UNIVERSITY OF CALIFORNIA
RIVERSIDE

Scaling Spatial Overlay Operations and Flock Pattern Discovery

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Andres Oswaldo Calderon Romero

November 2024

Dissertation Committee:

Dr. Vassilis Tsotras, Chairperson
Dr. Amr Magdy
Dr. Petko Bakalov
Dr. Ahmed Eldawy
Dr. Vagelis Hristidis

Chapter 1

Scaling DCEL Overlay Operations to Support Dangle and Cut Edges

1.1 Introduction

This chapter extends the previous work in [2]. The main new contributions are summarized as follows. First, we introduce a new spatial partitioner, based on the kd-tree partitioning strategy, for constructing overlay DCELs (section ??). Since it better utilizes the data distributions in optimizing DCEL partitions, it leads to noticeably improved performance. The new partitioning strategy contrasts with the original strategy that employed space-partitioning techniques based on quadtrees.

In many applications, there is a need to support overlay DCEL operations when the input contains complete polygon data (as is the case in the previous chapter) with polygons that may contain two new types of edges, namely, “dangle” and “cut” edges.

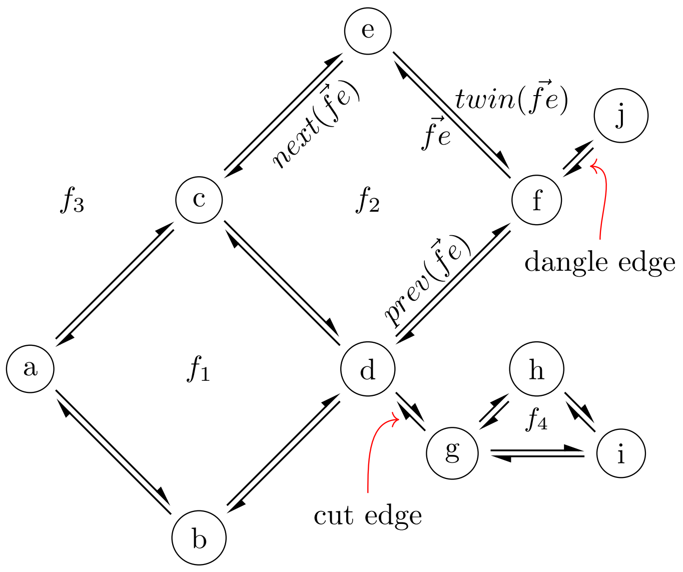

Figure 1.1: Components of the DCEL structure with dangle and cut edges.

Here, we extend the overlay DCEL approach to accept scattered and noisy line segments as input, rather than being restricted to clean polygon data. This enhancement builds on the scalable polygonization methods presented in [1], enabling the overlay of real-world datasets composed of vast sets of line segments —datasets that existing techniques are unable to process effectively. The proposed solution is an extension of the approach we used in Chapter **??**, for handling polygons with holes.

Figure 1.1 illustrates the fundamental components of a DCEL including the two new types of special half-edges. *Dangles* are half-edges with one or both endpoints not incident on another half-edge endpoint; both half-edge $\overrightarrow{fj}$ and its twin are considered dangle edges. *Cut-edges* are half-edges connected at both ends that do not form part of any polygon. The half-edge $\overrightarrow{dg}$ and its twin are classified as cut-edges.

The remainder of this chapter is organized as follows. Section 1.2 details the implementation of the new kd-tree partitioner and describes the polygon extraction process for adapting line segment inputs, which extends the overlay method to support dangle and cut edges. In Section 1.3, we present additional experiments to quantify the benefits of the kd-tree-based strategy and assess the performance of the proposed polygonization on datasets with large volumes of line segments.

1.2 Scalable Partitioning with Dangle and Cut Edges

Kd-tree Partition Strategy

In Section **??**, we use the quadtree spatial index as the baseline for our partitioning strategy. The quadtree follows a space-oriented approach, as it does not consider the content of each cell when determining potential splits. In contrast, kd-tree-based partitioning employs a data-oriented approach by sorting and selecting the midpoint within a cell to guide the placement of splits for future child nodes.

Building and populating the kd-tree partitioning follows a process similar to that of the quadtree. First, a kd-tree is constructed from a sample representing 1% of the input data to define the tree's structure, where the leaves represent the partition's cells. The input data is then fed into this kd-tree structure, with each edge assigned to the leaf cell containing its boundaries. After partitioning, the local DCELs for each layer are constructed, and the overlay operation is performed within each cell as described in Section **??**.

Section 1.3 will compare two partitioning strategies, the one presented in **??** based on the quadtree (i.e. space-oriented) and one on the kd-tree (i.e. data-oriented) indexes. Note that both tree-based data partitioning involves shuffling all edges; this however, happens only once. Our experimental evaluation (see Section 1.3.1) shows that the data-oriented approach leads to better performance.

1.2.1 Overlaying Polygons with Dangle and Cut Edges

Beyond scalability challenges, many modern applications receive spatial polygon datasets as scattered line segments—for example, road segments that form city blocks. Such datasets can be extremely large and are common in fields like urban planning, geo-targeted advertising, economic and demographic studies, and more. However, existing polygon overlay techniques are not equipped to process them directly at scale. In this section, we extend the overlay method presented in Section **??** to support polygonal input by integrating a scalable, distributed polygon extraction approach. This enhancement enables the merging of polygons with dangle and cut edges.

We built on in the scalable polygonization procedure presented in [1]. The result of that polygonization procedure generates two outputs: first, a set of closed polygons formed by the input planar line segments, and second, any edges that are not a part of any polygon (i.e., dangle or cut edges). Overlaying the polygons generated with any polygon layer follows the approaches discussed in sections **??** and **??**. However, we need to modify the algorithms provided in these previous sections to overlay an input polygon layer A with the dangle and cut edges (layer B). In particular, we modify the reduce phase.

We build upon the scalable polygonization procedure presented in [1] (see Figure 1.2), which produces two outputs: (1) a set of closed polygons formed from the input planar line segments, and (2) any edges that are not part of any polygon (i.e., dangle or cut edges). Overlaying these generated polygons with any polygon layer follows the methods discussed in Sections **??** and **??**. However, to overlay an input polygon layer A with the dangle and cut edges (layer B), we modify the algorithms from these sections, particularly adjusting the reduce phase.

Figure 1.3(a) illustrates the spatial partitioning of two input layers, A and B. Layer A contains two input polygons, A_0 and A_1, while Layer B includes of three dangle edges, B_0, B_1, and B_2.

Each edge in layer B is assigned a unique label and provided as input to the overlay module. The local overlay processs indentifies intersections between the input polygon layer A and layer B within each data partition. If a polygon with $id = i$ from layer A intersects with edges labeled $id = a$, $id = b$ and $id = c$ from layer B in a given partition, a composite label $A_iB_aB_bB_c$ is generated to represent these intersections.

During the reduce phase, we re-partition the data based on the first label, consolidating all edges that intersect with it. For instance, if two data partitions generate the labels $A_iB_aB_bB_c$ and $A_iB_xB_y$, we reassign the data so that A_i is grouped within a single partition along with all intersecting edges, specifically B_a, B_b, B_c, B_x, B_y. In Figure 1.3(b), polygon A_0 is re-partitioned with the edges it intersects, namely B_0, B_1, and B_2.

After re-partitioning, all intersecting edges from both layers are consolidated within the same partition. The next step is to identify the polygons formed by these intersections. Since there is no guarantee that only one polygon will be generated, we replace the polygon concatenation method proposed in Section **??** with a *polygonization* procedure within each partition. This polygonization process ensures that all possible new polygons are generated.

The polygonization procedure follows the algorithm outlined in [1]. It begins by

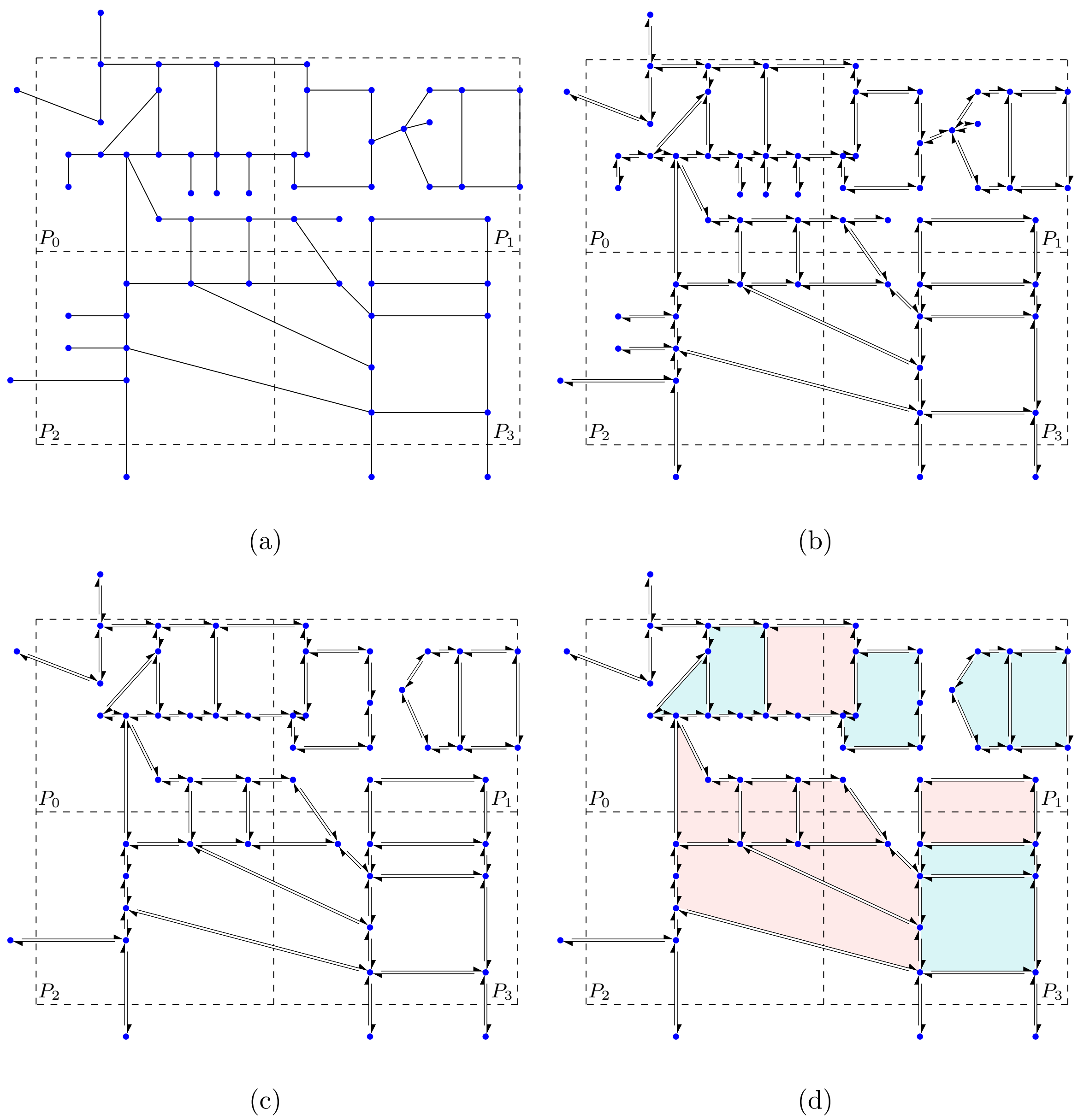

Figure 1.2: An example of four leaf nodes in a quadtree constructed for input spatial line segments. Solid lines represent the line segments, while dashed lines indicate the Minimum Bounding Rectangles (MBRs) of the partitions. (a) shows the partitioned input spatial lines. (b) shows the DCEL vertices and half-edges. (c) the resulting DCEL after dangle and cut edge removal. Finally, (d) shows the final DCEL faces. (taken from [1]).

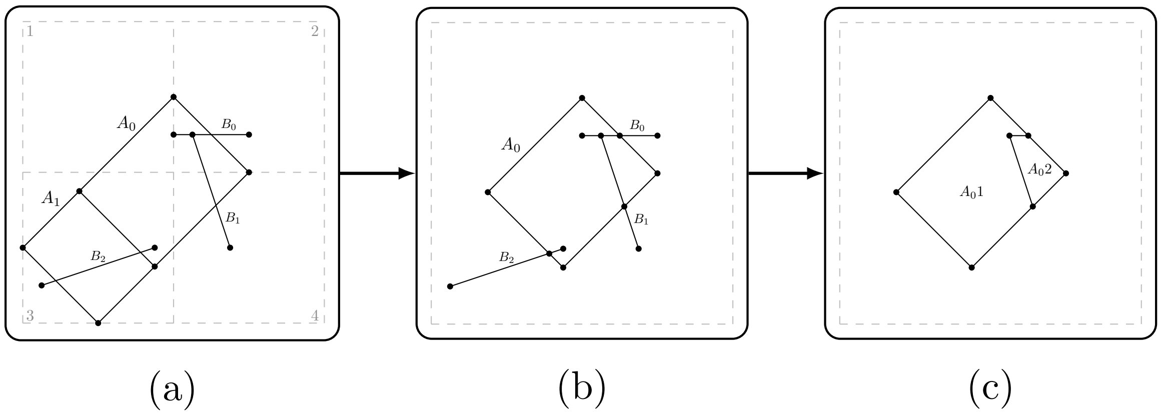

Figure 1.3: (a) Spatial partitioning of input layers A and B, (b) Re-Partitioning of polygon A_0 with edges it intersects with, and (c) the result of polygonization of A_0 with B_0, B_1, B_2.

generating new vertices and half-edges, marking the current dangle and cut edges, setting the next pointers, and finally constructing the partition polygons. Figure 1.3(c) illustrates the result of polygonizing the edges from polygonA_0 and B_0, B_1, and B_2, yielding two polygons,$A_0$1 and $A_0$2. The polygons generated from all partitions together form the overlay between polygon layer A and layer B.

1.3 Experimental Evaluation

1.3.1 Kd-tree versus quadtree performance

To compare the quadtree and kd-tree partition strategies, we analyze their performance across several stages: constructing the spatial data structure to define the partition cells based on the sample, the cost of partitioning, populating the cells with the full datasets, and the overall time required to complete each phase of the overlay operation using each partitioning approach. We use the MainUS and GADM datasets, as described in Table **??**.

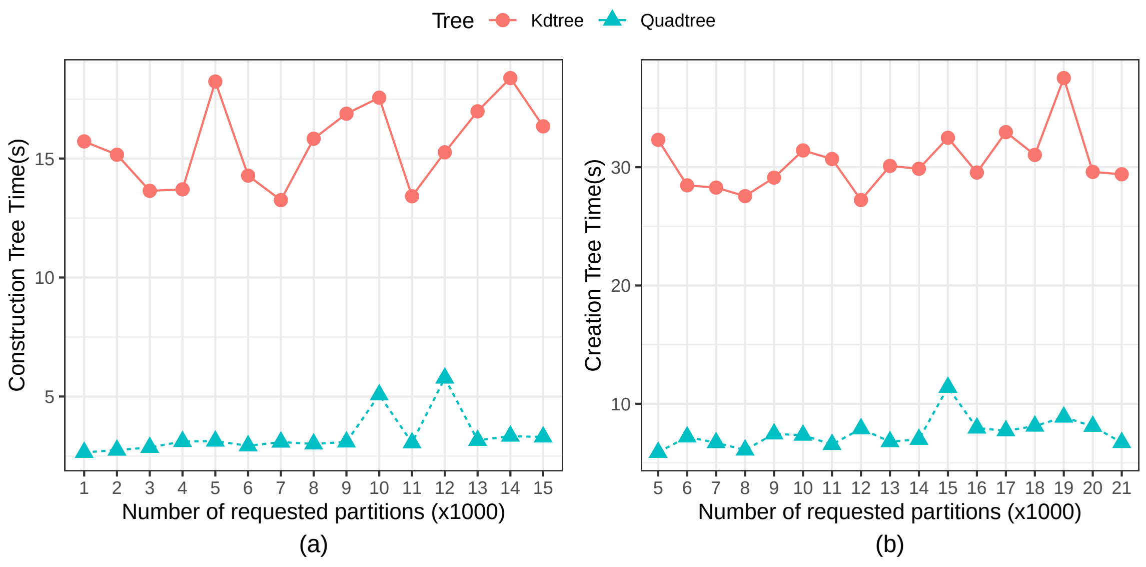

Figure 1.4: Construction time for the spatial data structure in the (a) MainUS and (b) GADM datasets.

Figure 1.4 illustrates the construction time for sampling the input layers and generating partitioning cells with varying numbers of divisions. The kd-tree requires more time, primarily due to the sorting involved at each split to organize the data and locate the midpoint. On average, the quadtree takes only 23.13% of the time needed to create the kd-tree (21.55% for MainUS and 24.72% for GADM). However, kd-tree creation accounts for only 5.86% of the total time required for complete DCEL construction (6.88% for MainUS and 4.87% for GADM).

An important characteristic of each partitioning scheme is the number of cells (partitions) generated by each sample data structure. Figure 1.5 shows the number of cells created by each spatial data structure. Since the quadtree follows a space-oriented technique, it creates more nodes (four at each split), resulting in a larger number of leaf

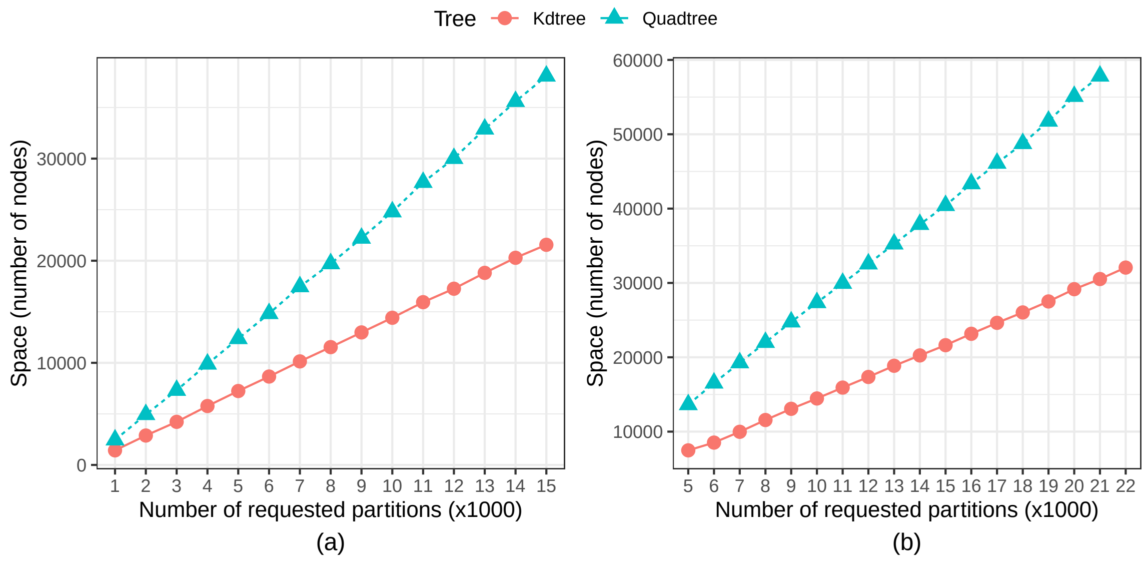

Figure 1.5: Number of cells created by each spatial data structure in the (a) MainUS and (b) GADM datasets.

cells, many of which are likely to be empty compared to those generated by the kd-tree.

Figure 1.6 presents the cost of partitioning the full content of both layers. Based on the sample tree data structure, each edge is assigned to a cell (partition) according to the leaf in which it is located; edges are assigned (or duplicated) to all leaves they intersect. A shuffle operation is then performed to move the data to the corresponding node responsible for handling each cell (partition). The figure shows that quadtree partitioning takes more time, primarily due to the larger number of leaves generated by the sample tree and the higher number of edges overlapping multiple partitions, which is expected with the quadtree's use of smaller, more numerous cells.

Once the data is assigned to their respective partitions, the overlay operation can be executed. Figure 1.7 illustrates the overlay performance for each partitioning strategy

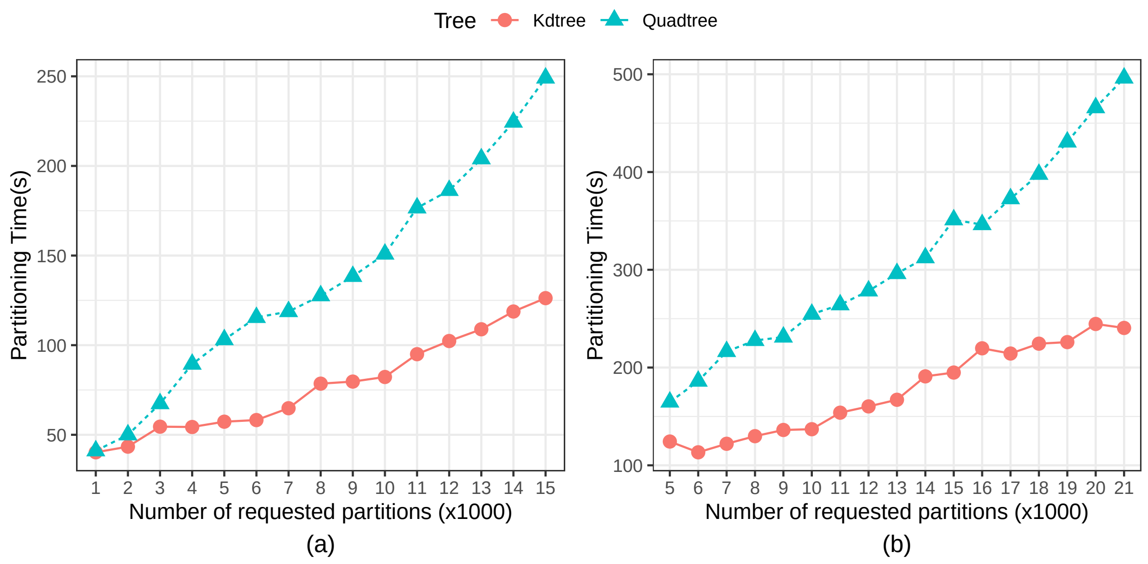

Figure 1.6: Data partitioning time using a spatial data structure (a) in the MainUS dataset and (b) in the GADM dataset.

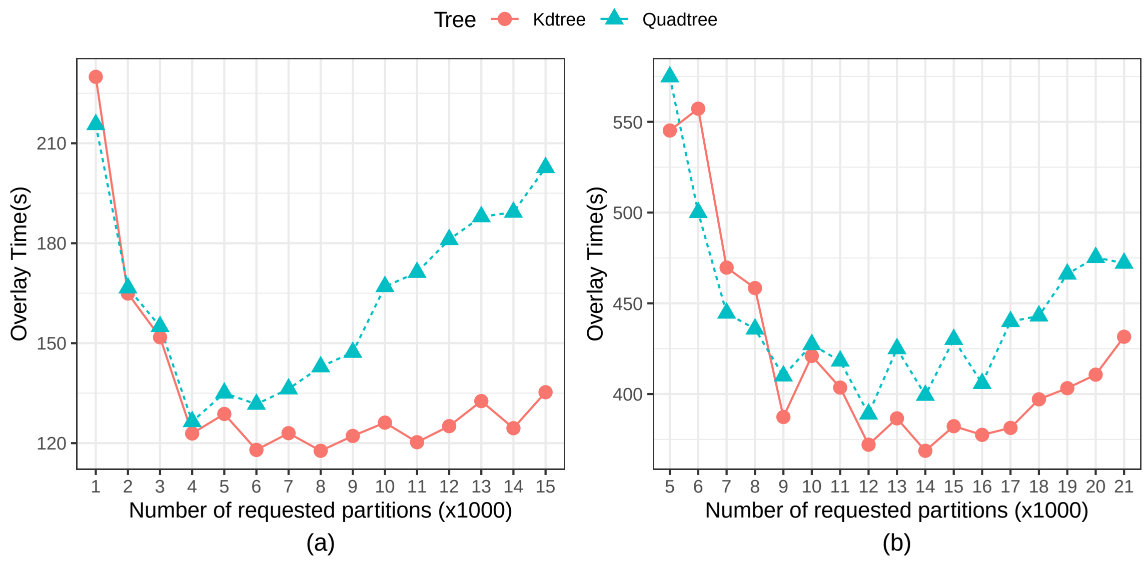

Figure 1.7: Execution time for the overlay operation using a spatial data structure in the MainUS (a)and GADM (b) dataset.

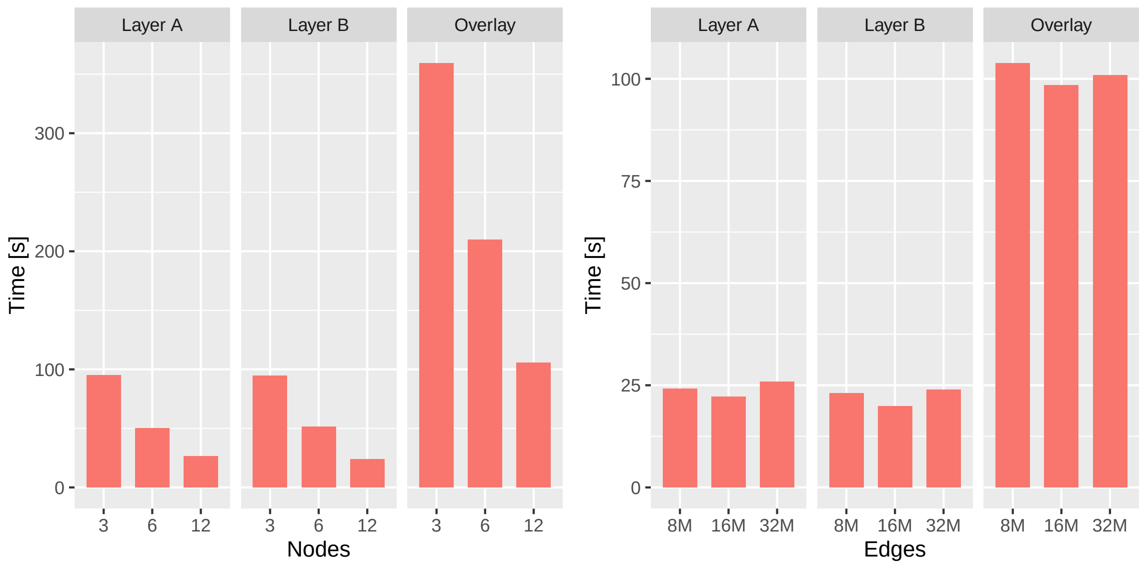

Figure 1.8: (a)Speed Up and (b) Scale Up performance of the Kdtree partitioning using the MainUS dataset.

with varying numbers of cells. The kd-tree approach performs better, as the quadtree's tendency to generate a higher number of empty cells negatively impacts its performance.

Finally, we evaluate the speed-up and scale-up performance of the kd-tree partitioning. Figure 1.8(a) presents the speed-up performance for the MainUS dataset (36 million edges) as the number of nodes varies (3, 6, and 12 nodes). Similar to the quadtree partitioning strategy, the kd-tree partitioning demonstrates strong speed-up performance. Doubling the resources nearly halves the execution time, indicating effective scalability.

Figure 1.8(b) illustrates the scale-up performance of the kd-tree partitioning approach. Following the procedure outlined in Section **??**, we generated datasets with 8M, 16M, and 32M edges from the MainUS dataset and applied the kd-tree partitioning strategy using 3, 6, and 12 nodes, respectively. The kd-tree partitioning demonstrates strong

Table 1.1: Overlaying Polygons with Dangle and Cut Edges Dataset

Dataset	Number Layer A of Polygons	Number of Layer B Edges	Result Polygons
TN	1,272	3,380,780	41,761
GA	1,633	4,647,171	49,125
NC	1,272	7,212,604	22,413
TX	4,399	8,682,950	98,635
VA	1,554	8,977,361	38,941
CA	7,038	9,103,610	96,916

scale-up performance, maintaining consistent speed-up as the load per node remains nearly equal.

1.3.2 Overlaying Polygons with Dangle and Cut Edges

In this section, we examine the performance of overlaying polygons with dangle and cut edges resulting from the polygonization process, as detailed in 1.2.1. Table 1.1 presents the number of polygons per state for the first overlay layer, the number of dangle and cut edges per state for the second overlay layer, and the number of resulting polygons per state.

From Figure 1.9shows that the running time is influenced by both the number of dangle and cut edges and the number of intersections between the two layers (indicated by the number of generated polygons). TN and GA have relatively fewer dangle and cut edges, leading to lower execution times compared to VA, TX, and CA. However, because NC has

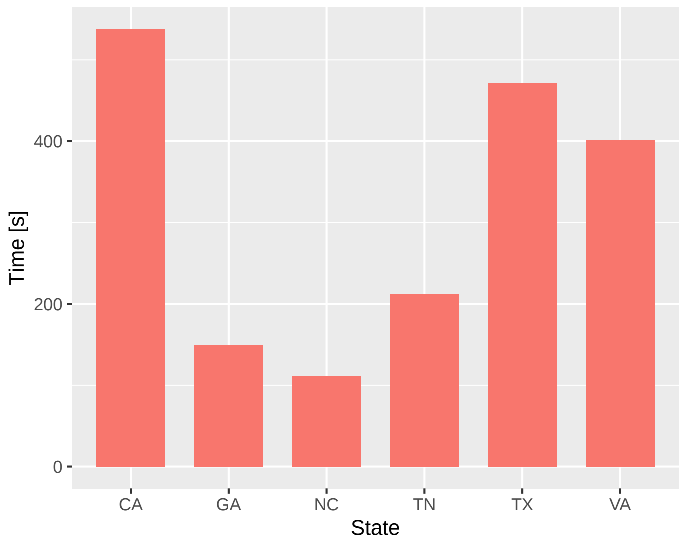

Figure 1.9: Overlaying State polygons with dangle and cut edges.

significantly fewer intersections than TN and GA, it exhibits the lowest execution time overall. While TX, VA, and CA have a comparable number of edges, VA's lower number of intersections results in a shorter execution time compared to TX and CA.

Bibliography

[1] Laila Abdelhafeez, Amr Magdy, and Vassilis Tsotras. DDCEL: Efficient Distributed Doubly Connected Edge List for Large Spatial Networks. In *2023 24th IEEE International Conference on Mobile Data Management (MDM)*, pages 122–131, 2023.

[2] Andres Calderon, Vassilis Tsotras, and Amr Magdy. Scalable Overlay Operations over DCEL Polygon Layers. In *Proceedings of the 18th International Symposium on Spatial and Temporal Data*, SSTD '23, pages 85–95, New York, NY, USA, 2023. Association for Computing Machinery.