Pyneapple-G: Scalable Spatial Grouping Queries

Laila Abdelhafeez
Department of Computer Science and Engineering
University of California, Riverside
labde005@ucr.edu

Andres Calderon-Romero
Department of Computer Science and Engineering
University of California, Riverside
acald013@ucr.edu

Amr Magdy
Department of Computer Science and Engineering
University of California, Riverside
amr@cs.ucr.edu

Vassilis J. Tsotras
Department of Computer Science and Engineering
University of California, Riverside
tsotras@cs.ucr.edu

ABSTRACT

This paper demonstrates *Pynapple-G*, an open-source library for scalable spatial grouping queries based on Apache Sedona (formerly known as GeoSpark). We demonstrate two modules, namely, *SGPAC* and *DDCEL*, that support grouping points, grouping lines, and polygon overlays. The *SGPAC* module provides a large-scale grouping of spatial points by highly complex polygon boundaries. The grouping results aggregate the number of spatial points within the boundaries of each polygon. The *DDCEL* module provides the first parallelized algorithm to group spatial lines into a DCEL data structure and discovers planar polygons from scattered line segments. Exploiting the scalable DCEL, we support scalable overlay operations over multiple polygon layers to compute the layers' intersection, union, or difference. To showcase *Pyneapple-G*, we have developed a frontend web application that enables users to interact with these modules, select their data layers or data points, and view results on an interactive map. We also provide interactive notebooks demonstrating the superiority and simplicity of *Pyneapple-G* to help social scientists and developers explore its full potential.

PVLDB Reference Format:
Laila Abdelhafeez, Andres Calderon-Romero, Amr Magdy, and Vassilis J. Tsotras. Pyneapple-G: Scalable Spatial Grouping Queries. PVLDB, 17(12): 4469 - 4472, 2024.
doi:10.14778/3685800.3685902

1 INTRODUCTION

Grouping spatial units, such as points, line segments, and polygons, are widely used in different spatial analysis operations, e.g., polygonization, overlay, or counting-based spatial statistics. These queries typically involve very large dataset sizes, pushing the boundaries of the traditional processing frameworks and demanding novel, highly parallelized, and large-scale frameworks.

Through collaborations with social scientists and domain experts, we have identified emerging challenges in different grouping queries, particularly regarding their performance scalability. The need for this research has been triggered during a collaboration with social scientists [4] to perform large-scale analysis on user-generated social media data at the world scale. In that study, we analyzed one billion tweets through location quotients and spatial Markov analysis over complex spatial polygons. The fundamental operations for this analysis are discovering polygons out of spatial road networks, aggregating counts of spatial points within each polygon, and comparing changes in polygons at different points in time. Due to the prohibitive cost of performing such aggregations, our study was limited to a small number of polygons and a portion of the available Twitter datasets. For example, running a single counting query using traditional filter-refine techniques on 100 million points over only 255 country borders takes 83 minutes, using a twelve-node Apache Spark cluster with a total memory of 1TB. In addition, a traditional polygon extraction process from the USA road network (152 million line segments) processes only 20 Million segments, i.e., 13% of the dataset, in four hours using PostGIS on a 64GB RAM machine with 1.8TB disk space, and then breaks down. Such inefficient runtime limits spatial data scientists from performing large-scale analysis on modern spatial datasets. To enable social scientists to perform large-scale studies of world-scale data, it is pivotal to efficiently support such fundamental operations on modern large-scale datasets for real complex polygons and multiple spatial scales.

This paper demonstrates *Pyneapple-G*, an open-source library for scalable spatial grouping queries. *Pyneapple-G* builds upon our research [1–3, 5] on scaling up grouping spatial points, grouping lines, and polygon extraction through the Doubly-Connected Edge List (DCEL) data structure, in addition to scalable polygon overlays. These three queries are heavily used in various spatial statistical analysis applications, such as spatial regionalization, spatial harmonization, segregation analysis, join-count analysis, hot-spot and cold-spot analysis, and spatial autocorrelation analysis. The DCEL is a popular data structure used to represent planar subdivisions in a wide variety of applications, such as Voronoi diagrams, planar graphs, polyhedron, and TIN data. It is further utilized in graph simplification, triangulation, subdivision traversal, and topology manipulation. Thus, supporting such operations efficiently at scale empowers many applications that are currently limited to exploit large spatial datasets.

Pyneapple-G consists of two modules; SGPAC [1, 3] and DDCEL [2, 5]. SGPAC efficiently supports grouping large-scale datasets containing hundreds of millions of data points by real polygons with very complex perimeter geometries (i.e., tens of thousands

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685902

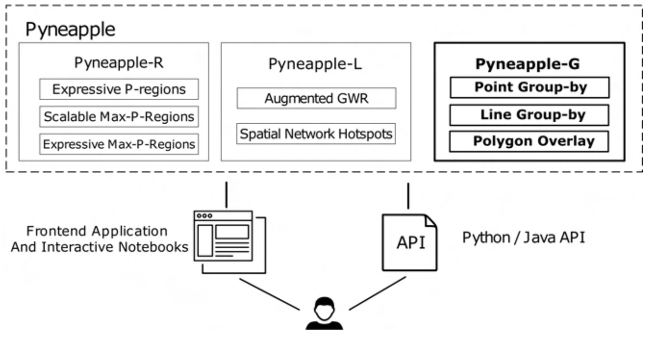

Figure 1: *Pyneapple* Overview

of perimeter points). DDCEL supports the polygonization of large-scale datasets with hundreds of millions of lines of real complex networks and scalable overlay operations on large-scale polygon layers. Existing approaches face several challenges in efficiently handling such operations on modern datasets. The first challenge arises from the prohibitive computations of point-in-polygon checks on real complex polygons due to the excessive number of points on the polygon perimeter. The second challenge is the high skewness of real spatial data due to skewed spatial distributions of the data generators, e.g., web users or city sensors. Due to load imbalance, such skewness leads to prohibitive runtime costs in parallel algorithms. The third challenge is reducing the communication overhead and handling parallel dependencies coming from distributing DCEL polygon extraction over multiple computing nodes.

Pyneapple-G overcomes these challenges and provides efficient algorithms that smartly address the bottlenecks to significantly reduce the runtime while ensuring the correctness of the output. Using our techniques, counting points of a Twitter dataset containing 100 million geotagged tweets over the world countries' polygon layer with an average number of perimeter points of 1,345 took 30 seconds compared to 83 minutes for the distributed filter-refine approach. Moreover, extracting polygons from the USA road network (152 million line segments) using DDCEL runs in under 2 minutes compared to the breakdown of the previous approaches.

Attendees at our demonstration can acquaint themselves with *Pyneapple-G* through a frontend application, which allows interactive file selection and visualization of results. Additionally, interactive Jupyter Notebooks for Python enthusiasts will showcase API usage and the edge of our modules. Subsequent sections offer an in-depth exploration of the *Pyneapple-G* library (Section 2) and the demonstration scenarios (Section 3).

2 *PYNEAPPLE-G* OVERVIEW

Pyneapple-G is an integral sub-package of the more extensive *Pyneapple* library [8], which is currently under active development with more features being added. Figure 1 depicts an overview of *Pyneapple*. The current version of *Pyneapple* comprises three main sub-packages: *Pyneapple-R* for regionalization queries [6, 7], *Pyneapple-L* for machine learning (ML) assisted analysis, and ***Pyneapple-G*** (bolded in the figure) for group-by and overlay queries.

Pyneapple-G consists of two modules: (1) Spatial Groupby Polygon Aggregate Counting (SGPAC) [1, 3]: a module that groups points within polygons of highly complex perimeters. (2) Distributed Doubly-Connected Edge List (DDCEL) [2, 5]: a module that groups line segments into polygons and overlay polygons at a large scale. Each module in *Pyneapple-G* is equipped with thorough Python and Java API documentation, facilitating a seamless integration into the broader data science landscape. The rest of this section delves deeper into the specifics of modules and queries housed within the *Pyneapple-G* sub-package.

2.1 Point Group-by Queries (SGPAC Module)

The SGPAC [1, 3] query processing framework exploits partitioning to significantly reduce the complexity of the perimeters of the polygons through two-level clipping and distributing large volumes of spatial points over several machines, contributing to reducing computation overheads and scaling up processing. It facilitates a global distributed spatial index to partition data points and query polygons across distributed machines (worker nodes). Each worker node j covers a specific spatial area represented with a minimum bounding rectangle (MBR) B_j. Then, on each worker node, the local portion of data points is indexed with a local spatial index, which does not necessarily have the same structure as the global index. The local index further divides data into small chunks. Meanwhile, when a new query polygon set L arrives, each worker node receives a subset L_j of query polygons that overlap with its partition MBR B_j; that is, for all $l_i \in L_j$, $l_i \cap B_j \neq \phi$. Each polygon $l_i \in L_j$ goes through a *Two-level Clipper* module that significantly reduces the complexity of its perimeter through two phases of polygon clipping. The first phase is based on the global index partition boundaries B_j. This phase replaces l_i with $l_i \cap B_j$, its intersection with the partition MBR, as any part of the polygon outside B_j will not produce any results from the data points assigned to node j. The newly clipped polygon l_i is passed as an input to the second level of clipping, which further clips l_i based on the local index partitions to produce multiple smaller polygons, each of them corresponding to one of the local index partitions and clipped with its MBR boundaries.

After the two-level clipping of input polygons, the query input turns into small crumbles of local data partitions and simple query polygons fed to a multi-threaded *Point-in-Polygon Refiner* module. This module takes pairs of data partitions and clipped query polygons with overlapping boundaries, where each pair follows one of two cases. The first case is that the boundaries of the local partition and the clipped query polygon are the same. This means the local partition is wholly contained inside the query polygon, and all data points are counted in the result set without further refinement. The second case is that the clipped query polygon intersects with part of the local partition boundaries. In this case, the refinement module iterates over all the points within the local partition. It uses the standard point-in-polygon algorithms to filter out points outside the polygon boundaries. Such point-in-polygon operation is much less expensive on the clipped polygon than the original one, with up to an order of magnitude cost reduction. Each thread maintains a list of $< polygonid, count >$ pairs that record the count of points in each polygon. Lists of pairs from different threads and partitions are forwarded to a shuffling phase that aggregates total counts of each input polygon, based on polygon IDs, in a similar fashion to the standard map-reduce word counting procedure.

To support efficient performance on various query workloads, we proposed a query optimization technique that distinguishes query polygons that are simple enough for which a plain filter-refine approach would suffice (i.e., *SGPAC* adds unneeded overhead). Furthermore, our query processing and optimization techniques are generalized for any underlying distributed spatial index structures.

2.2 Line Group-by Queries (DDCEL Module)

The *Doubly-Connected Edge List (DCEL)* is a popular data structure used to represent planar subdivisions. Given an input spatial network represented by a set of line segments, the DCEL constructor generates and stores a record for each subdivision's vertex, half-edge, and face. A vertex in a subdivision is a node where two or more line segments meet, corresponding to a graph vertex of the spatial network. A half-edge is a line segment split along its length and has a directional component: an origin vertex and a destination vertex. Two opposite-direction half-edges (twin half-edges), where the origin of the first is the destination of the second and vice versa, represent each undirected line segment. So, each half-edge corresponds to a directed graph edge of the spatial network. The face of a subdivision is a polygonal region whose boundary is formed by the subdivision's vertices and half-edges with the same direction.

To generate a DCEL object representing an input spatial network on distributed big-data systems, we face two main challenges: (1) First, the extraction of vertices and half-edges depends on one data record only, i.e., the line segment. The extraction process can be distributed directly along with the data records. However, in the case of the faces, one face depends on multiple data records, i.e., connected line segments. Such records do not necessarily end up in the same data partition. (2) Second, to parallelize the polygonization procedure, data partitions need means to share data. This data communication severely affects performance, especially since these data partitions do not necessarily reside in the same machine.

To overcome these challenges, we proposed a novel *Distributed DCEL (DDCEL)* [2] data structure extending the well-known DCEL to work in a scalable way. Like DCEL, the DDCEL data structure consists of three collections that store the subdivision's vertices, half-edges, and faces in a distributed way. Given an input spatial network, the DDCEL constructor aims to populate these collections. To achieve this goal, the constructor undergoes a two-phase paradigm: (1) $\underline{\text{Gen}}$erate each partition DCEL (*Gen Phase*), in which the vertices and the half-edges collections are fully populated, whereas only a portion of the faces collection is generated. (2) Generate the $\underline{\text{Rem}}$aining Faces (*Rem Phase*), in which the polygonization procedure proceeds to generate the remaining faces.

2.3 Polygon Overlay Queries (DDCEL Module)

A primary usage of the DCEL is computing the overlay of two polygon layers. Using DCELs for overlay operations offers the advantage that the result is also a DCEL, which can be directly used for subsequent operations. For example, creating an overlay between the intersection of two layers with a third layer. Even though the DCEL has essential advantages for implementing the overlay operations, current approaches are sequential in nature and do not scale for layers with thousands of polygons.

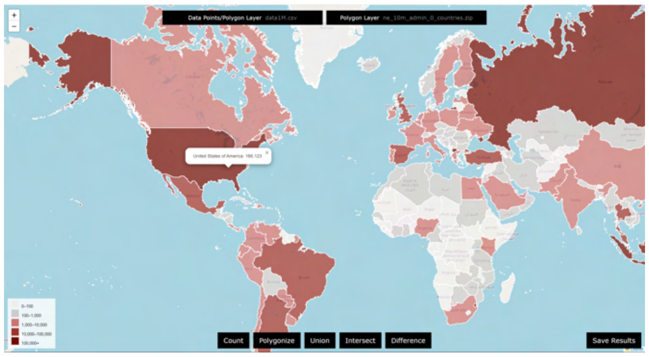
Data Points/Polygon Layer data1M.csv
Polygon Layer ne_10m_admin_0_countries.zip
United States of America: 166,123
0–100
100–1,000
1,000–10,000
10,000–100,000
100,000+
Count
Polygonize
Union
Intersect
Difference
Save Results

Figure 2: Counting Worldwide Tweets

Implementing a distributed overlay over DCELs creates novel problems. First, there are potential challenges that are not present in the sequential DCEL execution. For example, the implementation should consider features like holes that could lay on different partitions. Such features must be connected with their components residing in other partitions not to compromise the combined DCEL's correctness. Secondly, once a distributed overlay DCEL has been built, it must transparently support a set of binary overlay operators (namely union, intersection, difference, and symmetric difference). That is, such operators should take advantage of the scalability of the overlay DCEL and be able to run in a parallel fashion. Additionally, users should be able to apply the various operators multiple times without rebuilding the overlay DCEL data structure.

To address these challenges, we proposed *SDCEL* [5] as a scalable and distributed approach to compute the overlay between two DCEL layers. Given an input of two polygon layers, first, the polygon layers are partitioned in a way that guarantees that each partition collects the required data from each layer DCEL to work independently to minimize duplication and transmission costs. We then build local DCEL representations of them at each partition and compute the overlay of the DCELs at each partition. In addition, we present a merging procedure that collects all partition results and consolidates them in the final combined DCEL.

3 DEMONSTRATION SCENARIOS

To demonstrate *Pyneapple-G*, we design different scenarios for different groups of target audiences. Users will interact with the *Pyneapple-G* library through user-friendly web applications and Jupyter Notebooks, illustrating its ease of use and interactivity. Attendees will be able to visualize the results on an interactive map and gain a deep understanding of the potential applications of *Pyneapple-G* in various domains.

3.1 Scenario 1: Counting Worldwide Tweets

This scenario, depicted in Figure 2, demonstrates the usage of *Pyneapple-G* in a count-aggregate example. Assume a user is interested in exploring the tweet count per country. The user will click the *Data Points* button and choose the tweets CSV file. Then, choose the shapefile of the world countries for the *Polygon Layer* button. Upon pressing the *Count* button, the query is sent to the backend, and the corresponding module (in this case, the SGPAC

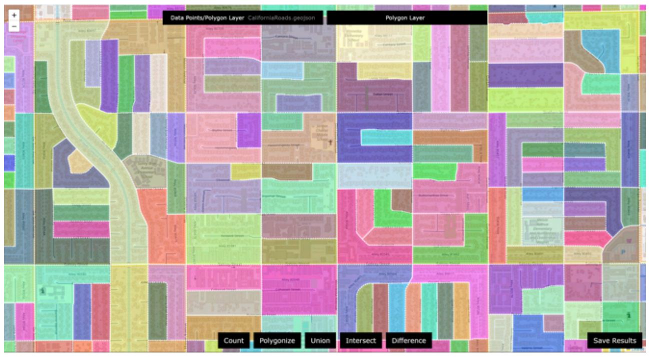

Figure 3: California Neighbourhood Extraction

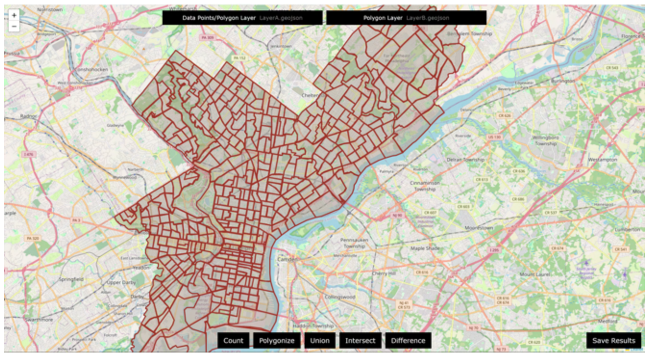
Data Points/Polygon Layer LayerA.geojson
Polygon Layer LayerB.geojson
Count
Polygonize
Union
Intersect
Difference
Save Results

Figure 4: Philadelphia Census Tracts Change

module) performs the group-by and returns the region labels associated with their counts. These countries' boundaries are visualized on the interactive map colored based on their tweet count. These boundaries include data counts. The users can interact with the visualization to view the count attribute values by clicking on the country's polygon. The resulting map (country and its count) can be saved by clicking the *Save Results* button.

3.2 Scenario 2: Neighborhood Blocks Extraction

This scenario, depicted in Figure 3, demonstrates the usage of *Pyneapple-G* in a polygonization example. Assume a user is interested in generating all neighborhood blocks in California, using its road network as an input, meaning generating all polygons created by the road network. The user will choose the GeoJSON file for the input road network for the first polygon layer. Upon pressing the *Polygonize* button, the query is sent to the backend, and the corresponding module (in this case, the DDCEL module) performs the polygonization and returns the new polygon layer. On the interactive map, users can see the input road network (displayed in black) and the resultant polygons (displayed in random colors). The result polygons can be saved into a GeoJSON file by clicking the *Save Result* Button.

3.3 Scenario 3: Hot-Spot Exploration

This scenario demonstrates the usage of *Pyneapple-G* in integrating polygonization and count-aggregate queries. Assume the user is interested in exploring hot spots in California based on current social media posts. The user has the California road network and geo-tagged data traces available. Similar to the Scenario in 3.2, *Pyneapple-G* extracts all California neighborhood blocks. Feeding this output polygon set as an input to the count-aggregate query along with the geo-tagged data similar to the Scenario in 3.1 produces a heat map that helps the user identify the hot spots.

3.4 Scenario 4: Census Tracts Overlay

This scenario, depicted in Figure 4, demonstrates the usage of *Pyneapple-G* in a map overlay example. Assume a user is interested in exploring the change in the same district over time. The user chose the GeoJSON files for Census Tracts in Philadelphia in the years 2000 and 2010. Upon pressing the *Intersect* button, the query is sent to the backend, and the corresponding module (in this case, the DDCEL module) performs the overlay and returns the intersection between the two input layers as a new polygon layer. The result intersection is displayed on the interactive map. Similarly, the user can find the union or the difference between these two layers. The user can also save the newly created layer using the *Save Result* Button.

3.5 Scenario 5: Building Applications

For social scientists working primarily in the Python environment, *Pyneapple-G* encapsulates all low-level Java implementations as black boxes and exposes just the essential functions as plug-and-use modules. To demonstrate the simplicity of *Pyneapple-G* Python APIs, we design Jupyter Notebooks for each of the modules. To get started, we show that social scientists can install the *Pyneapple-G* Python package through pip. All dependencies, including the JDK and the configuration of the Java environment, are configured by the script, and the users are unaware of the Java-Python bridge. We adopt familiar, classic, or popular use cases to demonstrate the usage of *Pyneapple-G* Python APIs. Through this demonstration, we show users that their work is painlessly improved with *Pyneapple-G* APIs while gaining huge benefits in scalability and expressiveness.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation, USA, under grants IIS-2237348 and CNS-2031418, the Google-CAHSI research grant, and Microsoft unrestricted gift.

REFERENCES

[1] Laila Abdelhafeez, Amr Magdy, and Vassilis J Tsotras. Scalable Spatial GroupBy Aggregations Over Complex Polygons. In *SIGSPATIAL*, 2020.
[2] Laila Abdelhafeez, Amr Magdy, and Vassilis J Tsotras. DDCEL: Efficient Distributed Doubly Connected Edge List for Large Spatial Networks. In *MDM*, 2023.
[3] Laila Abdelhafeez, Amr Magdy, and Vassilis J Tsotras. SGPAC: Generalized Scalable Spatial GroupBy Aggregations over Complex Polygons. *GeoInformatica*, 2023.
[4] Abdulaziz Almaslukh, Amr Magdy, and Sergio J. Rey. Spatio-temporal Analysis of Meta-data Semantics of Market Shares Over Large Public Geosocial Media Data. *Journal of Location Based Services*, 2018.
[5] Andres Calderon-Romero, Vassilis J Tsotras, and Amr Magdy. Scalable Overlay Operations over DCEL Polygon Layers. In *SSTD*, 2023.
[6] Yunfan Kang, Yongyi Liu, Hussah Alrashid, Akash Bilgi, Siddhant Purohit, Ahmed Mahmood, Sergio Rey, and Amr Magdy. Pyneapple-R: Scalable and Expressive Spatial Regionalization. In *ICDE*, 2024.
[7] Yongyi Liu, Ahmed Mahmood, Amr Magdy, and Sergio J. Rey. PRUC : P-regions with user-defined constraint. *VLDB Endowment*, 15(3):491–503, 2021.
[8] MagdyLab. Pyneapple-r. https://github.com/MagdyLab/Pyneapple.