. Ooi, Eds., vol. 692. Springer, 1993, pp. 277–295. [Online]. Available: https://doi.org/10.1007/3-540-56869-7_16
- [41] C. Strobl, "Dimensionally extended nine-intersection model (DE-9IM)," in *Encyclopedia of GIS*, S. Shekhar, H. Xiong, and X. Zhou, Eds. Springer, 2017, pp. 470–476. [Online]. Available: https://doi.org/10.1007/978-3-319-17885-1_298
- [42] ESRI, "Arcgis," 2024, www.esri.com/en-us/arcgis/products/arcgis-pro.

- [43] PostGIS Development Team, *PostGIS*, PostGIS Development Group, 2023. [Online]. Available: <https://postgis.net/>
- [44] Oracle, “Oracle spatial database,” 2024, www.oracle.com/database/spatial/.
- [45] E. F. LocationTech group, “Jts topology suite,” 2024, www.github.com/locationtech/jts.
- [46] GEOS contributors, *GEOS Coordinate Transformation Software Library*, Open Source Geospatial Foundation, 2021. [Online]. Available: <https://libgeos.org/>
- [47] Boost, “Boost C++ Libraries,” <http://www.boost.org/>, 2025.
- [48] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars, *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. [Online]. Available: <https://doi.org/10.1007/978-3-540-77974-2>
- [49] J. M. Patel and D. J. DeWitt, “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 I. S. Mumick, Eds. ACM Press, 1996, pp. 259–270. [Online]. Available: <https://doi.org/10.1145/233269.233338>
- [50] T. Georgiadis, E. T. Zacharitou, and N. Mamoulis, “Raster interval object approximations for spatial intersection joins,” *VLDB J.*, vol. 34, no. 1, p. 8, 2025. [Online]. Available: <https://doi.org/10.1007/s00778-024-00887-4>
- [51] A. Eldawy and M. F. Mokbel, “Spatialhadoop: A mapreduce framework for spatial data,” in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, J. Gehrke, W. Lehner, K. Shim, S. K. Cha, and G. M. Lohman, Eds. IEEE Computer Society, 2015, pp. 1352–1363. [Online]. Available: <https://doi.org/10.1109/ICDE.2015.7113382>
- [52] Apache, “Apache sedona,” <https://sedona.apache.org/>.
- [53] A. Eldawy and M. F. Mokbel, “A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data,” *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1230–1233, 2013.

- [54] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo, “Simba: Efficient in-memory spatial analytics,” in *SIGMOD*, 2016.
- [55] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref, “Locationspark: A distributed in-memory data management system for big spatial data,” *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1565–1568, 2016.
- [56] Boost.Geometry Development Team, *Boost.Geometry*, 2023. [Online]. Available: <https://www.boost.org/doc/libs/release/libs/geometry/>
- [57] D. Tsitsigkos, P. Bouros, K. Lampropoulos, N. Mamoulis, and M. Terrovitis, “Two-layer space-oriented partitioning for non-point data,” *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 3, pp. 1344–1355, 2024. [Online]. Available: <https://doi.org/10.1109/TKDE.2023.3297975>
- [58] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>
- [59] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T. Chua, and Q. Li, “A survey on RAG meeting llms: Towards retrieval-augmented large language models,” in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds. ACM, 2024, pp. 6491–6501. [Online]. Available: <https://doi.org/10.1145/3637528.3671470>
- [60] R. Manvi, S. Khanna, G. Mai, M. Burke, D. B. Lobell, and S. Ermon, “Geollm: Extracting geospatial knowledge from large language models,” in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: <https://openreview.net/forum?id=TqL2xBwXP3>

- [61] S. Singh, M. Fore, and D. Stamoulis, “Geollm-engine: A realistic environment for building geospatial copilots,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024 - Workshops, Seattle, WA, USA, June 17-18, 2024*. IEEE, 2024, pp. 585–594. [Online]. Available: <https://doi.org/10.1109/CVPRW63382.2024.00063>
- [62] Y. Zhang, C. Wei, Z. He, and W. Yu, “Geogpt: An assistant for understanding and processing geospatial tasks,” *Int. J. Appl. Earth Obs. Geoinformation*, vol. 131, p. 103976, 2024. [Online]. Available: <https://doi.org/10.1016/j.jag.2024.103976>
- [63] V. Leis, A. Kemper, and T. Neumann, “The adaptive radix tree: Artful indexing for main-memory databases,” in *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, C. S. Jensen, C. M. Jermaine, and X. Zhou, Eds. IEEE Computer Society, 2013, pp. 38–49. [Online]. Available: <https://doi.org/10.1109/ICDE.2013.6544812>
- [64] F. P. Preparata and M. I. Shamos, *Computational Geometry - An Introduction*, ser. Texts and Monographs in Computer Science. Springer, 1985. [Online]. Available: <https://doi.org/10.1007/978-1-4612-1098-6>
- [65] J. Yu, Z. Zhang, and M. Sarwat, “Spatial data management in apache spark: the geospark perspective and beyond,” *GeoInformatica*, vol. 23, no. 1, pp. 37–78, 2019. [Online]. Available: <https://doi.org/10.1007/s10707-018-0330-9>
- [66] R. Zhang, H. Li, and P. Yue, “A spatial-aware representation learning model for link completion in geokg: A case study on wikidata and openstreetmap,” in *11th International Conference on Agro-Geoinformatics, Agro-Geoinformatics 2023, Wuhan, China, July 25-28, 2023*. IEEE, 2023, pp. 1–6. [Online]. Available: <https://doi.org/10.1109/Agro-Geoinformatics59224.2023.10233488>
- [67] G. Mann, A. Dsouza, R. Yu, and E. Demidova, “Spatial link prediction with spatial and semantic embeddings,” in *The Semantic Web - ISWC 2023 - 22nd International Semantic Web Conference, Athens, Greece, November 6-10, 2023, Proceedings, Part I*, ser. Lecture Notes in Computer Science, T. R. Payne, V. Presutti, G. Qi, M. Poveda-Villalón, G. Stoilos, L. Hollink, Z. Kaoudi, G. Cheng, and J. Li, Eds., vol. 14265. Springer, 2023, pp. 179–196. [Online]. Available: https://doi.org/10.1007/978-3-031-47240-4_10

- [68] B. Liu, M. Peng, W. Xu, X. Jia, and M. Peng, “Unilp: Unified topology-aware generative framework for link prediction in knowledge graph,” in *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, T. Chua, C. Ngo, R. Kumar, H. W. Lauw, and R. K. Lee, Eds. ACM, 2024, pp. 2170–2180. [Online]. Available: <https://doi.org/10.1145/3589334.3645592>
- [69] Z. Shang, P. Wang, Y. Liu, J. Liu, and W. Ke, “ASKRL: an aligned-spatial knowledge representation learning framework for open-world knowledge graph,” in *The Semantic Web - ISWC 2023 - 22nd International Semantic Web Conference, Athens, Greece, November 6-10, 2023, Proceedings, Part I*, ser. Lecture Notes in Computer Science, T. R. Payne, V. Presutti, G. Qi, M. Poveda-Villalón, G. Stoilos, L. Hollink, Z. Kaoudi, G. Cheng, and J. Li, Eds., vol. 14265. Springer, 2023, pp. 101–120. [Online]. Available: https://doi.org/10.1007/978-3-031-47240-4_6
- [70] S. Ray, B. Simion, A. D. Brown, and R. Johnson, “Skew-resistant parallel in-memory spatial join,” in *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014*, C. S. Jensen, H. Lu, T. B. Pedersen, C. Thomsen, and K. Torp, Eds. ACM, 2014, pp. 6:1–6:12. [Online]. Available: <https://doi.org/10.1145/2618243.2618262>
- [71] N. Mamoulis and D. Papadias, “Slot index spatial join,” *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 1, pp. 211–231, 2003. [Online]. Available: <https://doi.org/10.1109/TKDE.2003.1161591>
- [72] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, “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*, H. Garcia-Molina and H. V. Jagadish, Eds. ACM Press, 1990, pp. 322–331. [Online]. Available: <https://doi.org/10.1145/93597.98741>
- [73] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter, “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*, A. Gupta, O. Shmueli, and J. Widom, Eds. Morgan Kaufmann, 1998, pp. 570–581. [Online]. Available: <http://www.vldb.org/conf/1998/p570.pdf>

- [74] Y. Fang, M. T. Friedman, G. Nair, M. Rys, and A. Schmid, "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*, J. T. Wang, Ed. ACM, 2008, pp. 1207–1216. [Online]. Available: <https://doi.org/10.1145/1376616.1376737>
- [75] M. I. Shamos and D. Hoey, "Geometric intersection problems," in *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*. IEEE Computer Society, 1976, pp. 208–215. [Online]. Available: <https://doi.org/10.1109/SFCS.1976.16>
- [76] D. Aghajarian, S. Puri, and S. K. Prasad, "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*, S. Ravada, M. E. Ali, S. D. Newsam, M. Renz, and G. Trajcevski, Eds. ACM, 2016, pp. 18:1–18:10. [Online]. Available: <https://doi.org/10.1145/2996913.2996982>
- [77] Y. Liu, J. Yang, and S. Puri, "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, 2019, pp. 141–151. [Online]. Available: <https://doi.org/10.1109/HiPC.2019.00027>
- [78] W. M. Badawy and W. G. Aref, "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*, C. B. Medeiros, Ed. ACM, 1999, pp. 97–102. [Online]. Available: <https://doi.org/10.1145/320134.320160>
- [79] Y. Liu and S. Puri, "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*, C. Lu, F. Wang, G. Trajcevski, Y. Huang, S. D. Newsam, and L. Xiong, Eds. ACM, 2020, pp. 487–496. [Online]. Available: <https://doi.org/10.1145/3397536.3422264>

- [80] J. A. Orenstein, "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*, J. Clifford, B. G. Lindsay, and D. Maier, Eds. ACM Press, 1989, pp. 295–305. [Online]. Available: <https://doi.org/10.1145/67544.66954>
- [81] S. Singla and A. Eldawy, "Raptor zonal statistics: Fully distributed zonal statistics of big raster + vector data," in *2020 IEEE International Conference on Big Data (IEEE BigData 2020)*, Atlanta, GA, USA, December 10-13, 2020, X. Wu, C. Jermaine, L. Xiong, X. Hu, O. Kotevska, S. Lu, W. Xu, S. Aluru, C. Zhai, E. Al-Masri, Z. Chen, and J. Saltz, Eds. IEEE, 2020, pp. 571–580. [Online]. Available: <https://doi.org/10.1109/BigData50022.2020.9377907>
- [82] S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay, and E. Scudiero, "Experimental study of big raster and vector database systems," in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 2243–2248. [Online]. Available: <https://doi.org/10.1109/ICDE51399.2021.00231>
- [83] —, "The raptor join operator for processing big raster + vector data," in *SIGSPATIAL '21: 29th International Conference on Advances in Geographic Information Systems, Virtual Event / Beijing, China, November 2-5, 2021*, X. Meng, F. Wang, C. Lu, Y. Huang, S. Shekhar, and X. Xie, Eds. ACM, 2021, pp. 324–335. [Online]. Available: <https://doi.org/10.1145/3474717.3483971>
- [84] D. Aghajarian and S. K. Prasad, "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*, E. G. Hoel, S. D. Newsam, S. Ravada, R. Tamassia, and G. Trajcevski, Eds. ACM, 2017, pp. 56:1–56:4. [Online]. Available: <https://doi.org/10.1145/3139958.3140056>
- [85] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: a unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016. [Online]. Available: <https://doi.org/10.1145/2934664>

- [86] Apache Software Foundation, “Hadoop,” 2024. [Online]. Available: <https://hadoop.apache.org>
- [87] S. T. Leutenegger, J. Edgington, and M. A. López, “STR: A simple and efficient algorithm for r-tree packing,” in *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997, Birmingham, UK*, W. A. Gray and P. Larson, Eds. IEEE Computer Society, 1997, pp. 497–506. [Online]. Available: <https://doi.org/10.1109/ICDE.1997.582015>
- [88] S. You, J. Zhang, and L. Gruenwald, “Large-scale spatial join query processing in cloud,” in *CloudDB, ICDE Workshops*, 2015, pp. 34–41.
- [89] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. H. Saltz, “Hadoop-gis: A high performance spatial data warehousing system over mapreduce,” *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1009–1020, 2013. [Online]. Available: <http://www.vldb.org/pvldb/vol6/p1009-aji.pdf>
- [90] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, “Open MPI: goals, concept, and design of a next generation MPI implementation,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users’ Group Meeting, Budapest, Hungary, September 19-22, 2004, Proceedings*, ser. Lecture Notes in Computer Science, D. Kranzlmüller, P. Kacsuk, and J. J. Dongarra, Eds., vol. 3241. Springer, 2004, pp. 97–104. [Online]. Available: https://doi.org/10.1007/978-3-540-30218-6_19
- [91] J. Goll, S. V. Rajagopala, S. C. Shiau, H. Wu, B. T. Lamb, and P. Uetz, “MPIDB: the microbial protein interaction database,” *Bioinform.*, vol. 24, no. 15, pp. 1743–1744, 2008. [Online]. Available: <https://doi.org/10.1093/bioinformatics/btn285>
- [92] M. A. Haddad and J. Robinson, “Using a network of workstations to enhance database query processing performance,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 8th European PVM/MPI Users’ Group Meeting, Santorini/Thera, Greece, September 23-26, 2001, Proceedings*, ser. Lecture Notes in Computer Science, Y. Cotronis and J. J.

Dongarra, Eds., vol. 2131. Springer, 2001, pp. 352–359. [Online]. Available: https://doi.org/10.1007/3-540-45417-9_48

- [93] F. Liu, C. Barthels, S. Blanas, H. Kimura, and G. Swart, “Beyond MPI: new communication interfaces for database systems and data-intensive applications,” *SIGMOD Rec.*, vol. 49, no. 4, pp. 12–17, 2020. [Online]. Available: <https://doi.org/10.1145/3456859.3456862>
- [94] C. Sun, D. Agrawal, and A. E. Abbadi, “Hardware acceleration for spatial selections and joins,” in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, A. Y. Halevy, Z. G. Ives, and A. Doan, Eds. ACM, 2003, pp. 455–466. [Online]. Available: <https://doi.org/10.1145/872757.872813>
- [95] J. Chen, H. Lin, X. Han, and L. Sun, “Benchmarking large language models in retrieval-augmented generation,” in *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, M. J. Wooldridge, J. G. Dy, and S. Natarajan, Eds. AAAI Press, 2024, pp. 17754–17762. [Online]. Available: <https://doi.org/10.1609/aaai.v38i16.29728>
- [96] R. Friel, M. Belyi, and A. Sanyal, “Ragbench: Explainable benchmark for retrieval-augmented generation systems,” *CoRR*, vol. abs/2407.11005, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2407.11005>
- [97] X. Yang, K. Sun, H. Xin, Y. Sun, N. Bhalla, X. Chen, S. Choudhary, R. D. Gui, Z. W. Jiang, Z. Jiang, L. Kong, B. Moran, J. Wang, Y. Xu, A. Yan, C. Yang, E. Yuan, H. Zha, N. Tang, L. Chen, N. Scheffer, Y. Liu, N. Shah, R. Wanga, A. Kumar, S. Yih, and X. Dong, “CRAG - comprehensive RAG benchmark,” in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/1435d2d0fca85a84d83ddcb754f58c29-Abstract-Datasets_and_Benchmarks_Track.html

- [98] D. Rau, H. Déjean, N. Chirkova, T. Formal, S. Wang, S. Clinchant, and V. Nikoulina, “BERGEN: A benchmarking library for retrieval-augmented generation,” in *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Y. Al-Onaizan, M. Bansal, and Y. Chen, Eds. Association for Computational Linguistics, 2024, pp. 7640–7663. [Online]. Available: <https://doi.org/10.18653/v1/2024.findings-emnlp.449>
- [99] G. Xiong, Q. Jin, Z. Lu, and A. Zhang, “Benchmarking retrieval-augmented generation for medicine,” in *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024, pp. 6233–6251. [Online]. Available: <https://doi.org/10.18653/v1/2024.findings-acl.372>
- [100] N. Pipitone and G. H. Alami, “Legalbench-rag: A benchmark for retrieval-augmented generation in the legal domain,” *CoRR*, vol. abs/2408.10343, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2408.10343>
- [101] Y. Hui, Y. Lu, and H. Zhang, “UDA: A benchmark suite for retrieval augmented generation in real-world document analysis,” in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/7c06759d1a8567f087b02e8589454917-Abstract-Datasets_and_Benchmarks_Track.html
- [102] Y. Tang and Y. Yang, “Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries,” *CoRR*, vol. abs/2401.15391, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2401.15391>
- [103] H. Han, Y. Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang, Q. He, Z. Hua, B. Long, T. Zhao, N. Shah, A. Javari, Y. Xia, and J. Tang, “Retrieval-augmented generation with graphs (graphrag),” *CoRR*, vol. abs/2501.00309, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2501.00309>

- [104] X. He, Y. Tian, Y. Sun, N. V. Chawla, T. Laurent, Y. LeCun, X. Bresson, and B. Hooi, “G-retriever: Retrieval-augmented generation for textual graph understanding and question answering,” in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/efaf1c9726648c8ba363a5c927440529-Abstract-Conference.html
- [105] C. Mavromatis and G. Karypis, “GNN-RAG: graph neural retrieval for efficient large language model reasoning on knowledge graphs,” in *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Association for Computational Linguistics, 2025, pp. 16 682–16 699. [Online]. Available: <https://aclanthology.org/2025.findings-acl.856/>
- [106] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html
- [107] D. Yu, R. Bao, G. Mai, and L. Zhao, “Spatial-rag: Spatial retrieval augmented generation for real-world spatial reasoning questions,” *CoRR*, vol. abs/2502.18470, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2502.18470>
- [108] F. Li, D. C. Hogg, and A. G. Cohn, “Advancing spatial reasoning in large language models: An in-depth evaluation and enhancement using the stepgame benchmark,” in *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada,*

- M. J. Wooldridge, J. G. Dy, and S. Natarajan, Eds. AAAI Press, 2024, pp. 18 500–18 507. [Online]. Available: <https://doi.org/10.1609/aaai.v38i17.29811>
- [109] Z. Shi, Q. Zhang, and A. Lipani, “Stepgame: A new benchmark for robust multi-hop spatial reasoning in texts,” in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 2022, pp. 11 321–11 329. [Online]. Available: <https://doi.org/10.1609/aaai.v36i10.21383>
- [110] A. G. Cohn and J. Hernández-Orallo, “Dialectical language model evaluation: An initial appraisal of the commonsense spatial reasoning abilities of llms,” *CoRR*, vol. abs/2304.11164, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2304.11164>
- [111] J. Qi, Z. Li, and E. Tanin, “Maasdb: Spatial databases in the era of large language models (vision paper),” in *Proceedings of the 31st ACM International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2023, Hamburg, Germany, November 13-16, 2023*, M. Renz and M. A. Nascimento, Eds. ACM, 2023, pp. 54:1–54:4. [Online]. Available: <https://doi.org/10.1145/3589132.3625597>
- [112] Z. Li and H. Ning, “Autonomous GIS: the next-generation ai-powered GIS,” *Int. J. Digit. Earth*, vol. 16, no. 2, pp. 4668–4686, 2023. [Online]. Available: <https://doi.org/10.1080/17538947.2023.2278895>
- [113] X. Zhang, F. Xiao, L. Yan, and Z. Zhang, “GS-SQL: modeling spatial semantics in spatial text-to-sql,” in *International Joint Conference on Neural Networks, IJCNN 2024, Yokohama, Japan, June 30 - July 5, 2024*. IEEE, 2024, pp. 1–7. [Online]. Available: <https://doi.org/10.1109/IJCNN60899.2024.10651125>
- [114] OpenStreetMap contributors, “Planet dump retrieved from <https://planet.osm.org>,” <https://www.openstreetmap.org>, 2017.
- [115] D. Hilbert, “Über die stetige abbildung einer linie auf ein flächenstück,” in *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes: Nebst Einer Lebensgeschichte*. Springer, 1935, pp. 1–2.

- [116] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company, 1966.
- [117] D. R. Cutting and J. O. Pedersen, "Optimizations for dynamic inverted index maintenance," in *SIGIR'90, 13th International Conference on Research and Development in Information Retrieval, Brussels, Belgium, 5-7 September 1990, Proceedings*, J. Vidick, Ed. ACM, 1990, pp. 405–411. [Online]. Available: <https://doi.org/10.1145/96749.98245>
- [118] L. H. Thiel and H. S. Heaps, "Program design for retrospective searches on large data bases," *Inf. Storage Retr.*, vol. 8, no. 1, pp. 1–20, 1972. [Online]. Available: [https://doi.org/10.1016/0020-0271\(72\)90024-1](https://doi.org/10.1016/0020-0271(72)90024-1)
- [119] D. Lemire and L. Boytsov, "Decoding billions of integers per second through vectorization," *Softw. Pract. Exp.*, vol. 45, no. 1, pp. 1–29, 2015. [Online]. Available: <https://doi.org/10.1002/spe.2203>
- [120] cruppstahl, *libvbyte - Fast C Library for 32bit and 64bit Integer Compression*, 2017. [Online]. Available: <https://github.com/cruppstahl/libvbyte>
- [121] J. Dittrich and B. Seeger, "Data redundancy and duplicate detection in spatial join processing," in *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, D. B. Lomet and G. Weikum, Eds. IEEE Computer Society, 2000, pp. 535–546. [Online]. Available: <https://doi.org/10.1109/ICDE.2000.839452>
- [122] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *8th European Computer Graphics Conference and Exhibition, Eurographics 1987, Amsterdam, The Netherlands, August 24-28, 1987, Proceedings*, G. Maréchal, Ed. North-Holland / Eurographics Association, 1987. [Online]. Available: <https://doi.org/10.2312/egtp.19871000>
- [123] A. R. Smith, "Tint fill," in *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1979, Chicago, Illinois, USA, August 8-10, 1979*, T. A. DeFanti, B. H. McCormick, B. W. Pollack, N. I. Badler, and S. H. Chasen, Eds. ACM, 1979, pp. 276–283. [Online]. Available: <https://doi.org/10.1145/800249.807456>

- [124] J. Bresenham, “Algorithm for computer control of a digital plotter,” *IBM Syst. J.*, vol. 4, no. 1, pp. 25–30, 1965. [Online]. Available: <https://doi.org/10.1147/sj.41.0025>
- [125] K. Museth, “Hierarchical digital differential analyzer for efficient ray-marching in opendb,” in *Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH ’14, Vancouver, Canada, August 10-14, 2014, Talks Proceedings*. ACM, 2014, p. 40:1. [Online]. Available: <https://doi.org/10.1145/2614106.2614136>
- [126] tengdj, “Farm: A repository on github,” [urlhttps://github.com/tengdj/FARM](https://github.com/tengdj/FARM), 2025, gitHub repository.
- [127] T. Georgiadis and N. Mamoulis, “Scalable spatial topology joins,” in *Proceedings 29th International Conference on Extending Database Technology, EDBT 2026, Tampere, Finland, March 24-27, 2026*, W. Lehner, V. Braganholo, K. Stefanidis, Z. Zhang, A. Krause, and J. F. N. Pimentel, Eds. OpenProceedings.org, 2026, pp. 110–116. [Online]. Available: <https://doi.org/10.48786/edbt.2026.10>
- [128] N. Koutroumanis, C. Doukeridis, and A. Vlachou, “Parallel spatial join processing with adaptive replication,” in *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, A. Simitsis, B. Kemme, A. Queralt, O. Romero, and P. Jovanovic, Eds. OpenProceedings.org, 2025, pp. 464–476. [Online]. Available: <https://doi.org/10.48786/edbt.2025.37>
- [129] A. Michalopoulos, D. Tsitsigkos, P. Bouros, N. Mamoulis, and M. Terrovitis, “Efficient distance queries on non-point data,” *ACM Trans. Spatial Algorithms Syst.*, vol. 11, no. 1, pp. 1:1–1:37, 2025. [Online]. Available: <https://doi.org/10.1145/3698194>
- [130] S. Shende and A. D. Malony, “The tau parallel performance system,” *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 287–311, 2006. [Online]. Available: <https://doi.org/10.1177/1094342006064482>
- [131] M. de Berg, *Computational geometry: algorithms and applications*. Springer, 1997. [Online]. Available: <https://www.worldcat.org/oclc/36800677>

- [132] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, “spaCy: Industrial-strength Natural Language Processing in Python,” 2020.
- [133] E. Clementini, J. Sharma, and M. J. Egenhofer, “Modelling topological spatial relations: Strategies for query processing,” *Comput. Graph.*, vol. 18, no. 6, pp. 815–822, 1994. [Online]. Available: [https://doi.org/10.1016/0097-8493\(94\)90007-8](https://doi.org/10.1016/0097-8493(94)90007-8)
- [134] D. A. Randell, Z. Cui, and A. G. Cohn, “A spatial logic based on regions and connection,” in *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR’92). Cambridge, MA, USA, October 25-29, 1992*, B. Nebel, C. Rich, and W. R. Swartout, Eds. Morgan Kaufmann, 1992, pp. 165–176.
- [135] S. Skiadopoulos and M. Koubarakis, “Composing cardinal direction relations,” *Artif. Intell.*, vol. 152, no. 2, pp. 143–171, 2004. [Online]. Available: [https://doi.org/10.1016/S0004-3702\(03\)00137-1](https://doi.org/10.1016/S0004-3702(03)00137-1)
- [136] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with gpus,” *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021. [Online]. Available: <https://doi.org/10.1109/TBDATA.2019.2921572>
- [137] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 3980–3990. [Online]. Available: <https://doi.org/10.18653/v1/D19-1410>
- [138] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Rozière, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino,

- D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. M. Kloumann, I. Misra, I. Evtimov, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, and et al., “The llama 3 herd of models,” *CoRR*, vol. abs/2407.21783, 2024.
- [139] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de Las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mistral 7b,” *CoRR*, vol. abs/2310.06825, 2023.
- [140] Q. Team, “Qwen2.5: A party of foundation models,” September 2024. [Online]. Available: <https://qwenlm.github.io/blog/qwen2.5/>
- [141] “Silk,” 2024, <http://www.silkframework.org/>.

AUTHOR'S PUBLICATIONS

Main Publications

- [Thanasis Georgiadis](#), John Pavlopoulos, Nikos Mamoulis, **SpaRAGraph: Spatial Reasoning using Retrieval-Augmented Generation**, submitted.
- [Thanasis Georgiadis](#), Achilleas Michalopoulos, Dimitris Dimitropoulos, Dimitris Tsitsigkos, Nikos Mamoulis, **Hecatoncheir: Scaling Up and Out Spatial Data Management**, 33rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL), Minneapolis, MN, USA, November, 2025 (demo paper).
- [Thanasis Georgiadis](#), Nikos Mamoulis, **Scalable Spatial Topology joins**, in *Proceedings 29th International Conference on Extending Database Technology (EDBT)*, Tampere, Finland, March, 2026 (short paper).
- [Thanasis Georgiadis](#), Eleni Tzirita Zacharatou, Nikos Mamoulis, **Raster Interval Object Approximations for Spatial Intersection Joins**, in *The Very Large Database Journal (VLDBJ)*, 34, 1 (2025), 8.
- [Thanasis Georgiadis](#), Nikos Mamoulis, **Raster Intervals: An Approximation Technique for Polygon Intersection Joins**, in *Proceedings of the 2023 International Conference on Management of Data (ACM SIGMOD)*, Seattle, WA, USA, June, 2023.

Other Publications

SHORT BIOGRAPHY

Thanasis Georgiadis was born in 1995. He received his integrated MSc in Computer Science & Engineering from the Department of Computer Science & Engineering in the University of Ioannina in 2021. His diploma thesis "Development of a Big Spatial Data Management System" was supervised by Prof. Nikos Mamoulis. During his PhD studies, he worked as a research intern at the Huawei Technologies R&D (UK) lab in Edinburgh, Scotland from September 2023 to February 2024. From June 2024 to June 2025, he interned as a doctoral researcher at the Archimedes Unit of Athena Research Center in Athens. His research interests include data management systems, databases, query optimization, parallel programming and distributed systems.