THE UNIVERSITY OF SYDNEY

COMP5338 Advanced Data Models
Performance Comparison of MongoDB & PostGIS using Geospatial Data Model

Group: W16A Group 5

Kai Hirota (khir2693)
Ronald Pai (spai1131)
Ling Nga Meric Tong (lton2609)

1. Introduction

The objective of this study is to compare PostGIS and MongoDB with respect to the geospatial data model. PostGIS refers to PostgreSQL with geospatial support enabled by installing an extension. In particular, we are interested in the extent to which the two database systems support various use cases for geospatial data, and the differences in performances. By comparing the two, we seek to test the hypothesis: Since PostGIS is built specifically for geospatial data, it exceeds MongoDB in performance when using geospatial data.

Geospatial data refers to data with embedded geographical information and can be roughly divided into two types: raster data and vector data. Raster data defines space as a matrix or array of equal-sized cells, where each cell contains attributes about that particular location. Raster data is often used for continuous data that doesn’t require geometric shapes, such as rainfall volume, elevation, temperature, soil type, population density. On the other hand, vector data consists of a collection of geometries such as points, lines (LineString), polygons, as well as more complex data such as MultiPolygons. Each geometry consists of one or more points of latitude and longitude and attributes describing that particular geometry. For example, vector data includes data like roads, buildings, waterways, land-use, freeways, or boundaries. Both types of geospatial data also contain geographic metadata that describes the coordinate reference system used, as all coordinates are dependent on which coordinate reference system is used. In this study, we focus on vector data. When we refer to geospatial data in the context of this study, we are referring specifically to vector data.

In the three sections following the Datasets section below, we will outline and detail the comparison of one feature per section. In the first feature section, we cover geospatial features that deal primarily with points, such as proximity to a specific geographic point. In the second feature section, we cover features that deal with polygons, such as checking whether geometries intersect or contain one another. Due to the limitation of geospatial support in MongoDB, in the third section, we examine the DE-9IM feature which is available in PostGIS but not MongoDB.

Test Environment

- System Version: macOS 10.15.7 (19H2)
- Number of Processors: 1
- Intel(R) Core(TM) i5-8279U CPU @ 2.40GHz
- Total Number of Cores: 4
- Memory: 16 GB
- PostgreSQL Version: 12.3
- PostGIS Version: 3.0.2
- MongoDB Version: 4.0.6

Datasets

In order to compare the performance of PostGIS and MongoDB objectively, we used open-source datasets to create sample datasets on which test queries will be run on. The two primary datasets are Sydney Airbnb Open Data, which includes longitudes and latitudes of Airbnb listings in Sydney, and OpenStreetMap (OSM) Australia, which includes points, lines, and polygons used to create a map of Australia. In particular, we will be using the road dataset within OSM. Queries on the Airbnb dataset and OSM road dataset will use the same polygon as a constant. This polygon is from the Australia Local Government Authority Border Map, which includes polygons of local government jurisdiction such as Council of the City of Sydney, and Inner West City Council. The last dataset, NSW Neighbourhood polygons, was used to attach neighborhood data to the roads dataset for additional context and checking the correctness of queries. All of the data were converted to the same coordinate system, EPSG:4326, which is used by the global positioning system (GPS). Geopandas and shapely libraries were primarily used for manipulating and creating geospatial data, and PyMongo was used to interact with MongoDB, while SQLAlchemy and Geoalchemy libraries were used to interact with PostGIS. We also enabled spatial index in both PostGIS and MongoDB.

1. Sydney Airbnb Open Data
    a. https://www.kaggle.com/tylerx/sydney-airbnb-open-data
2. OpenStreetMap Australia
    a. https://download.geofabrik.de/australia-oceania.html
3. Australia Local Government Authority Border (e.g Shire/Council) Map
    a. https://www.igismap.com/australia-shapefile-download
4. NSW Neighbourhood Polygons
    a. https://data.gov.au/dataset/ds-dga-91e70237-d9d1-4719-a82f-e71b811154c6/distribution/dist-dga-5e295412-357c-49a2-98d5-6caf099c2339/details

2. Geospatial Functions: NearSphere vs. ST_DWithin

In our first section, it is aimed to compare the performance between MongoDB and PostGIS with respect to spherical spatial functions. In MongoDB, having the functions *nearSphere* combined with the *max distance* parameters, the query will return all the geometries at a certain distance sorted by distance. On the other hand, *ST_DWithin* has been used as an alternative in PostGIS in order to test the performance in both databases. In this spherical spatial function experiment, the first is used to test the time of performance of queries in the PostGIS, while the second is to test the performance of MongoDB databases.

For executing this experiment, it is mainly to check whether any points are located within a given distance from selected coordinates. The selected point coordinate (longitude, latitude) has been set as Sydney Bondi Beach coordinates (151.2767, -33.8915) respectively. The Airbnb listings dataset is extracted from Sydney Airbnb Open Data from Kaggle. It contains all the Airbnb listings in Sydney, NSW. Then, we created subsets of the listings dataset that contains 100 data points, 1000 data points, 10000 data points; in addition, 1 million data points are created by duplicating the original dataset. With the selected coordinates as the centre point, we wrote the SQL and MongoDB query that retrieves all the Airbnb listings within the given distance from the selected centre point in five different data sizes.

For the evaluation performance, each query was executed ten times on each dataset from the smallest (100 data points) to the largest (1 million data points) of Airbnb listings with four different set up on maximum distance:

- listings within a maximum distance of 5 km from the centroid
- listings within a maximum distance of 10 km from the centroid
- listings within a maximum distance of 20 km from the centroid
- listings within a maximum distance of 30 km from the centroid

Besides repeating each query as ten times, we also measured how long each query took in seconds with calculated the mean and standard deviation of the runtime as the performance comparison.

Sample Query (PostGIS vs. MongoDB)

PostGIS ST_DWithin() with a maximum distance of 5km in 100 data points from using Python:

```
sql='''
SELECT *
FROM airbnb_sydney_100
WHERE ST_DWithin(airbnb_sydney_100.geometry,
ST_SetSRID(ST_MakePoint(151.2767,-33.8915),4326),0.05)'''

result =gpd.read_postgis(sql, engine, geom_col='geometry')
results = [item for item in result]
print(result.shape)
```

MongoDB $nearSphere + $maxdistance from using Pymongo:

```
from bson.son import SON # required for $maxDistance
def distance_to_radians(km):
    return km/6378.137

#bondi beach coordinates
lat = -33.8915
lon = 151.2767
point= [lon, lat]

km=5
result= db.airbnb_sydney_100.find(
    {
        "geometry": SON([
            ("$nearSphere", point),
            ("$maxDistance", distance_to_radians(km))
        ])
    }
)
results= [item for item in result]
print(len(results))
```

Table 2.1. Comparison of the mean and standard deviation of the query runtime.

	MongoDB		postGIS	
max distance (km)/Data Size	mean time [second]	standard deviation	mean time [second]	standard deviation
5km				
100	0.007187605	0.002276653	0.043274999	0.006481027
1k	0.025807548	0.016310987	0.065703702	0.022447455
10k	0.141123176	0.075665804	1.241498327	0.325197246
100k	1.089699531	0.856567986	1.241498327	0.325197246
1m	34.98340652	31.24455561	74.07140317	25.09023428
10km				
100	0.008389878	0.005950478	0.043648839	0.00233995
1k	0.035507584	0.010341971	0.081143594	0.027774079
10k	0.241815591	0.029083681	2.782282567	0.201445733
100k	2.787020278	0.520455389	2.782282567	0.201445733
1m	317.8399241	272.6782824	75.91673875	8.069031153
20km				
100	0.010199976	0.002533084	0.048395753	0.007962633
1k	0.040365314	0.017878762	0.082090163	0.015813456
10k	0.301468277	0.03964041	3.650822353	0.30310949
100k	4.641707802	0.674566761	3.650822353	0.30310949
1m	914.9281501	353.0617526	107.6204711	7.79144199
30km				
100	0.008744836	0.002221548	0.045145726	0.002509106
1k	0.03627193	0.004936658	0.103290701	0.031626384
10k	0.475698066	0.444679669	4.069888043	0.407288217
100k	6.599097824	1.361575328	4.069888043	0.407288217
1m	1149.652263	677.2832733	341.3338144	719.1305923

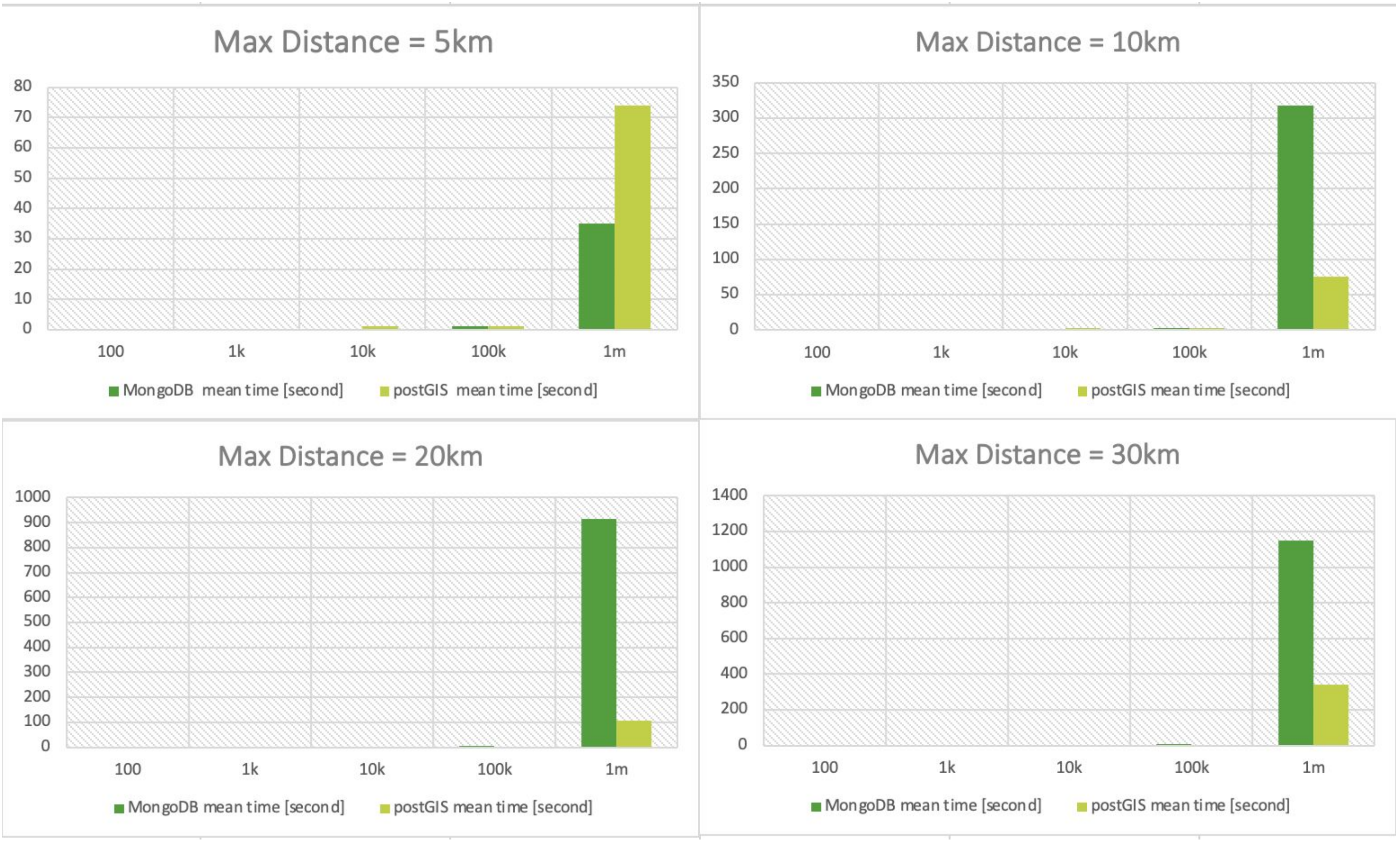


Figure 2.1. Runtime comparison between the two databases expressed in n km maximum distance.

Results

After comparing spherical spatial functions in both databases, there are some interesting findings. MongoDB definitely proves advantageous in time performance, when executing the query of the radius of 5, 10, 20, 30 kilometers in 100, 1,000, 10,000 data sizes. From the given chart above, it is shown clearly that the query time in MongoDB operates much faster with lower maximum distance and smaller data points. However, when taking the account of testing with larger data points, it shows PostGIS is ahead in executing the geospatial query, especially in 100,000 and 1 million data sizes. This observation presents a conclusion regarding our hypothesis that PostGIS is a better geospatial database for larger amounts of data. The standard deviation of running 1 million data points in MongoDB is much higher than running in PostGIS. Therefore, to sum up, with a larger radius of a circle (and therefore larger data size) to proximity to a specific geographic point, MongoDB begins to clearly slow down by comparison with PostGIS.

3. Geospatial functions: Intersects & Within

In this section, we compare the performance of PostGIS and MongoDB with respect to two frequently used geospatial filtering functions, "within" and "intersects". As the name suggests, "within(A.geometry, b.geometry)" returns data points from table or collection "A" that are inside the polygon "b," while "intersects(A.geometry, b.geometry)" returns data from "A" that intersect geometry "b" in any way. For "within" operation, we wrote a SQL and MongoDB query that retrieves all Airbnb listings located inside a polygon specified in the query. For "intersects," we wrote a query that retrieves all roads that intersect with a polygon, also specified in the query. The polygon was held constant for both queries, and we used the Council of the City of Sydney jurisdiction polygon from the "Australia Local Government Authority Border (e.g Shire/Council) Map" dataset. Each of the two queries ran on subsets of the Airbnb and the OSM road dataset of various sizes. For example, for the road dataset, we first filtered out all the roads that are not in New South Wales, since the polygon used for filtering is Sydney. Then, we created subsets of the road dataset that contain 100 data points, 1,000 data points, 10,000 data points, 100,000 data points, and 500,000 data points. For the Airbnb dataset, in addition to 100, 1,000, 10,000, and 100,000, a dataset containing 1 million data points was created by duplicating the original data and modifying the coordinates, before concatenating with the original data - instead of 500,000 data points like the road dataset.

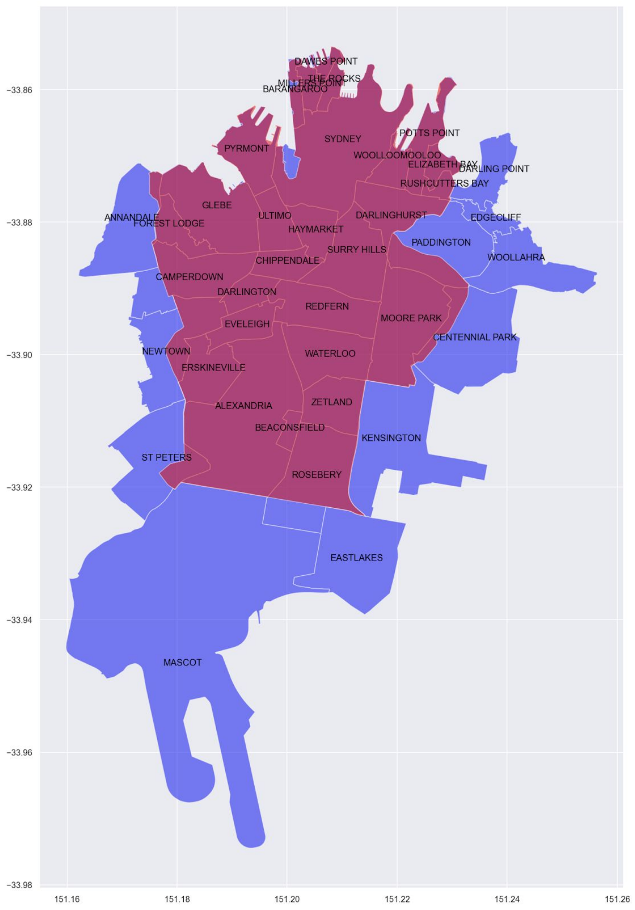


Figure 3.1. Red area is the Council of the City of Sydney polygon used as the filtering polygon. Note that the shape is not a perfect union of overlapping neighborhoods.

For performance evaluation, each query was executed 50 times on each dataset from the smallest (100 data points) to the largest (500,000 data points for road, 1 million data points for Airbnb). We measured how long each query took in milliseconds and calculated the mean runtime, as well as the standard deviation.

Intersects

Sample query: PostGIS ST_Intersects() where district_polygon is a Polygon object from shapely library for Python

```
sql = f"""
SELECT *
FROM roads_sydney_500k as roads
WHERE ST_Intersects(roads.geometry, ST_GeometryFromText('{district_polygon.wkt}', 4326));
"""
start = time.time()
df = gpd.read_postgis(sql, engine, geom_col='geometry')
end = time.time()
time_in_ms = (end - start) * 1000
```

Sample query: MongoDB $geoIntersects where js is a geojson.feature.FeatureCollection object

```
cursor = db['roads_sydney_500k'].find(
    {
        'geometry': {
            '$geoIntersects': {'$geometry': js['features'][0]['geometry']}
        }
    }
)
print(len([item for item in cursor]), 'documents')
stats = cursor.explain()
time_in_ms = stats['executionStats']['executionTimeMillis']
```

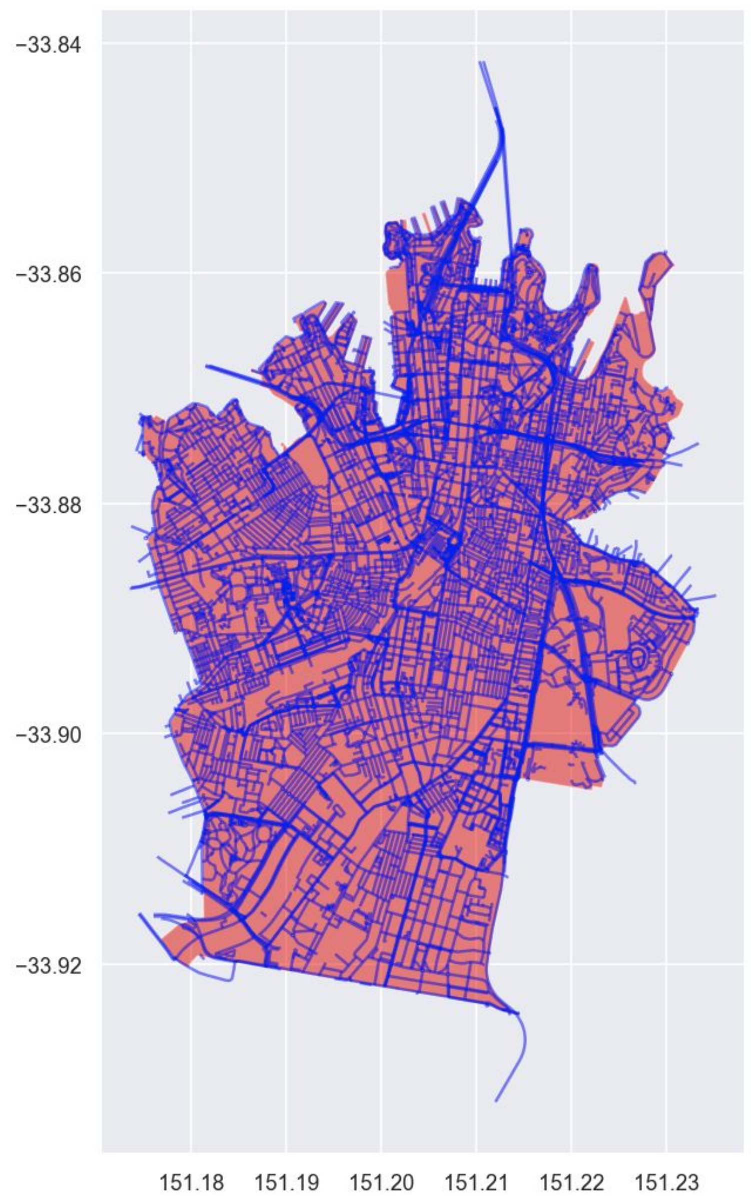


Figure 3.2. Result of filtering roads by an intersection with polygon.

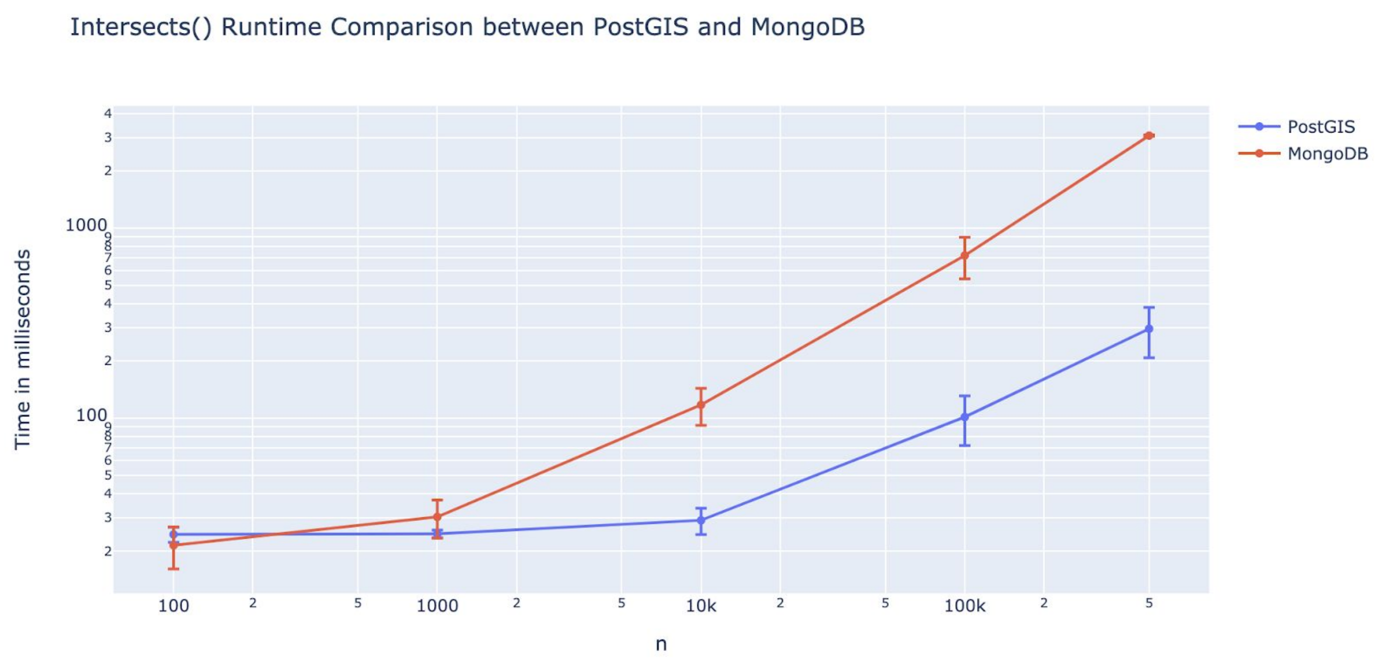
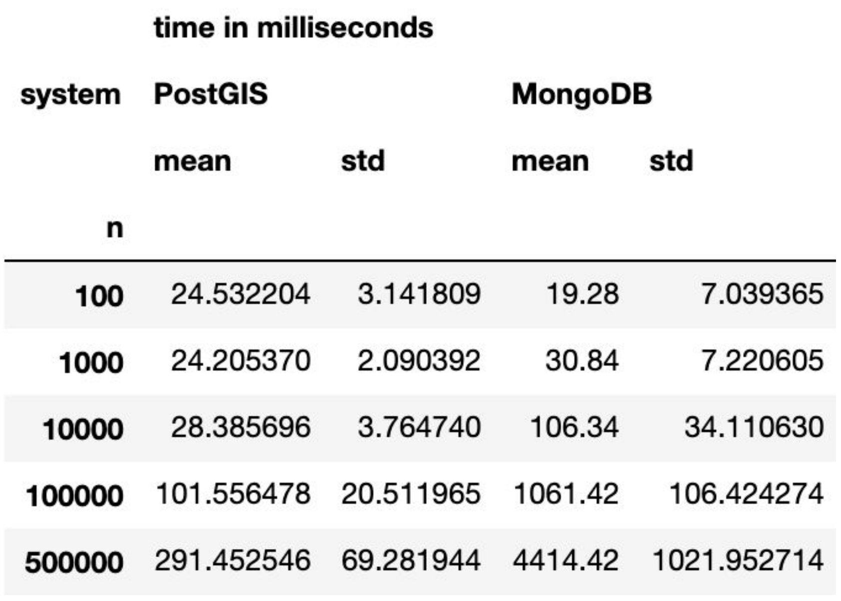


Figure 3.3 : Runtime comparison between the two databases expressed in logarithmic scale.



	time in milliseconds			
system	PostGIS		MongoDB	
	mean	std	mean	std
n				
100	24.532204	3.141809	19.28	7.039365
1000	24.205370	2.090392	30.84	7.220605
10000	28.385696	3.764740	106.34	34.110630
100000	101.556478	20.511965	1061.42	106.424274
500000	291.452546	69.281944	4414.42	1021.952714

Figure 3.4 : Execution summary statistics of PostGIS and MongoDB on Intersection query.

Within

Sample query: PostGIS ST_Within() where district_polygon is a Polygon object from shapely library for Python

```
sql = f"""
SELECT *
FROM airbnb
WHERE ST_Within(airbnb.geometry, ST_GeometryFromText('{district_polygon.wkt}', 4326));
"""
start = time.time()
df = gpd.read_postgis(sql, engine, geom_col='geometry')
end = time.time()
time_in_ms = (end - start) * 1000
```

Sample query: MongoDB $geoWithin where js is a geojson.feature.FeatureCollection object

```
cursor = db['airbnb_sydney_1m'].find(
    {
        'geometry': {
            '$geoWithin': {'$geometry': js['features'][0]['geometry']}
        }
    }
)
print(len([item for item in cursor]), 'documents')
stats = cursor.explain()
time_in_ms = stats['executionStats']['executionTimeMillis']
```

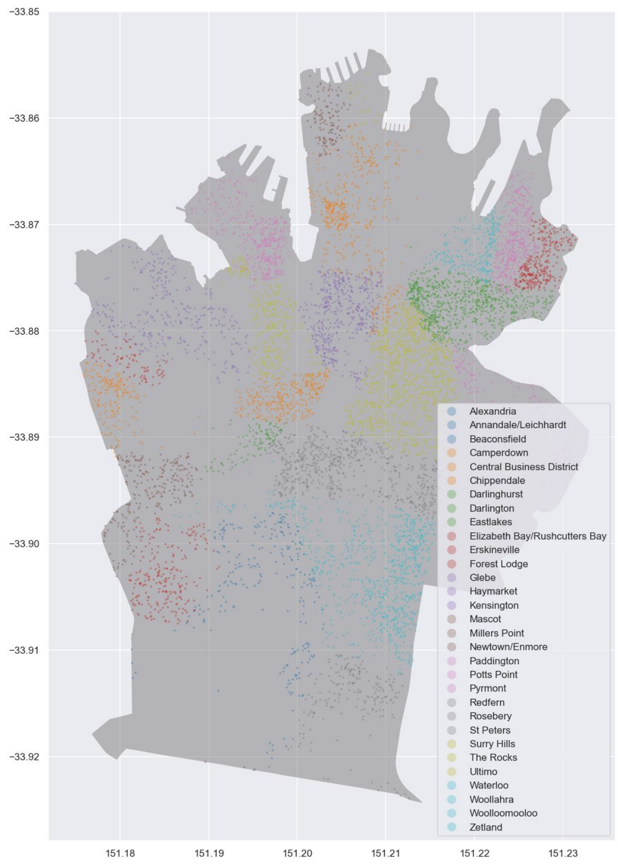


Figure 3.5 : Result of filtering Airbnb listings by “within” Sydney polygon.

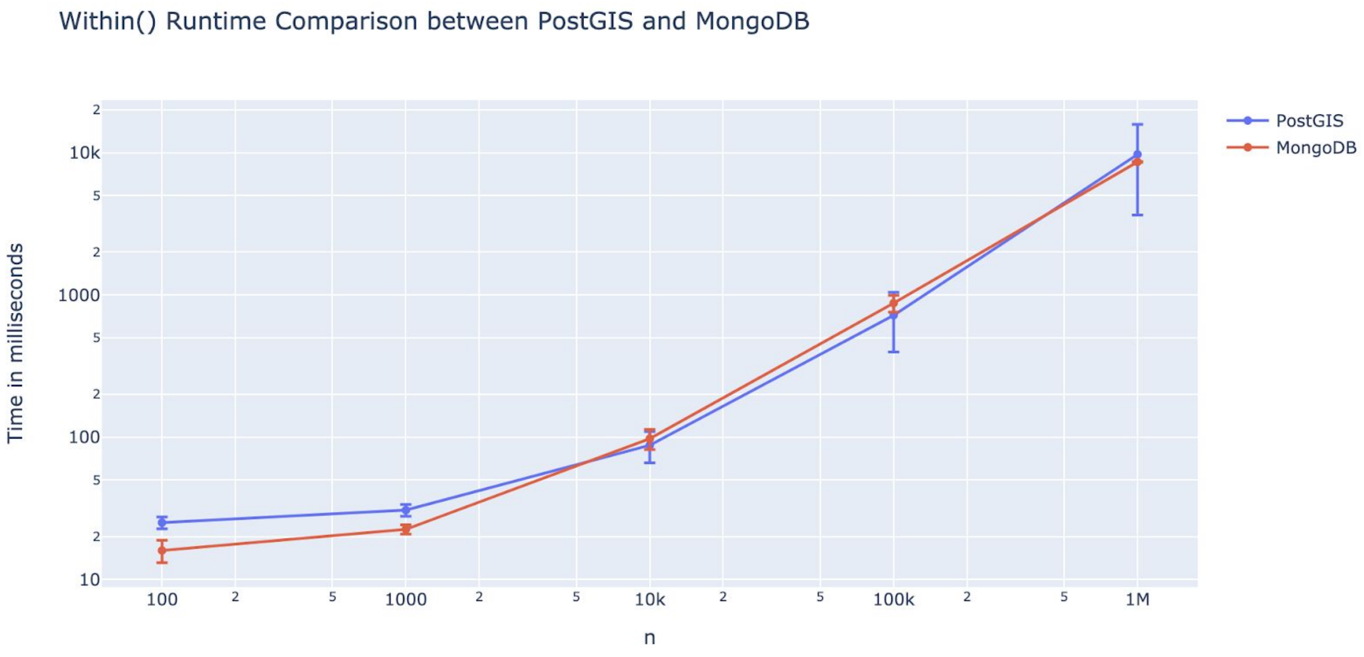
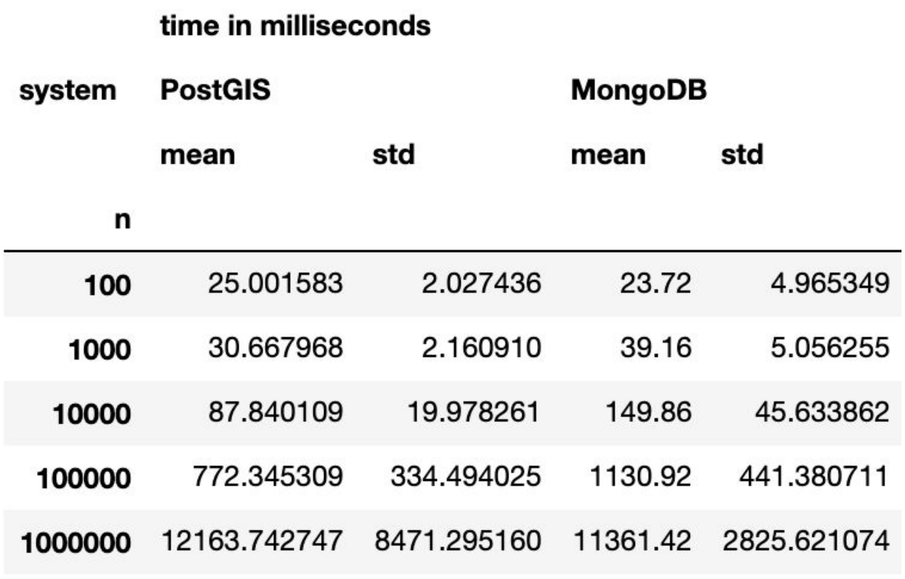


Figure 3.6 : Runtime comparison between the two databases expressed in logarithmic scale.



	time in milliseconds			
system	PostGIS		MongoDB	
	mean	std	mean	std
n				
100	25.001583	2.027436	23.72	4.965349
1000	30.667968	2.160910	39.16	5.056255
10000	87.840109	19.978261	149.86	45.633862
100000	772.345309	334.494025	1130.92	441.380711
1000000	12163.742747	8471.295160	11361.42	2825.621074

Image 3.7 : Execution summary statistics of PostGIS and MongoDB on Intersection query.

Results

For “intersects” operation, MongoDB executed the query faster than PostGIS on smaller datasets, but the runtime increased at a higher rate than PostGIS as the size of the data increased. PostGIS seems to handle geospatial queries on larger datasets much better than MongoDB. Furthermore, when querying large datasets, MongoDB had much higher fluctuations in runtime, while PostGIS remained consistent with lower standard deviations.

For “within” operation, both PostGIS and MongoDB had similar runtimes, but at 1 million data points, MongoDB seemed to execute queries and return results slightly faster and with more consistency. One key difference was that at 1 million data points, the standard deviation of runtime for PostGIS was order of magnitude higher than at 100,000 data points - while the data size grew by 10 times, standard deviation increased more than 20 times, and the mean runtime also increased to nearly 20 times. On the other hand, MongoDB’s increase in the mean runtime was proportional to the increase of dataset size, while the standard deviation of runtime increased much less, from 441.38 milliseconds to 2825.62 milliseconds.

4. Dimensionally Extended 9 Intersection Model

PostGIS has an extensive framework that describes how two spatial objects interact, called the Dimensionally Extended 9-Intersection Model (DE-9IM). It uses a matrix (Table 4.1) that compares the relationship between the features of two spatial objects which are interior, boundary and exterior, forming 9 possible intersections.

Table 4.1. The Dimensionally Extended 9-Intersection Model (DE-9IM) matrix (PostGIS, 2020).

	Interior	Boundary	Exterior
Interior	$dim(I(a) \cap I(b))$	$dim(I(a) \cap B(b))$	$dim(I(a) \cap E(b))$
Boundary	$dim(B(a) \cap I(b))$	$dim(B(a) \cap B(b))$	$dim(B(a) \cap E(b))$
Exterior	$dim(E(a) \cap I(b))$	$dim(E(a) \cap B(b))$	$dim(E(a) \cap E(b))$

Figure 4.1 shows an example of a DE-9IM matrix for two intersecting polygons. The number in each cell refers to the dimension of the intersecting features (or "F" for no intersections). For example, in the upper left cell, because a 2D region of the interiors is formed for the intersection, then the value is 2 (which corresponds to the dimension).

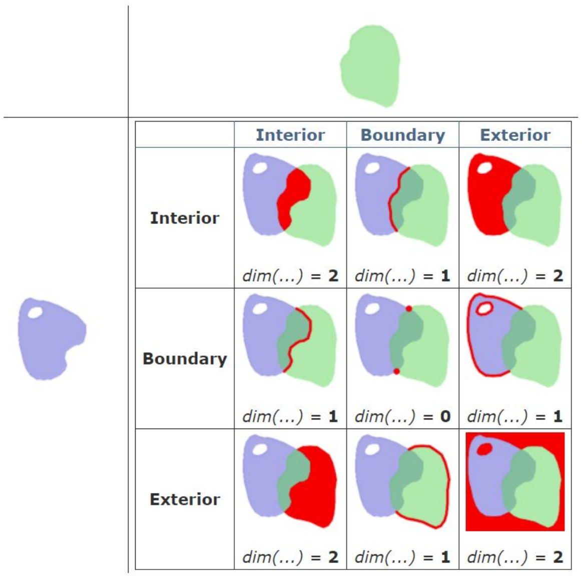


Figure 4.2. An example of a DE-9IM matrix for two intersecting polygons (PostGIS, 2020).

With the DE-9IM representation, spatial objects can be queried by how they are specifically related to each other. A prominent example that demonstrates this is a data model containing lakes and docks (PostGIS, 2020), where lakes are polygons and docks are lines, and are placed randomly on a 2D-plane. A query can be generated to validate docks within lakes by first identifying the physical conditions and then the corresponding DE-9IM representation:

1. The docks are attached to the boundary of the lake:
    a. The dock's boundary point intersects with the lake's boundary.
2. The entire dock is within the lake:
    a. The dock's interior linearly intersects the lake's interior.
    b. The dock's boundary point intersects with the lake's interior.
    c. The dock's interior does not intersect with the lake's exterior.

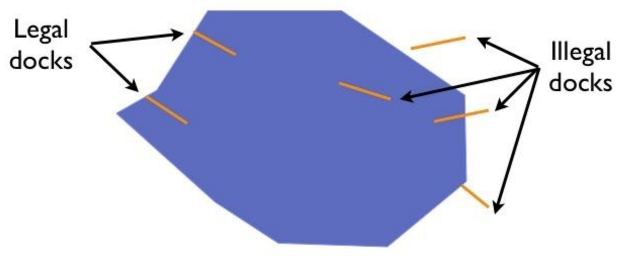
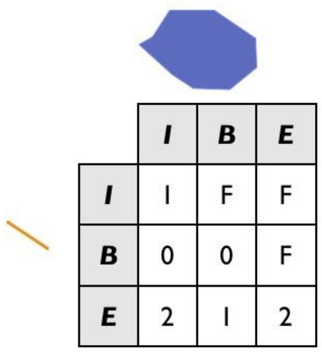


Figure 4.3. A data model containing a lake and docks (PostGIS, 2020).

In this case, the conditions are represented by the following DE-9IM matrix.



	I	***B***	***E***
I	1	F	F
B	0	0	F
E	2	1	2

Figure 4.4. DE-9IM matrix for lake and docks data model (PostGIS, 2020).

Therefore to find all the valid docks, it makes uses of **ST_Relate()** in the following query.

```
SELECT docks.*
FROM docks JOIN lakes ON ST_Intersects(docks.geom, lakes.geom)
WHERE ST_Relate(docks.geom, lakes.geom, '1FF00F212');
```

(PostGIS, 2020)

Experiment

A problem that makes use of DE-9IM is finding specific types of intersecting roads (or streets, but for simplicity, they will be referred to as roads). Cross intersections and T intersections (labelled “c” and “a” respectively in Figure 4.5) are of particular interest.

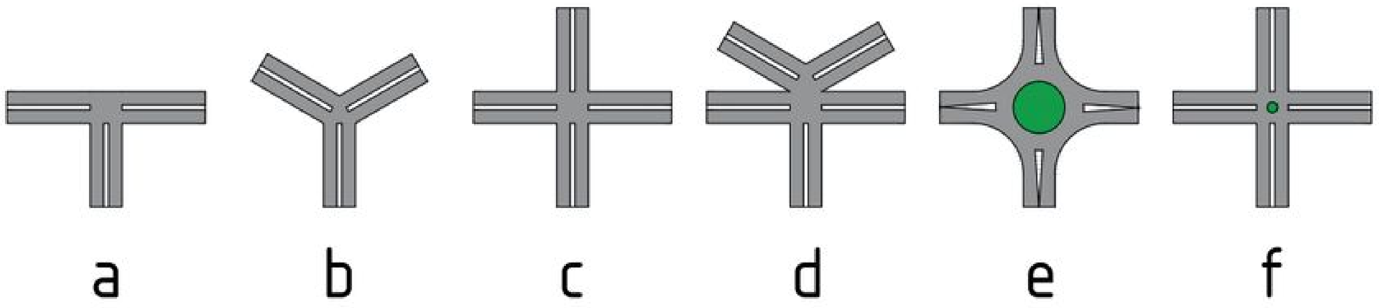


Figure 4.5. Types of intersecting roads (Khrapova Bc M, 2020).

In the OpenStreetMap dataset, roads are line objects. This means they have boundary points (boundaries) which define the ends of the roads, and the line connecting the boundary points forms the interior of the road. The space surrounding the road is the exterior.

Cross Intersection

A physical representation of a cross intersection is two roads intersecting somewhere along the roads. The ends of the roads do not intersect with any part of the other road at all. Parts of the road lie in the space surrounding the other road. Interpreting this for roads X and Y with DE-9IM gives:

- The interiors intersect at a point.
  dim(I(X)) n dim(I(Y)) = 0

- The boundaries of a line do not intersect with the other line’s boundaries or interior.
  dim(B(X)) n dim(B(Y)) = dim(B(X)) n dim(I(Y)) = dim(I(X)) n dim(B(Y)) = F

- The boundaries of a line intersects with the other line’s exterior at a point.
  dim(B(X)) n dim(E(Y)) = dim(E(X)) n dim(B(Y)) = 0

- The interior of a line intersects with the other line’s exterior.
  dim(I(X)) n dim(E(Y)) = dim(E(X)) n dim(I(Y)) = 1

- The exteriors intersect in a 2D region.
  dim(E(X)) n dim(E(Y)) = 2

Table 4.2. DE-9IM for a cross intersection.

		Road Y X------------------X		
		Interior	**Boundary**	**Exterior**
Road X X | | X	**Interior**	0	F	1
	Boundary	F	F	0
	Exterior	1	0	2

T Intersection

Similar for a T Intersection, the physical representation is that one end of the road must lie somewhere anong another road. The other ends of the roads do not intersect each other nor along the roads. Interpreting this for roads X and Y with DE-9IM gives:

- The interiors do not intersect.
  dim(I(X)) n dim(I(Y)) = F
- The boundaries of a line intersect with the other line’s interior at a point.
  dim(B(X)) n dim(I(Y)) = dim(I(X)) n dim(B(Y)) = 0
- The boundaries do not intersect.
  dim(B(X)) n dim(B(Y)) = F
- The interior of a line intersects with the other line’s exterior.
  dim(I(X)) n dim(E(Y)) = dim(E(X)) n dim(I(Y)) = 1
- The boundaries of a line intersects with the other line’s exterior at a point.
  dim(B(X)) n dim(E(Y)) = dim(E(X)) n dim(B(Y)) = 0
- The exteriors intersect in a 2D region.
  dim(E(X)) n dim(E(Y)) = 2

Table 4.3. DE-9IM for a T intersection.

		Road Y X-------------------X		
		Interior	**Boundary**	**Exterior**
Road X X | | X	**Interior**	F	0	1
	Boundary	0	F	0
	Exterior	1	0	2

The queries for cross intersection and T intersection were constructed to be the following:

```sql
SELECT r1.geometry FROM roads_sub AS r1
JOIN roads_sub AS r2 ON ST_Intersects(r1.geometry, r2.geometry)
WHERE ST_Relate(r1.geometry, r2.geometry, '0F1FF0102');

SELECT r1.geometry FROM roads_sub AS r1
JOIN roads_sub AS r2 ON ST_Intersects(r1.geometry, r2.geometry)
WHERE ST_Relate(r1.geometry, r2.geometry, 'F010F0102');
```

Results

PostGIS: Cross Intersection

The cross intersections in the suburb of Chippendale were plotted in Figure 4.6. It was observed that while the highlighted cross intersections were correct, some cross intersections were not highlighted.

By inspecting the data, it was realised that some roads had multiple rows of data. In other words, some roads were composed of multiple segments instead of being one continuous road in one record. Table 4.4 shows an example of Cleveland St being made up of several records (5 shown out of 18). This may be due to each segment of the road representing a different feature eg. different speed limit, number of directions of travel, fork in the road, roundabout, or created out of convenience.

As such, these roads technically do not form a cross intersection and could simply be made of 4 line segments intersecting at a point, and therefore not highlighted as a cross intersection in the query.

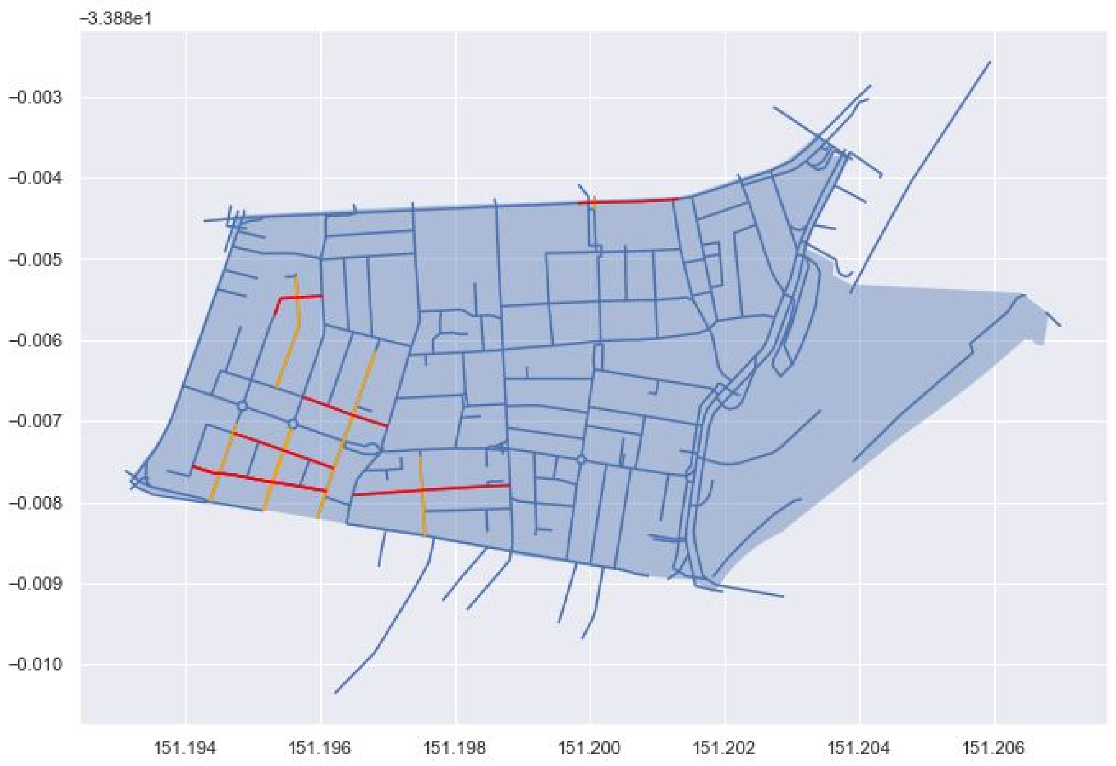


Figure 4.6. A plot of cross intersections in Chippendale from the sample query.

Table 4.4. Example of one road composed of multiple segments.

	osm_id	code	fclass	name	ref	oneway	maxspeed	layer	bridge	tunnel	geometry
7	411192715	5113	primary	Cleveland Street	None	B	50	0	F	F	LINESTRING (151.19981 -33.88874, 151.20001 -33...
12	411017206	5113	primary	Cleveland Street	None	F	50	0	F	F	LINESTRING (151.19349 -33.88781, 151.19426 -33...
19	411192716	5113	primary	Cleveland Street	None	B	50	0	F	F	LINESTRING (151.20037 -33.88882, 151.20049 -33...
23	595889192	5113	primary	Cleveland Street	None	B	50	0	F	F	LINESTRING (151.19822 -33.88851, 151.19849 -33...
40	175969863	5113	primary	Cleveland Street	None	B	50	0	F	F	LINESTRING (151.19516 -33.88811, 151.19507 -33...

PostGIS: T Intersection

Similarly, the T intersections in the suburb of Chippendale were queried and plotted. The resulting plot in Figure 4.7 showed no T intersections were found. A wider search was performed instead on neighbouring suburbs and two were found in the suburb of Alexandria and plotted in Figure 4.8. This validated the sample query was correct and working, and that there were actually no T intersections in Chippendale from the data model's perspective.

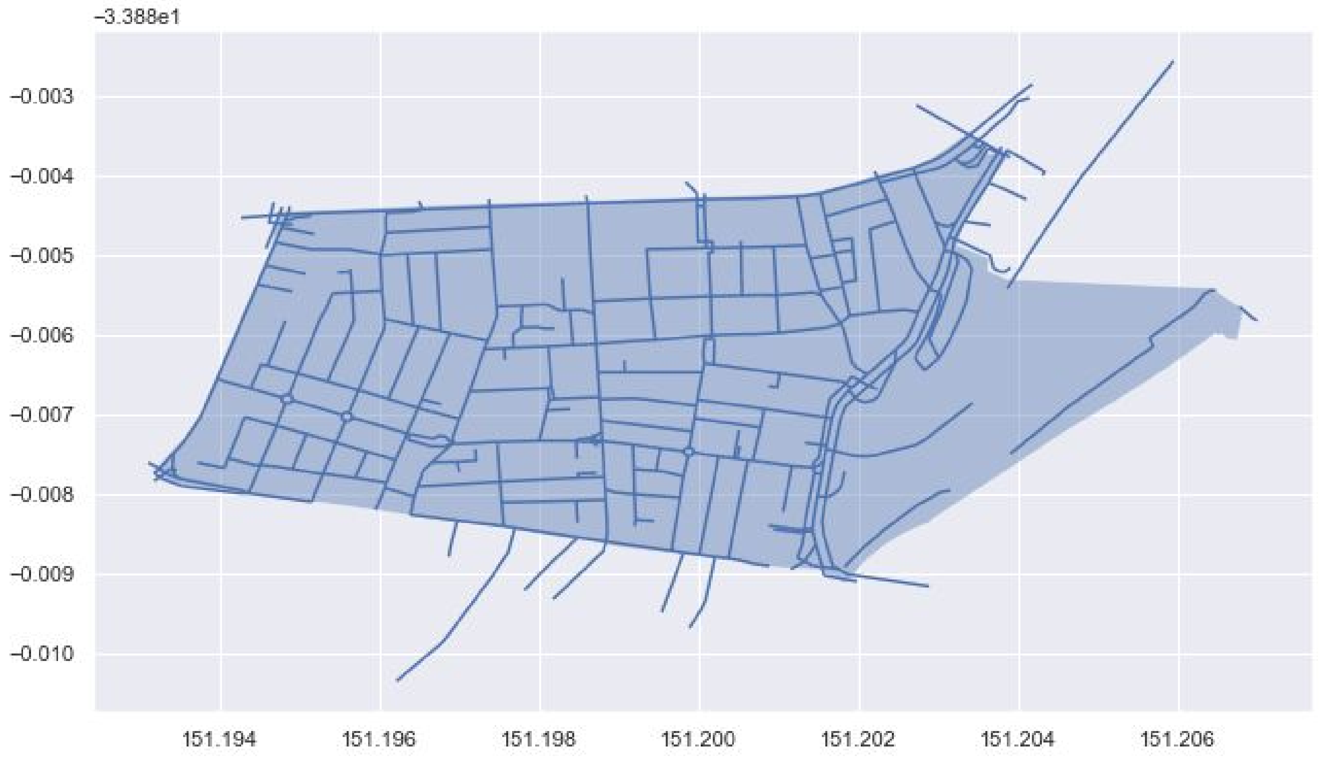


Figure 4.7. A plot of T intersections in Chippendale from the sample query (None).

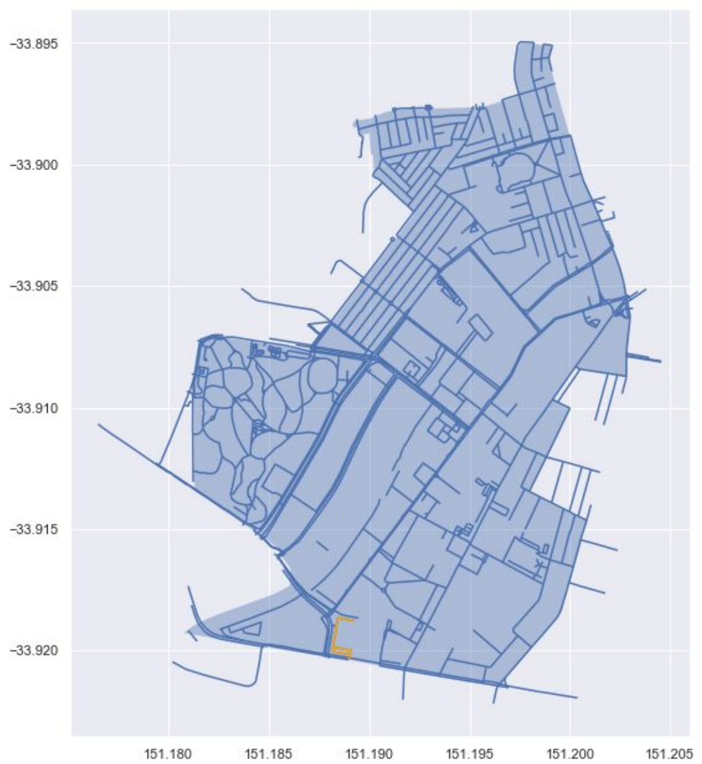


Figure 4.8. A plot of T intersections in Alexandria from the sample query.

Similar to the discovery in the previous subsection, because roads were composed of multiple records or segments, they may simply be 3 segments intersecting at a point, which was why none were showing in Chippendale and only a few were showing in Alexandria, even though T intersections exist by observation.

A possible solution to increasing the number of intersections in the queries that is outside the scope of this experiment, is to pre-process the data. The pre-processing would involve merging records of line objects by matching road names and boundary points, and concatenating line objects. The road segments would then form longer continuous roads, and therefore increasing the chance of road intersections. Irrelevant features such as maximum speed and directions of travel would be omitted as part of the merge. This could then be stored as a new table for separate querying.

MongoDB

Because the DE-9IM feature is not available in MongoDB, $geoIntersects would return all the road intersections similarly in Chapter 3, which cannot be projected or filtered to either cross intersections or T intersections.

Conclusion

Since the iPhone was first unveiled in 2007, smartphones have become first class citizens of our pockets, in both developed and emerging economies. In addition to being equipped with cameras, constant connectivity also leads to explosive growth in the volume of geospatial data generated on a daily basis. Many technology companies, such as Uber, Airbnb, Tesla, SpaceX, Google, and Apple heavily use geospatial data. Thus, extracting valuable insights and harnessing those insights to deliver better products and user experience has become an opportunity of significant importance. As such, the capability to analyze, store, manipulate, and transmit geospatial data better and faster while using less resources will lead to competitive advantage in the technology space, whether in academia or in industry. The objective of this study was to compare the performance of MongoDB and PostGIS with respect to geospatial data. Through experiments, we tested our hypothesis that PostGIS, being made specifically for geospatial data, should outperform MongoDB. Specifically, we compared the performance of spherical and polygon filter functions in sections 2 and 3; in section 4, we highlighted some of the features that PostGIS offers, that MongoDB lacks, such as the DE-9IM.

In conclusion, PostGIS is better than MongoDB when it comes to executing geospatial queries that involve large data, or complex geometries such as lines and polygons. Execution time for PostGIS increases order of magnitude slower than MongoDB, and tends to be more consistent with lower standard deviation in the time it takes to execute queries. However, if only dealing with points data, and it isn't necessary to use a large radius to filter for proximity to points, then MongoDB may suffice, or even outperform PostGIS under the right conditions. Nevertheless, PostGIS offers a far more comprehensive set of tools and functionalities for dealing with geospatial data. The DE-9IM, which is available in PostGIS but not MongoDB, allows further projection or filtering of intersections of spatial objects. However, its effectiveness is as good as the structure and definition of the data model. As proven in the experiment of this dataset, if a physical feature is modelled by multiple objects then DE-9IM may not return all the expected results.

Appendix

Github Repository

Contains Jupyter notebook used to create, transform, and load the test datasets:
https://github.com/from81/PostGIS_MongoDB_Geospatial_Data_Model_Comparison

Contributions

Kai Hirota (khir2693)

- Set up PostGIS and MongoDB and created a brief how-to guide for replicating the same environment.
- Researched and collected GIS datasets, and created test datasets of various sizes for both PostGIS and MongoDB by utilizing geospatial libraries and Python.
- Conducted performance comparison of Within and Intersects operation for both systems.
- Wrote the introduction of this report in addition to feature 2 (section 3 of this report).

Ronald Pai (spai1131)

- Researched possible features and literature review.
- Performed experiment and reported on DE-9IM.
- Formatted report, grammar and spellcheck.

Ling Nga Meric Tong (lton2609)

- Run the performance comparison of spherical operation for both systems.
- Wrote section 2 of this report.
- Wrote the conclusion.

References

1. https://www.researchgate.net/publication/332921357_The_Comparison_of_Processing_Efficiency_of_Spatial_Data_for_PostGIS_and_MongoDB_Databases
2. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0262-8
3. https://postgis.net/workshops/postgis-intro/de9im.html
4. https://postgis.net/docs/using_postgis_dbmanagement.html
5. https://www.enterprisedb.com/blog/postgres-outperforms-mongodb-and-ushers-new-developer-reality