I. Introduction

The use of spatial data structures is ubiquitous in many areas ranging from computational geometry, robotics and geographic information systems. It has important advantages to offer to current spatial algorithms in particular the application of spatial indexes and opportunities to exploit proximity among the data. Many works depends on it to ensure efficiency for example spatial joins, voronio diagrams and robot motion planning [1].

Although spatial data structure such as grids, the different flavours of rtrees and quadtrees are widely reported in literature, other structures have been less the focus of attention. In particular, edge-list structures appear as a technique used in diverse applications, specially related with geometric computation, but their employment have been limited to very specific domains. The use of such kind of structures has been reported in application like obtaining silhouettes of polyhedra, efficient Minkowksi sums and offset of polygons and diverse types of triangulations mainly in computational geometry projects.

The most representative data structure in the edge-list family is the Double Connected Edge List (DCEL). A DCEL [2], [3] is a data structure which collect topological information of the edges, vertices and faces contained by a surface in the plane. The DCEL and its components represent a planar subdivision of that surface in the plane. In a DCEL, the faces (polygons) represents the cells of the subdivision; the edges are boundaries which divide adjacent faces; and the vertices, itself, are boundaries between adjacent edges.

One interesting problem in which edge-list data structures have been shown utility are the thematic overlay maps. In this problem, two polygon input layers capture geospatial information and attribute data for different kind of phenomena. In many areas, such as ecology, economics and climate change, it is important to be able of join those layers and match their attributes in order to unveil patterns or anomalies in data. Several operation are relevant depending of the study, for instance, sometime the user would like to find intersections between the layers and other times be able to see the difference between them.

A. *What is the problem?*

Even the DCEL data structure has interesting advantages for overlay map operations, current methods are sequential based. For layers with thousand of polygons the execution time is not feasible. Further more, most of distributed techniques that have been used in this matter are oriented to a specific spatial operation (intersection, union or difference) and they have to be run from scratch if other operation is required. In addition, parallel techniques divide the data into partitions and replicate features if needed in order to solve the problem locally which could increase the size of the problem. Up to now, data structures collecting topological properties, like the DCEL, are not explored in a distributed and scalable fashion.

B. *Why is it important?*

With the scale and volume of available geodata, the rise of big (spatial) data makes necessary to count with fast and efficient techniques for spatial analysis. For example, today GIS researchers have to deal with spatial operations between layers collecting thousand of counties nation-wide. The versatility and efficiency of the spatial methods is cardinal for their studies. Given the advantages shown by the DCEL data structure, it should be interesting to count with intermediate data structures that allow multiple map overlay queries and exploit distributed frameworks at the same time.

C. *What are the limitations of related work?*

We already know that topological data structures are common in computational geometry. However, most implementations are sequential and they do not scale appropriately on large spatial datasets. In addition, distributed alternatives like parallel spatial joins add complexity due to spatial partitioning and they are unable to run more than one operation once they have been created.

D. *Why is it challenging?*

However, adapt the design and implementation concepts of the DCEL to a distributed environment should face challenge such as how to deal with initial stages of the construction which are expecting to fit in main memory. In addition, the partition strategy must divide the data appropriately and collect back the results without any data loss. Also, such development must guarantee that subsequent Boolean operations should be able to query the DCEL in a transparent way.

At the best of our knowledge, there is not a distributed and scalable DCEL implementation available that meet those criteria. We think it could be a great tool to support key challenges and operations in Geoscience today.

II. Literature review

The data explosion we are living today has impacted not just common areas of our daily lives such as our health, economy or social relationship but also how we move and how we locate ourselves in space. Nowadays, huge amount of information with a spatial component are generated. For example, GPS devices and smart-phones are able to collect your location and support navigation queries. Similarly, earth observation satellites captures gigabytes of imagery all around the world at daily basis.

This new availability has brought novel and interesting applications but also it is demanding most difficult challenges. The special characteristics of spatial data (i.e. topology, precision, types of data) demands special techniques, particularly when we are dealing with large and heterogeneous datasets. The study of efficient algorithms to deal with spatial data has touched diverse domains such as computer sciences, mathematics and physical geography. In one of these aspects, the computational geometry has explored different techniques and data structures to support such efficiency.

Particularly, one of this problem is related to special operations between layers of geographic data. One of the most commons outputs of geographic information systems are maps. They can provide interesting information in an agile and visual manner, and it is not a surprise than many areas are interested on performing further analysis from this kind of data. For example, ecology map users are usually interested on discovering areas where different attributes co-exist, for instance, the presence of certain vulnerable animal species and the impact of road construction or accessibility to water resources.

This kind of queries are supported by binary operators such as intersection, union and difference. They are implemented in different ways and their applications are widely reported in literature. Most of this solutions are based on edge-list data structures. Perhaps the most common implementation of this kind of structures is the DCEL (Double Connected Edge List). The fundamentals of the DCEL are explained in the seminal paper of [2] and illustrated with more examples in [3]. The authors highlight among the main advantages of DCELs the opportunity of capturing topological information and allowing multiple overlay operations once the DCEL is created. In addition, a DCEL can be constructed in $\mathcal{O}(nlog(n))$ time using $\mathcal{O}(n)$ additional memory. Once created, the DCEL allows Boolean overlay operations in $\mathcal{O}(n)$ time using $\mathcal{O}(n)$ additional space.

Nowadays, implementations of DCEL data structures have been presented and used in diverse applications. [4] presented the description and modelling of a database-oriented implementation of a DCEL. It presents a geometric package aims to support polyhedra or polyhedral surface operations. The design is discussed in detail together with application examples for collision detection, gap filling and inter-slice interpolation.

In [5], the author explains in more detail the different spatial operations (intersection, union and difference) supported by a DCEL while describes a sequential implementation over the Computational Geometry Algorithms Library - CGAL. This work allows, given two simple polygon layers, the intersections of their boundaries keeping track of related attributes of the inputs.

A cardinal reference for DCEL description and design should be [1]. It describes the core of the DCEL construction algorithm but also go deeper in the explanation of related concepts and techniques used during the DCEL construction such as line segment intersection and plane sweeping methods. It also compiles an number of well-explained examples and application of its usage, for instance, triangulations, point location and robot motion planning.

[6] describes the use of sequential DCELs to store the needed topological information of mesh networks extracted from LiDAR data to reconstruct roof polygons in 2D. They highlight the usefulness of 2D topological data structures to avoid the complexities in data storage and handling of 3D configurations.

Although there is not reference to distributed DCEL implementations, other dynamic parallel data structures has been described. For example, the DD-Rtree [7] claims to preserve spatial locality while distributing data across compute nodes. By contrast to static counterparts, it can be constructed incrementally making it useful for handling big data. It evaluates its communication cost, query time and data performance of data mining algorithms. In addition, this work provides an interesting survey of multi-dimensional indexing structures and their parallel versions.

Similarly, [8] describe different techniques to support distributed spatial joins, including options to perform spatial partitions in layers previous to the join phase. Together with this, it evaluates cost-based and rule-based query optimization. Spatial joins could support overlay operations in certain way but just for an individual operator at a time. [9] also explore parallel support for common computational geometry operations including polygon union, convex hull and skylines. They apply the MapReduce paradigm over the Hadoop framework presenting the CG_Hadoop tool. However, it just supports limited number of operations over an individual input layer once at a time.

[10] also present a survey of parallel spatial algorithms but it focus on GPU and multi-core architecture. They mention hierarchical tree structures (octree or rtree), location point algorithms, line and plane sweep algorithm, cluster-based parallel map overlays, among others. Also, they mention the use of those techniques in the solution of diverse operations such as: union of large set of polygons, planar graphs, overlaying maps, 3D overlay triangulation and cross areas in overlay polygons.

[11] performs the map overlay in parallel, thereby utilizing the ubiquitous multi-core architecture. It uses a two-level grid partitioning strategy and proposed a conservative empirical formula for the grid size that gave a good execution time and a feasible memory size. Together with this, they implement exact computation using rational number to overcome round-off errors. As result, they provide the EPUG-Overlay package to support intersection operations.

[12] and [13] revisit the distributed polygon overlay problem and its implementation using the GPGPU model and the MapReduce paradigm respectively. [13] present the adaptation and implementation of a polygon overlay algorithm and describe

the system to execute a distributed version on a Linux cluster using Hadoop MapReduce framework. However, they only analyze and report intersection overlay operation since it is the most widely used and representative operation. They ported an MPI based spatial overlay system (GPC library) to Hadoop MapReduce platform and a grid based overlay algorithm with two alternatives: a single map and reduce phase, and a map phase only using distributed caches.

In [12], the research focuses to develop an array based parallel overlay algorithm which can be easily implemented on GPUs using prefix sum and sorting to, potentially, speedup overlay computation. In this approach, they use a distributed Rtree construction using a top-down schema and a load-balanced overlay processing system using the MapReduce paradigm. The proposed parallel algorithm uses a similar idea of a DCEL with the representation of a new set of polygons by the merge of edges from the initial layers and their intersections. However it uses scan lines to partition the set of edges and process them concurrently. As the previous work, they test the implementation only with the intersection overlay operation.

Currently, there are just a few sequential implementations available in the market. The most important are LEDA[1] [14], Holmes3D[2] [15] and CGAL[3] [16]. LEDA and Holmes3D are close-source software and their access is limited. On the other hand, CGAL is an open-source project with a large trajectory in the area of computational geometry offering a wide-ranging number of packages and modules to support diverse areas from modular arithmetic to geometric optimization.

CGAL offers a solid support for DCEL construction and operations thought the Arrangement package. Its design concepts, implementation and well documented applications are available by different sources ([17], [18] and [19]). [16] deserves a particular entry. It does just not demonstrate the features of the DCEL implementation but also it collects a large number of application in diverse and novel areas like computer-assisted surgery and molecular biology.

[1]https://www.algorithmic-solutions.com/
[2]http://www.holmes3d.net/graphics/
[3]https://www.cgal.org/

III. Methods

In many cases, the operation provides by a DCEL could involve large spatial datasets. Our aim is to offer a solution to deal with data volumes that the current sequential solution are unable to process. In order to reach that goal we are purposing a partition strategy to build a scalable DCEL in a parallel fashion.

The main idea of the strategy is to split the study area into a number of cells which could be processed independently in a local basis. We can take the case of overlay operations over layers of polygons as an example to explains the details. In the sequential approach, the DCEL for each layer will be built and then both will be merged to generate a structure where the operator can be applied.

Our goal is to create DCELs for each layer in parallel to be able to solve large datasets the sequential alternatives are unable. The proposal can be summarized in the following steps: (i) Partition the input polygons and build local DCEL representations of them at each partition; (ii) Merge the local DCELs for each layer together locally scaling the processing cost; (iii) Overlay operations will be run over the local merged DCELs to finally be collected back to mix and generate the final answer.

The proposed partition schema is illustrated in figure 1. We will use a sample of edges from both layers to create a set of cells which will spatially partition the space into disjoint areas. In the example, a simple grid is used but any spatial index can be applied (in our experiments we use quadtrees for better balance and data distribution). We will use those cells to clip the input polygons to generate new ones that will lie inside of the boundaries of the cell. Although it will increase the number of edges and the size of the data, we expect that that the gain during parallel processing will make the addition worthwhile.

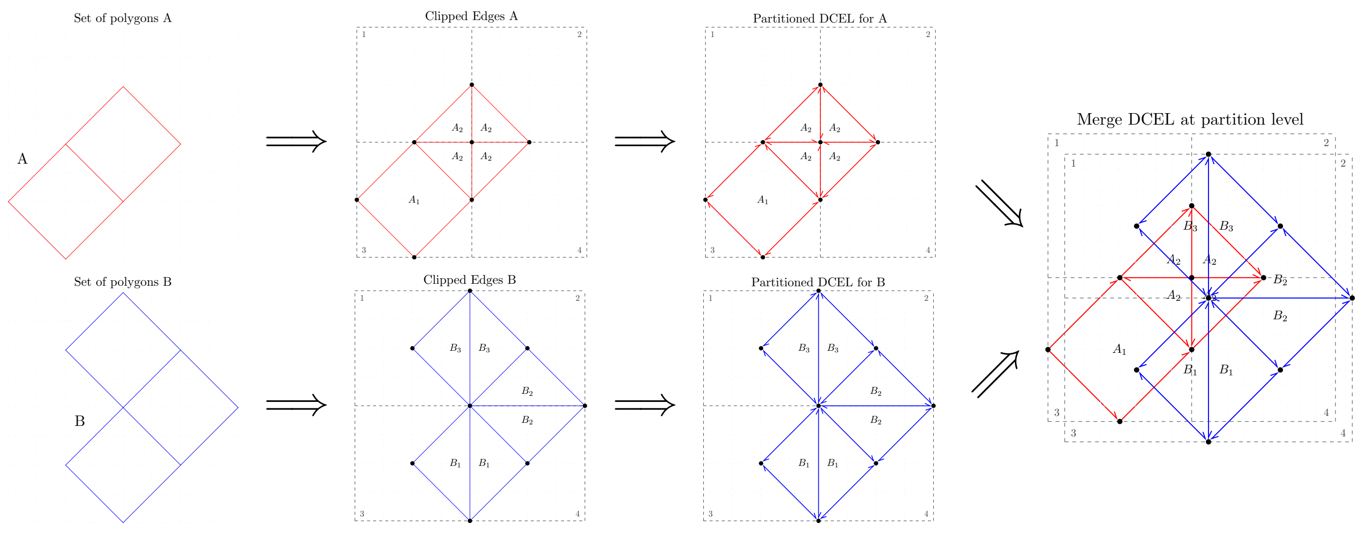

Fig. 1. Partition schema.

Now each cell have the enough data to build a DCEL representation of the new polygons for each layer. Data for each cell will be marked appropriately and submitted to different nodes to be processed in parallel. Note that the same partition schema (set of cells) is used in both layers. That is important due to it allows one-to-one matching between corresponding partitions in both layers.

During the local DCEL construction for individual layers, it is straightforward the connection between the edges inside of each polygon. From the inputs, it can easily be identified the position of each edge relate to its next and previous edges and to which face's (polygon's) boundary the edge belongs. Each edge is converted to a half-edge and their pointers are update accordingly. However, a query to identify the twin pointer of each half-edge is still needed given that the input polygons do not provide which polygons are in its neighborhood. The matching is done through a self-join query among the current half-edges to pair those which share the same vertices in opposite directions.

However, when we pair the partitions from the both layers, we got two local DCELs (each representing the polygons for each layer) and the processing we need now is a merge between both of them. It requires: (i) Identify the intersection points between the half-edges of each local DCEL and add them as new vertices; (ii) Split those half_edges involve in an intersection with a recently added vertex (prune duplicates if needed); (iii) Traverse the list of vertices and update their incident half-edges list; (iv) Update the pointers for next and previous half-edges in those affected by the splits; (v) Update the list of faces and their corresponding labels.

Figure 2 depicts an overview of the process taking as example the polygons and edges of partition 2 of figure 1. Similarly, figure 3 shows the full result of the merged DCEL once all the partitions have processed their corresponding edges. Note that red half-edges have been introduced artificially by the partition schema but they are marked accordingly to be used in the collect back process when we need to unify the results after the application of the overlay operations.

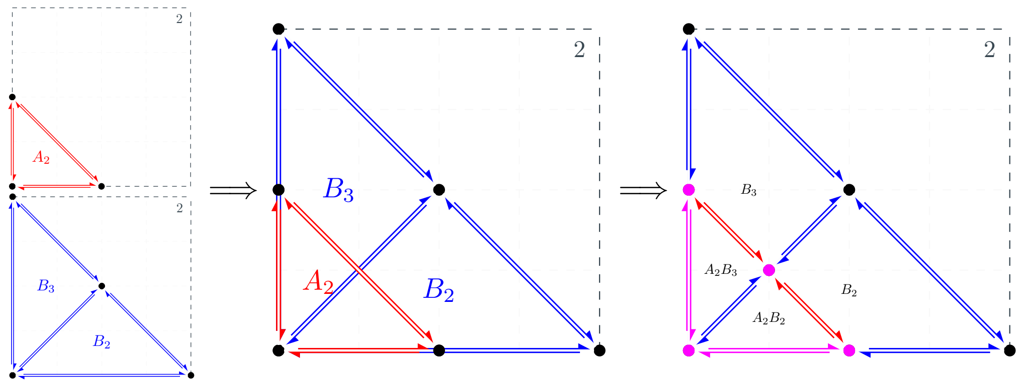

Fig. 2. Merge of local DCEL for partition 2.

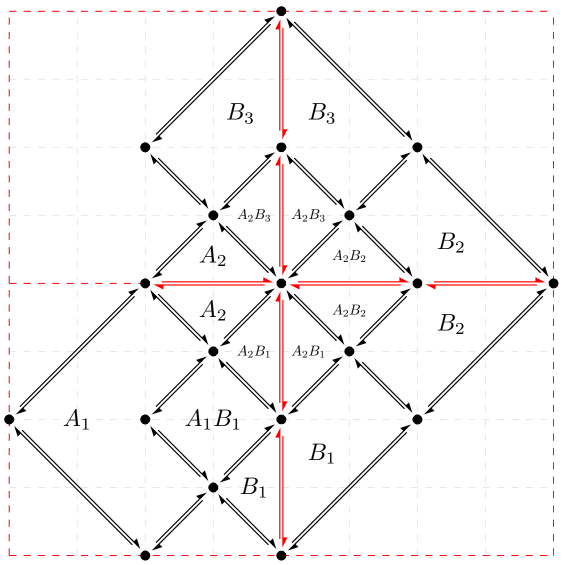

Fig. 3. Result of the merged DCEL.

At this point, we have access to a distributed spatial data structure which collects the individual DCEL representations of the full study area at local basis. It is easy to see that we can run overlay operations in parallel over the local DCELs and then just collect and merge the results to unify a final answer. For example, figure 4 illustrates the process to query for the intersection results over the input polygons described in figure 1.

Note that after getting the local results from the local DCELs at each partition, we still need to process those faces which touch the border and can, potentially, have additional sections to be merged in contiguous partitions. We keep the faces with marked half_edges for further processing but those ones not involved can be reported immediately. We use the information of the marked half_edges to match those that occupy the same place but point on opposite direction. The faces of those half_edges are dissolved in one and the involved marked half_edges are removed.

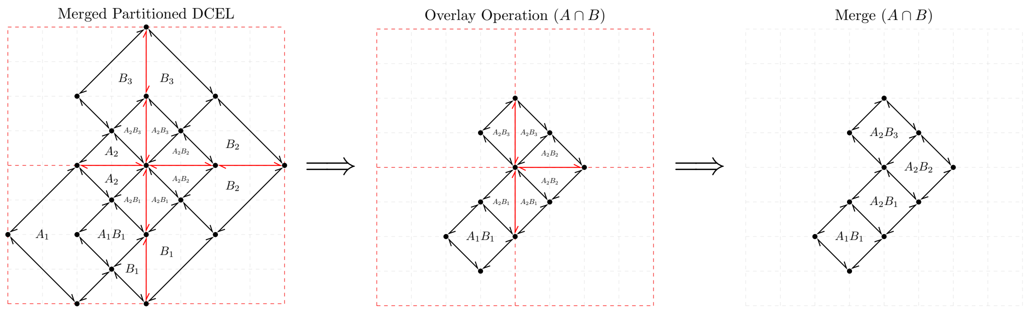

Fig. 4. Example of an overlay operation querying the distributed DCEL.

Figure 5 shows the results of the five overlay operations supported by the scalable DCEL. To obtain the results we query locally the DCEL filtering the faces according to the characteristics of its label. For intersection ($A \cap B$), it filters just faces where its label contains both letters (A and B); On the other hand, for symmetric difference ($A \triangle B$), it filters faces where its labels contains just one of the letters (A or B). For the case of difference between the layers ($A \setminus B$ or $B \setminus A$), it filters faces and labels according to the requested letter (either A or B). In the case of union ($A \cup B$), all the faces are retrieved.

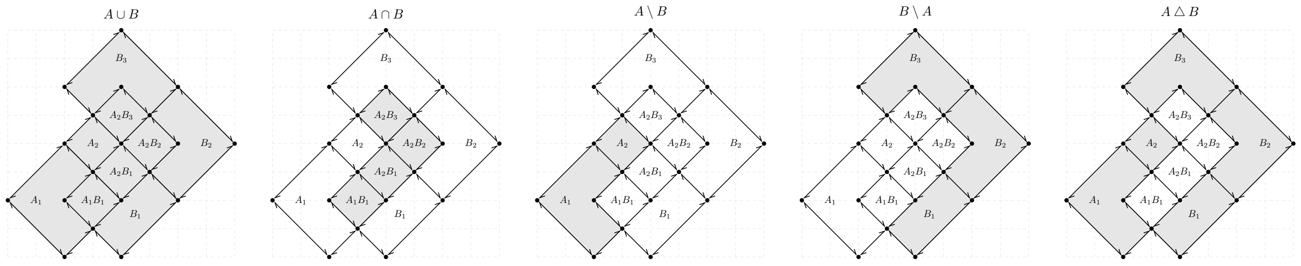

Fig. 5. Results of the overlay operations supported by the scalable DCEL.

A. *Partition/Cell inside polygon problem*

The main goal of the proposal is to be able to divide the problem into smalls partitions for efficient processing. Each partition collects the needed data and it is able to build its local DCEL without the need of query other partitions. However, under this partition strategy, a new problem arises. It happens when the partition schema (i.e. a quadtree) deliver a cell where no edges for any of the input layers are located. The problem is even more complicated when a just hole in located inside a cell (figure 6). The problem is that the empty cell (or the empty portion in the case of holes) has no access to which polygon it belongs making its corresponding labeling impossible.

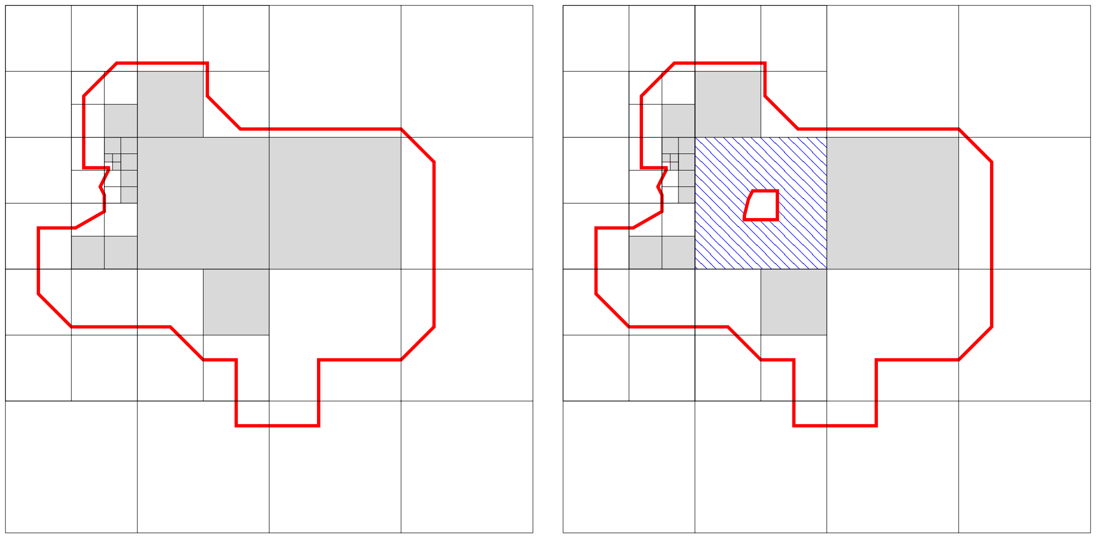

Fig. 6. Example of empty cell and empty cell with holes cases.

To solve the problem, an algorithm is proposed to find the next cell in the valid information about the polygon they are contained. It is based on the branch information of the cell inside of the spatial index structure used during partition, also know as lineage (see figure A for an example). For the sack of explanation, we will assume we use a quadtree, but the proposal could be easily adapted to other data structures such as grids.

After the partitioning strategy, a set of the cells is available with the following information: an unique cell identifier (id); a lineage, a string which provides the position and depth of the cell into the spatial index; and an envelope which is a polygon representation (a rectangle) of the boundaries of the cell.

The key of the proposal is to identify the centroid of the parent cell from the empty cell in question. That point will allow us to retrieve the neighbour cells which can easily be queried if they have edges to extract the needed polygon information. If all of them are still empty, we proceed to choose that one with the deepest level and recursively repeat the process. Eventually, a non-empty cell will emerge and all the involved empty cells can be updated. Figure III-A shows a three iteration run of the algorithm with the example of figure 6.

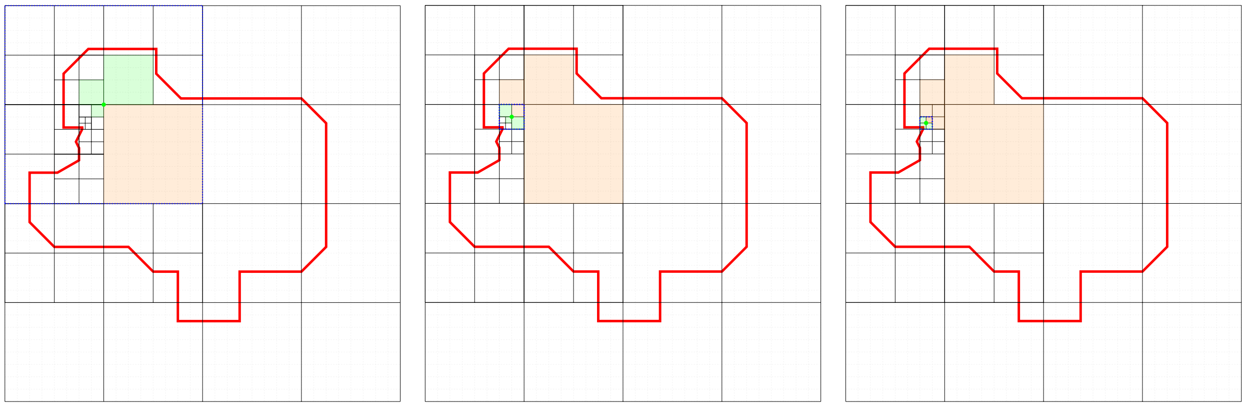

Fig. 7. Three iteration of the proposed algorithm to find the next cell with valid edges from a empty cell.

The details and pseudo code of the algorithm can be seen at appendix B. We based of proposal in the following lemma and used it to proved our point with the subsequent proof.

Lemma 1. *Four cells at the same level can not be empty. At least one of them must have edges in order to force the split.*

Proof. The GETCELLSINCORNER function will query the interior corner of a cell according to its position, that is the centroid of its cell parent. The only cells which can intersect that point are cells at the same level of the current cell or their children. If the 3 cells returned by GETCELLSINCORNER are empty, at least one of them must have a deeper level that the current cell. Following that cell guarantees that the search space will be shrank at each iteration. Eventually, the algorithm will reach the maximum level of the quadtree where all the involved cells will have the same level and, therefore, at least one of them must have edges. □

REFERENCES

[1] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin Heidelberg: Springer-Verlag, 2008. [Online]. Available: https://www.springer.com/us/book/9783540779735

[2] D. E. Muller and F. P. Preparata, "Finding the intersection of two convex polyhedra," *Theoretical Computer Science*, vol. 7, no. 2, pp. 217–236, Jan. 1978. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397578900518

[3] F. P. Preparata and M. Shamos, *Computational Geometry: An Introduction*, ser. Monographs in Computer Science. New York: Springer-Verlag, 1985. [Online]. Available: https://www.springer.com/us/book/9780387961316

[4] G. Barequet, "DCEL - A Polyhedral Database and Programming Environment," *International Journal of Computational Geometry & Applications*, vol. 08, no. 05n06, pp. 619–636, Oct. 1998, publisher: World Scientific Publishing Co. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0218195998000308

[5] W. Freiseisen, "Colored dcel for boolean operations in 2d," 1998. [Online]. Available: citeseer.nj.nec.com/freiseisen98colored.html

[6] D. Boltcheva, J. Basselin, C. Poull, H. Barthélemy, and D. Sokolov, "Topological-based roof modeling from 3D point clouds," in *Journal of WSCG and WSCG*, vol. 28, 2020, pp. 137–146.

[7] J. S. Challa, P. Goyal, S. Nikhil, A. Mangla, S. S. Balasubramaniam, and N. Goyal, "DD-Rtree: A dynamic distributed data structure for efficient data distribution among cluster nodes for spatial data mining algorithms," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec. 2016, pp. 27–36.

[8] I. Sabek and M. F. Mokbel, "On Spatial Joins in MapReduce," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3139958.3139967

[9] Y. Li, A. Eldawy, J. Xue, N. Knorozova, M. F. Mokbel, and R. Janardan, "Scalable computational geometry in MapReduce," *The VLDB Journal*, Jan. 2019. [Online]. Available: https://doi.org/10.1007/s00778-018-0534-5

[10] W. R. Franklin, S. V. G. de Magalhães, and M. V. A. Andrade, "Data Structures for Parallel Spatial Algorithms on Large Datasets (Vision paper)," in *Proceedings of the 7th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, ser. BigSpatial 2018. New York, NY, USA: Association for Computing Machinery, Nov. 2018, pp. 16–19. [Online]. Available: https://doi.org/10.1145/3282834.3282839

[11] S. V. G. Magalhães, M. V. A. Andrade, W. R. Franklin, and W. Li, "Fast exact parallel map overlay using a two-level uniform grid," in *Proceedings of the 4th International ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data*, ser. BigSpatial'15. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 45–54. [Online]. Available: https://doi.org/10.1145/2835185.2835188

[12] S. Puri and S. K. Prasad, "Efficient Parallel and Distributed Algorithms for GIS Polygonal Overlay Processing," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, May 2013, pp. 2238–2241.

[13] S. Puri, D. Agarwal, X. He, and S. K. Prasad, "MapReduce Algorithms for GIS Polygonal Overlay Processing," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, May 2013, pp. 1009–1016.

[14] K. Mehlhorn and S. Näher, "LEDA: a platform for combinatorial and geometric computing," *Communications of the ACM*, vol. 38, no. 1, pp. 96–102, Jan. 1995. [Online]. Available: https://doi.org/10.1145/204865.204889

[15] R. Holmes, "The DCEL Data Structure for 3D Graphics," 2021. [Online]. Available: http://www.holmes3d.net/graphics/dcel/

[16] E. Fogel, D. Halperin, and R. Wein, *CGAL Arrangements and Their Applications: A Step-by-Step Guide*. Springer Science & Business Media, Jan. 2012, google-Books-ID: u0CONtnwi9YC.

[17] E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra, "The design and implementation of panar maps in CGAL," *ACM Journal of Experimental Algorithmics*, vol. 5, pp. 13–es, Dec. 2001. [Online]. Available: https://doi.org/10.1145/351827.384255

[18] I. Haran and D. Halperin, "An experimental study of point location in planar arrangements in CGAL," Feb. 2009. [Online]. Available: https://doi.org/10.1145/1412228.1412237

[19] R. Wein, E. Fogel, B. Zukerman, and D. Halperin, “Advanced programming techniques applied to Cgal’s arrangement package,” *Computational Geometry*, vol. 38, no. 1-2, pp. 37–63, Sep. 2007. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925772107000077

Appendix A
Lineage example

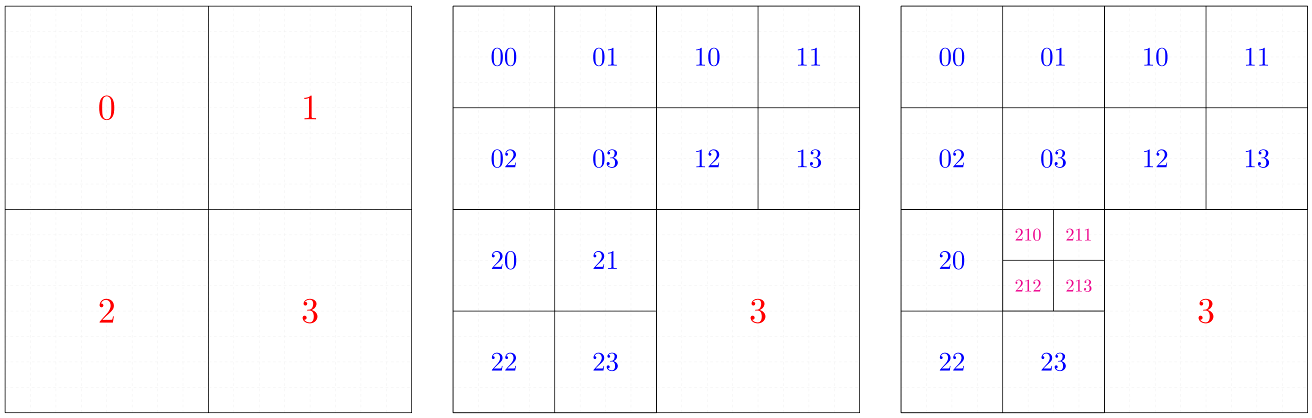

Fig. 8. Lineage can provide the cell's position (string's last character) and its depth (string's length).

Appendix B
Empty Cell Algorithm

Algorithm 1 GETNEXTCELLWITHEDGES algorithm

Require: a quadtree with cell envelopes $\mathcal{Q}$ and map of cells and their edge count $\mathcal{M}$.

```
function GETNEXTCELLWITHEDGES ( $\mathcal{Q}$, $\mathcal{M}$ )
    $\mathcal{C} \leftarrow$ list of empty cells in $\mathcal{M}$
    for each $emptyCell$ in $\mathcal{C}$ do
        initialize $cellList$ with $emptyCell$
        $nextCellWithEdges \leftarrow null$
        $referenceCorner \leftarrow null$
        $done \leftarrow false$
        while not $done$ do
            $c \leftarrow$ last cell in $cellList$
            $cells, corner \leftarrow$ GETCELLSATCORNER($\mathcal{Q}, c$)    ▷ return 3 cells and the reference corner
            for each $cell$ in $cells$ do
                $nedges \leftarrow$ get edge count of $cell$ in $\mathcal{M}$
                if $nedges > 0$ then
                    $nextCellWithEdges \leftarrow cell$
                    $referenceCorner \leftarrow corner$
                    $done \leftarrow true$
                else
                    add $cell$ to $cellList$
                end if
            end for
        end while
        for each $cell$ in $cellList$ do
            output($cell$, $nextCellWithEdges$, $referenceCorner$)
            remove $cell$ from $\mathcal{C}$
        end for
    end for
end function
```

Algorithm 2 GETCELLSATCORNER algorithm

Require: a quadtree with cell envelopes $\mathcal{Q}$ and a cell c.

```
1: function GETCELLSINCORNER ( $\mathcal{Q}$, $c$ )
2:     $region \leftarrow$ last character in $c.lineage$
3:     switch $region$ do
4:         case '0'
5:             $corner \leftarrow$ left bottom corner of $c.envelope$
6:         case '1'
7:             $corner \leftarrow$ right bottom corner of $c.envelope$
8:         case '2'
9:             $corner \leftarrow$ left upper corner of $c.envelope$
10:        case '3'
11:            $corner \leftarrow$ right upper corner of $c.envelope$
12:    $cells \leftarrow$ cells which intersect $corner$ in $\mathcal{Q}$
13:    $cells \leftarrow cells - c$                 ▷ Remove the current cell from the intersected cells
14:    $cells \leftarrow$ sort $cells$ on basis of their depth                 ▷ using $cell.lineage$
15:    return ($cells$, $corner$)
16: end function
```