K-NEAREST NEIGHBOUR QUERY PERFORMANCE ANALYSES ON A LARGE SCALE TAXI DATASET: POSTGRESQL vs. MONGODB

İ. B. Coşkun [1], S. Sertok [2], B. Anbaroğlu [1, *]

[1] Dept. of Geomatics Engineering, Hacettepe University, Turkey – (ihsan.coskun, banbar)@hacettepe.edu.tr
[2] Dept. of Statistics, Hacettepe University, Turkey – sibel.sertok@hacettepe.edu.tr

Commission IV, WG IV/4

KEY WORDS: spatial query, *k*NN, database, GIS, open-source

ABSTRACT:

The increasing volume of transport network data necessitates the use of a DataBase Management System (DBMS) to store, query and analyse data. There are two main types of DBMS: relational and non-relational. Many different DBMS are available on the market but only some of them could handle spatial data. Therefore, determining which DBMS to use for operational purposes is of interest to researchers and analysts working in spatial information science. One of the commonly used spatial queries in GIS is the *k*-Nearest Neighbour (*k*NN) of a given point. This paper analyses the performance of the *k*NN query in PostgreSQL and MongoDB, both being a representative of relational and NoSQL DBMS respectively. Two different metrics have been investigated to determine the performance: i) spatial accuracy and ii) run time. Haversine and Vincenty formulas are used to calculate the distance between the point and the determined neighbours, which are then used to determine the spatial accuracy of the DBMS. Sensitivity analysis have been carried out by varying the *k* value and the execution times are recorded. The experiments are carried out on New York City's openly available taxi dataset consisting of millions of taxi pickup and dropoff points. The results indicate that MongoDB outperforms Postgres both in terms of execution time and spatial accuracy regardless the value of *k*. In order to facilitate reproducibility of the results, the developed software is shared on GitHub.

1. INTRODUCTION

One of the commonly used spatial analysis methods on point datasets is the *k*-Nearest Neighbour (*k*NN) method. The method has initially been proposed for point classification (Cover and Hart 1967), but has been used for different purposes ranging from house rent price estimation to privacy preserving on spatiotemporal databases (Dritsas et al., 2018; Hu et al., 2019). The method relies on pairwise distance between points. The performance of the algorithm mainly depends on two parameters: the distance calculation method and the value of *k*. Considering that the amount of spatial point data generated on a daily basis is ever increasing, the need to rely on a DataBase Management System (DBMS) is apparent. However, as different DBMS could be used to store and query spatial data, it is important to investigate the *k*NN performance of different DBMS. In this way, researchers or companies would have a better-informed decision regarding the choice of the correct DBMS for their operational use.

There are mainly two types of DBMS, relational and non-relational. The traditional approach in GIS is to use relational DBMS, where data are stored in tables. Determining the attributes of tables and linking different tables with each other necessitates a data schema. The emergence of web-based systems makes it difficult to determine such a schema since modelled systems are dynamic, which led to the emergence of non-relational, or shortly NoSQL (Not Only SQL) databases. Consequently, they are well suited for today's need of modelling highly dynamic, usually web-based, systems (Veenendaal, Brovelli, and Li 2017). Considering that there are many different DBMS available on the market, this paper focused on a representative of each DBMS type: PostgreSQL (Postgres) with PostGIS extension and MongoDB to investigate the query performance of *k*NN on a large-scale, openly available dataset: taxi records of New York City (Donovan and Work 2014).

There are three main reasons behind relying on the aforementioned DBMSs. First, they both support spatial data types as well as provide spatial indexing mechanism to improve the performance of queries. Second, they are both open-source software. Third, they possess a large user base. The former two reasons indicate that technical issues could be resolved quickly.

The aim of this paper is to investigate the *k*NN query performance of Postgres and MongoDB in two different aspects. First, run times are investigated through a sensitivity analysis by varying the values of *k* for randomly chosen points. Second, spatial accuracy of different DBMS are compared with respect to Haversine and Vincenty formulas (Mahmoud and Akkari 2016). The results indicate the superiority of MongoDB compared to Postgres in both aspects. It detects the *k*NN of a query point instantly and the neighbours detected are more closer to the query point regardless the value of *k*. In order to ease reproducibility of the results as well as increase the collaborative efforts, the project is initiated as open-source and shared on GitHub (Coşkun, 2019).

The remainder of this paper is organised as follows. Section 2 provides the literature review regarding the use of *k*NN in spatial information science and the characteristics of different DBMS. Section 3 provides the methodology of the paper and the subsequent section provides the results on a large-scale taxi dataset. Finally, Section 5 concludes the paper by providing a discussion of the results and their implications for future studies.

* Corresponding author

2. LITERATURE REVIEW

The growth of data volume in all research areas led researchers and analysts to store their data in a DBMS. At the time of writing, 345 DBMSs are compared on db-engines.com. Therefore, the right choice of the DBMS for the operational purpose of a company or organisation is an important step to achieve their desired performance goal (Makris et al., 2019).

Researchers compared different DBMS for various tasks. For example, (Freire et al., 2016) compared the performance of XML databases, such as BaseX, eXistdb and Berkeley DB, with MySQL and Couchbase regarding population-based queries on a healthcare dataset. Having analysed parameters like disk storage, query response and indexing times, they suggest that Couchbase could be a better alternative amongst the alternatives. (Mehta et al., 2017) compared SciDB, Spark and Myria on two large image datasets belonging to astronomy and neuroscience. Their findings suggest the importance of parallelization of operations and memory management to eliminate out-of-memory errors. Pereira, Morais, and Freitas (2018) analysed the effectiveness of NoSQL DBMSs including Couchbase, MongoDB and RethinkDB in a single and multi-thread environment with a simple data model consisting of five attributes (i.e. id, number, date, customer and amount). The tests were carried out on a web-based platform on common operations such as post, patch, get and delete. Their findings suggest that Couchbase performs better in most scenarios regarding response times and server's throughput.

Spatial databases have the ability to index spatial data, support spatial data types such as points, lines and polygons and perform spatial queries. Therefore, researchers are also interested in assessing the performance of DBMSs regarding spatial queries. For instance, Schmid, Galicz, and Reinhardt (2015) compared the performance of Postgres and MongoDB regarding the point-in-polygon analysis. They found out that, as the size of the database increases, MongoDB performs better than Postgres. In another study, Matuszka and Kiss (2014) compared the relational DBMSs Postgres, MySQL and Oracle with non-relational DBMSs Jena and Sesame. Loading data and query time of point-in-polygon analysis are investigated. Results suggest that Oracle and Postgres outperform others in respective operations. Last, Agarwal and Rajan (2016) investigated the performance of Postgres and MongoDB regarding line intersection and point-in-polygon queries. Results suggest that MongoDB outperforms Postgres with an average factor of 25 and suggest that "NoSQL databases may be better stated for simultaneous multiple-user query systems including Web-GIS and mobile-GIS".

It should also be highlighted that even though some researchers overly mention the word 'spatial database', it is not clear which DBMS they relied on (Chen et al., 2010). In addition, not all of the common queries are investigated and there is lack of research evidence regarding the performance of detecting the *k*NN of a query point. Since *k*NN is applied in many different research areas, including indoor environments (Alamri,2018), researchers have investigated efficient ways to index spatial data to reduce *k*NN execution times (Zhong et al., 2013). The effectiveness of indexing spatial data regarding query execution times has already been demonstrated (Nguyen,2009). The existing research indicates the necessity to investigate the performance of spatial queries, and more specifically *k*NN on different DBMS.

3. METHODOLOGY

This paper analyses the query performance of *k*NN on two renowned DBMS: Postgres and MongoDB. Therefore, the first step of the methodology is to import the openly available taxi data set of New York City into Postgres. The main reason to first import the data to Postgres is to assure that each record has a unique ID, which is not readily available in raw data. Second, through a Python function, *postgres2GeoJSON*, the database is converted into chunks of GeoJSON files, which could then be imported into MongoDB. Third, Haversine and Vincenty distance functions are used to calculate the distance between two points to assess the spatial accuracy of the detected nearest neighbours (Veness, 2019). Finally, a sensitivity analysis is conducted by varying the value of *k* on randomly chosen taxi pickup locations. Methodology of the paper is summarised in Figure 1.

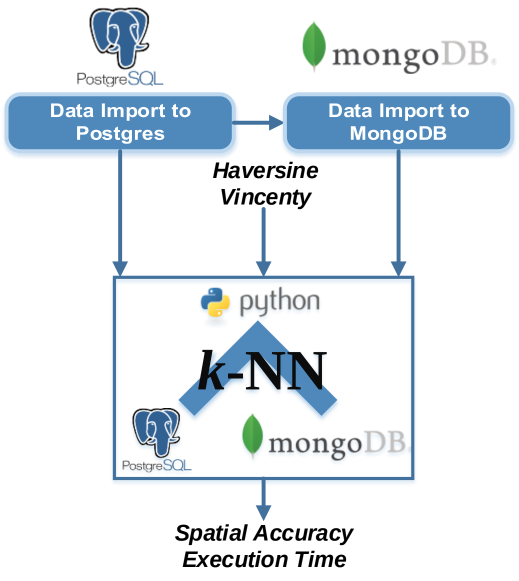
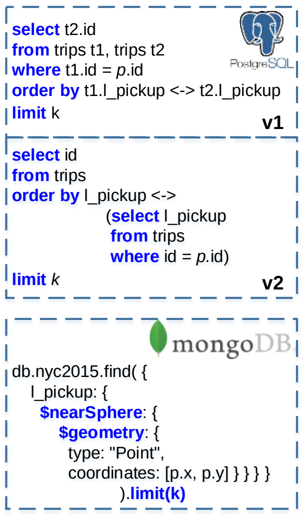


Figure 1 Performance analyses of Postgres and MongoDB regarding the *k*NN query

Indexing is used in databases to speed up query execution (Nguyen, 2009). There are different indexing methods, whose names and characteristics vary depending on the database and data type that the index is going to operate. Spatial indexes use for spatial data. One of the commonly used spatial indeces in Postgres is the Generalized Search Tree (GIST). On the other hand, the spatial index is mandatory to execute a spatial query in MongoDB, and '2Dsphere' spatial index method is used because it calculates geometries on a sphere instead of a two-dimensional plane. Consequently, the attributes that store the pickup and dropoff locations are indexed by using GIST and 2DSphere in Postgres and MongoDB respectively.

Two different queries could be executed in Postgres to determine the *k*NN of a given random query point *p*. The first one uses the same *trips* table twice. It calculates the *k*NN of the randomly chosen trip's pickup location, whose *id* field is the same in both of them. The second approach uses a subquery to first select the query point, and then the *k*NN of the pickup location of this trip is determined. Since the second approach do not multiply the same table twice, it is expected to be faster. In MongoDB, the database is referred to as *nyc2015* and the query depicted in Figure 2 is used to determine the *k*NN of the given query point *p*.



```
select t2.id
from trips t1, trips t2
where t1.id = p.id
order by t1.l_pickup <-> t2.l_pickup
limit k
                                        v1
```

```
select id
from trips
order by l_pickup <->
              (select l_pickup
               from trips
               where id = p.id)
limit k
                                        v2
```

```
db.nyc2015.find( {
   l_pickup: {
     $nearSphere: {
       $geometry: {
         type: "Point",
         coordinates: [p.x, p.y] } } } }
                   ).limit(k)
```

Figure 2 *k*NN queries in Postgres (v1 and v2) and MongoDB

In all of the aforementioned distance calculations, the shortest distance between two points on the earth is calculated. Sphere and ellipsoid are the most suitable geometric shapes to represent the earth mathematically. Haversine and Vincenty distance functions could be used to calculate the distance between two points, where the coordinates are represented in geographical projection system (i.e. latitude and longitude).

Different DBMS could detect different nearest neighbours for the same query point *p*. In order to assess the spatial accuracy of the detected nearest neighbours, Haversine and Vincenty distances could be used. Assume that the most distant nearest neighbour of point *p* for a given *k* value is denoted as x_k^P and x_k^M for Postgres and MongoDB respectively. The Haversine distance between the point of interest and its most distant nearest neighbour found in Postgres and MongoDB is denoted as $\mathrm{H}(p, x_k^P)$ and $\mathrm{H}(p, x_k^M)$ respectively. Similarly, Vincenty distance is denoted as $\mathrm{V}(p, x_k^P)$ and $\mathrm{V}(p, x_k^M)$. If, for instance, $\mathrm{H}(p, x_k^M) < \mathrm{H}(p, x_k^P)$, then MongoDB's most distant nearest neighbour is closer to point *p* than Postgres' most distant nearest neighbour. In other words, Postgres would have detected other points that should not be considered within the set of nearest neighbours making it less accurate.

3.1 Haversine Distance

The shape of the earth is considered as a sphere in the Haversine distance, which is also referred to as the great circle distance. It is calculated as shown in Equation 1:

$$d = 2r\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\Phi2-\Phi1}{2}\right)+\cos(\Phi1)\cos(\Phi2)\sin^2\left(\frac{\lambda2-\lambda1}{2}\right)}\right) \quad (1)$$

The radius of the earth (*r*) is assumed to be 6371 km and Φ_p, λ_p are the latitude and longitude of point *p* respectively.

The Haversine distance function requires a single parameter, which is the radius of the earth. Therefore, its calculation is simple. However, it is not as reliable as the Vincenty function as it assumes a sphrecial earth model.

3.2 Vincenty Distance

Vincenty distance could be used in two different ways. First, in the direct Vincenty problem, the coordinate of the first point and its distance to the second point are given, and the coordinate of the second point is calculated. Second, in the inverse Vincenty problem, the coordinate of two points is given and the distance between these two points and azimuth angle are calculated (Vincenty, 1975). This paper relies on the latter approach, as the aim is to calculate the distance between two points.

The shape of the world could be modelled using an ellipsoid, which is what Vincenty distance function does. Different ellipsoids such as WGS84, GRS50 and ED50 could be used to model the earth, and therefore its calculation is more complex compared to the Haversine function. This paper relies on the WGS84 ellipsoid parameters, as it is the most common used global reference system. (Veness, 2019) provides a detailed formulation as well as the Python script to calculate the Vincenty distance, which is what this paper relies on.

4. RESULTS

This section describes the experimentation results regarding the performance and effectiveness of the *k*NN query obtained by analysing the taxi data of New York City in 2015. The taxi trip records involving 19 attributes are first successfully imported into the staging table of Postgres amounting to 146,112,989 trips. While importing the raw data into Postgres, a unique ID attribute is also generated to ease the comparison between Postgres and MongoDB. Second, *postgres2GeoJSON* function is used to create chunks of GeoJSON files, which are then used to import the data from Postgres to MongoDB. While importing chunks of taxi trips into MongoDB one chunk failed to load, where each chunk contains two million trips. In order to conduct the analyses on time, those trips are also removed from Postgres. Finally, two geometry attributes (i.e. a point in WGS84 – EPSG: 4326) are created in Postgres on top of the staging table by using the latitude/longitude of the pickup and dropoff points respectively.

MongoDB requires the spatial field to be indexed in order to facilitate spatial queries. Therefore, *pickup* and *dropoff* locations are indexed using the '2dsphere' index. In order to provide a fair-comparison framework, the same attributes are also indexed in Postgres using the 'gist' index.

The overview of the systems and the data that are used in the analyses are provided in Table 1. All of the experiments are carried out on a computer having a 16 GB RAM with a CPU of 3.60 GHz.

Table 1 Overview of the experimentation platform

	Postgres	**MongoDB**
Version	9.6.11 with PostGIS 2.5	4.1.6
Licence	PostgreSQL License	GNU AGPL v3.0
Gui	pgAdmin III	Studio 3T
Spatial Index	Gist	2dsphere
Temporal Index	Btree	Ascending
Size on Disk	27.5 GB	22.3 GB
Total trips	144,112,989	

Two main criteria are evaluated in the experimentation. First, execution times of *k*NN queries are recorded for different values of *k*. Second, match percentages of the detected neighbours are compared between two types of Postgres queries (Postgres-v1 and Postgres-v2) and also between Postgres-v2 and MongoDB. Once it is shown that Postgres and MongoDB may identify different sets of points as neighbours, Vincenty and Haversine functions are used to determine which DBMS produces more accurate results. In order to have a better understanding of the performance of DBMSs, the analyses have been carried out on the whole year as well as on a single randomly chosen day.

It should also be noted that there are erroneous trips in the dataset such as trips taking longer than a day or trips having the same pickup and dropoff location or time (Donovan and Work 2017). However, one of the major source of error is the lack of GNSS signal while recording the pickup and dropoff locations. In that case, the value of the latitude and longitude would be zero. There are in total of 2,266,845 such trips in the investigated dataset. If such a point is randomly chosen, all of its *k*NN would be in that set, which would not be meaningful in our analysis. Therefore, it is assured that no randomly chosen query pickup point has a zero latitude or longitude.

4.1 Year-long analysis

In this section all of the available trips in both Postgres and MongoDB databases are used. Thirty random pickup points are determined and *k*NN of these points are determined for $k \in \{1, 10, 100, 1K, 10K, 100K\}$. Even though it is rare to determine 100K nearest neighbours of a query point, it is still important to observe how the databases behave when such large *k* values are provided. Average execution times for all the 30 query points are shown in Figure 3.

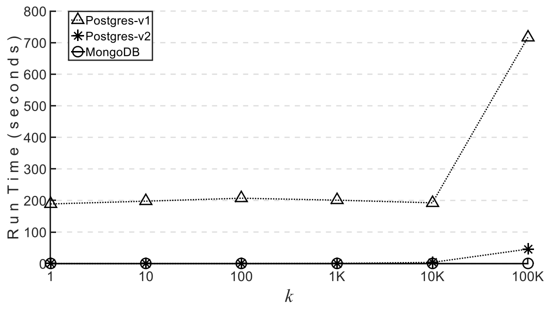


Figure 3 Execution analysis for whole year of 2015

As expected Postgres-v2 is faster than Postgres-v1, since it first finds the query point and then calculates the distance to the remaining points. Execution times of Postgres-v1 and Postgres-v2 remain steady until *k* reaches 100K. At that point, the average execution times increase substantially and reach 715 and 45 seconds in Postgres-v1 and Postgres-v2 respectively. This complex behaviour is probably related to the default parameter settings of Postgres (e.g. effective_cache_size, work_mem, max_fsm_pages etc.) as well as the index and the operating system (Cao et al., 2008). MongoDB, just like Postgres-v2, first determines the query point and then determines the *k*NN of that point. Surprisingly, MongoDB detected the *k*NN of a query point immediately and regardless of the value of *k* unlike Postgres-v2.

The second part of the analysis is to determine the match percentage of the detected neighbours of different approaches. Assume that the set of points belonging to the query point are denoted as **NN(P-v1)**, **NN(P-v2)** and **NN(M)** for Postgres-v1, Postgres-v2 and MongoDB respectively. Thereon, the match percentage between, for instance, Postgres-v2 and MongoDB would be calculated as $|\mathbf{NN(P\text{-}v2)} \cap \mathbf{NN(M)}| \times 100$. This process is repeated for all the 30 query points and *k* values, between Postgres-v1 and Postgres-v2; and Postgres-v2 and MongoDB. The boxplot of the results are illustrated in Figure 4 and Figure 5 respectively.

It should be noted that it is likely to observe several trips having exactly the same latitude and longitude values in their pickup locations. Therefore, once *k* is set to one, any of these points would be acceptable. For this reason, when *k* is equal to one, we could either observe a complete match or no match at all.

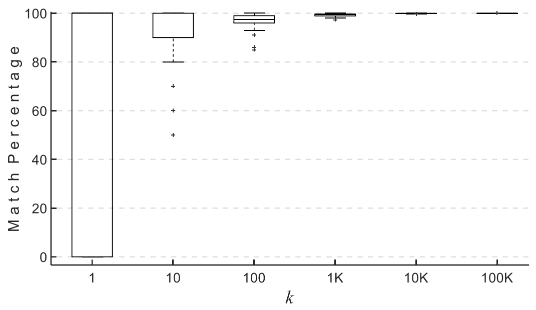


Figure 4 Match percentage of Postgres-v1 and Postgres-v2

It appears that Postgres-v1 and Postgres-v2 identified different portions of the points occurring at exact locations as the nearest neighbour; hence, both no match and perfect match were observed. As the value of *k* increases, so does the match percentage between different Postgres approaches. Specifically, the match percentage rises from 97.50% to 99.95% as *k* increases from 10 to 100K. The remaining part of the analysis relied on Postgres-v2 as it executes faster.

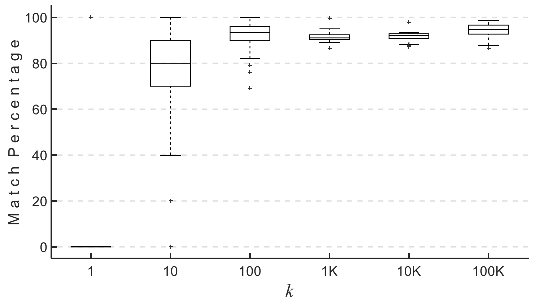


Figure 5 Match percentage of Postgres-v2 and MongoDB

MongoDB and Postgres-v2 match only on six points out of 30 randomly chosen points when *k* equals to one. These points defined as outliers because the matching rate is zero. The median match percentages increase from 80% to 94.8% as *k* increases from 10 to 100K. However, the increment in the match percentages are not linear and when *k*=1K, it becomes 91.05%. Even though, the match percentages are quite high for $k \geq 10$, it is still important to highlight the distinction between the detected nearest neighbours.

The results indicate that there the detected neighbours for the same point might vary depending on the DBMS. Therefore, it is important to determine which one to rely on when spatial accuracy is considered. Recall that $V(p, x_k^P)$ and $V(p, x_k^M)$ denote the Vincenty distance between the query point p to its most distant nearest neighbour in Postgres and MongoDB respectively. Our empirical results suggest that there is no significant difference between Haversine and Vincenty distance functions because points are very close to each other. Therefore, Vincenty distance function is preferred to determine the spatial accuracy as it is more accurate compared to the Haversine function (Mahmoud and Akkari 2016).

The $V(p, x_k^P)$ and $V(p, x_k^M)$ are calculated for each of the randomly chosen p values. This process is repeated for all of the 30 random points and $k \in \{1, 10, 100, 1K, 10K, 100K\}$. Consequently, a total of 180 distance values are recorded and the scatter plot of these values are illustrated in Figure 6, where x-axis denotes $V(p, x_k^P)$ and y-axis denotes $V(p, x_k^M)$.

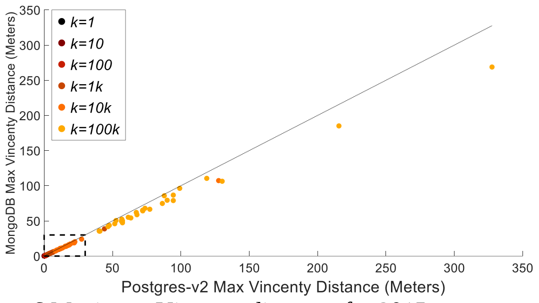


Figure 6 Maximum Vincenty distances for 2015

If all the points were on the illustrated diagonal line, then one would suggest that both DBMS are the same regarding spatial accuracy. If the plotted values are below the line and closer to Postgres, then it would mean that MongoDB is more accurate since its most distant nearest neighbour is closer than Postgres' most distant neighbour, which is indeed the outcome. Formally, the results suggest that $V(p, x_k^P) > V(p, x_k^M)$. Even though the difference between the Vincenty values gets generally larger with higher values of k, it is still possible to observe small differences as well, which is the main reason why k=1 points are not visible on the graph as other points were located on top of them. In order to improve the legibility of the results, Vincenty distances up to 30 metres are illustrated in Figure 7, which corresponds to the region denoted in dashed lines.

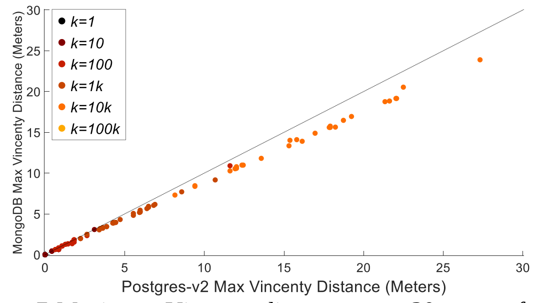


Figure 7 Maximum Vincenty distance up to 30 meters for the whole year

Out of the 180 points, 131 of them are below the line. Remaining 49 points are exactly on the x=y line. Interestingly, Postgres-v2 have not found a better result at any query point for any given k value.

4.2 Single day analysis

All of the aforementioned analyses has been repeated for a single day, which is randomly chosen to be 23 May 2015, on which approximately 383 thousand trips occurred. For this day, another set of 30 random pickup points are determined and kNN of these pickup locations are determined for $k \in \{1, 5, 10, 20, 50, 100\}$. Average execution times of the kNN query of these 30 points are compared on various k values, which are shown in Figure 8.

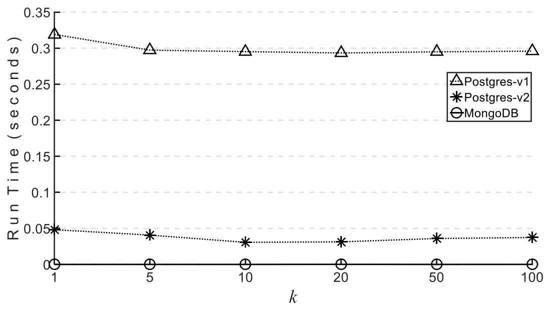


Figure 8 Average execution time analysis on 23 May 2015

As expected, MongoDB outperforms PostgreSQL in terms of execution time regardless the value of k, since all the neighbours were almost immediately found again. The average execution times of Postgres-v1 and Postgres-v2 are steady and about 0.3 and 0.04 seconds respectively. In this scenario, an abrupt increase in the execution times are not observed for the investigated k values, since probably default Postgres parameters were fine enough and that the k values are much smaller compared to the ones used in the year-long analysis.

The second analysis is about identifying the match percentage of the detected nearest neighbours between different approaches. If the ratio is one, then both of the queries would have detected the same points as the kNN of the query point. The match percentages for all the 30 query points are identified for all the k values between Postgres-v1 and Postgres-v2, and Postgres-v2 and MongoDB. The boxplots are illustrated in Figure 9 and Figure 10 respectively.

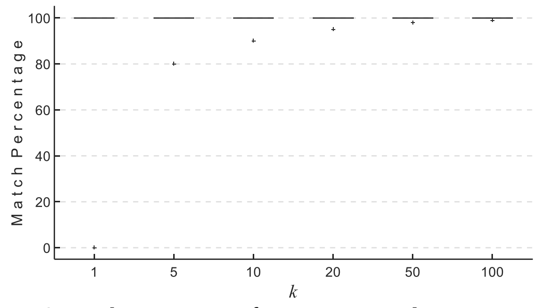


Figure 9 Match percentage of Postgres-v1 and Postgres-v2

Postgres-v1 and Postgres-v2 match perfectly regardless of the value of k except the outliers. For instance, there are three outliers when k=5. The closest four nearest neighbours are common in both Postgres-v1 and Postgres-v2; hence, the match percentage is 80%. Consequently, the most distant nearest neighbour is not common, since these points are equidistant to the query point but have different locations. When the third outlier is analysed, it is observed that Postgres-v2 detected a point that is actually farther from what Postgres-v1 had detected. It should also be noted that the opposite situation have also been observed when k=20, in which the spatial accuracy of Postgres-v1 is better than Postgres-v2. Last, it should be noted that two or more pickup locations could exactly be the same, which also contributes to the existence of outliers.

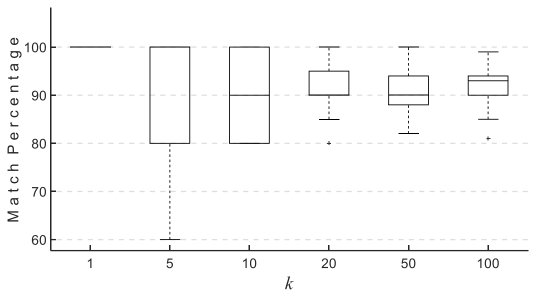


Figure 10 Match percentage of Postgres-v2 and MongoDB

Unlike the comparison between the Postgres approaches, there is more variation regarding the match percentage between Postgres-v2 and MongoDB. The match percentage ranged from 80% to a complete match when k is 20 with a median of 90%. Likewise, median match percentages are 92% and 92.5% when k values are 50 and 100 respectively. Therefore, it could be stated that over 90% of the detected neighbours match between Postgres and MongoDB regardless the value of k.

Just like the year-long analysis, the $\mathrm{V}(p, x_k^P)$ and $\mathrm{V}(p, x_k^M)$ are calculated for each of the randomly chosen p values. This process is repeated for all of the 30 random points and $k \in \{1, 5, 10, 20, 50, 100\}$. Consequently, a total of 180 distance values are recorded and the scatter plot of these values for the analysed day is illustrated in Figure 11, where x-axis denotes $\mathrm{V}(p, x_k^P)$ and y-axis denotes $\mathrm{V}(p, x_k^M)$.

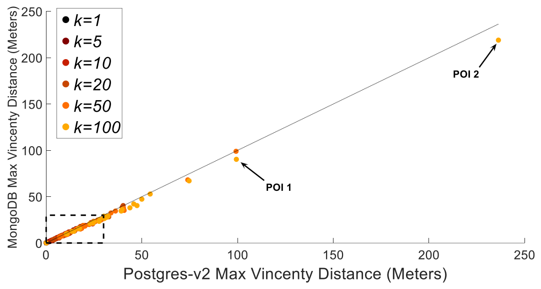


Figure 11 Maximum Vincenty distances on 23 May 2015

The results are in-line with what had been observed on the year-long analysis and that MongoDB outperformed Postgres in terms of spatial accuracy. In this analysis, two points deserve attention. Point-of-interest (POI) – 1 has the largest Vincenty difference $\mathrm{V}(p, x_k^P) - \mathrm{V}(p, x_k^M)$. On the other hand, POI-2 has the largest maximum $\mathrm{V}(p, x_k^P)$ and $\mathrm{V}(p, x_k^M)$ values, indicating that the randomly chosen point p is not at a central location. In order to improve the legibility of the results, distances up to only 30 metres are visualised in Figure 12 for a better interpretation.

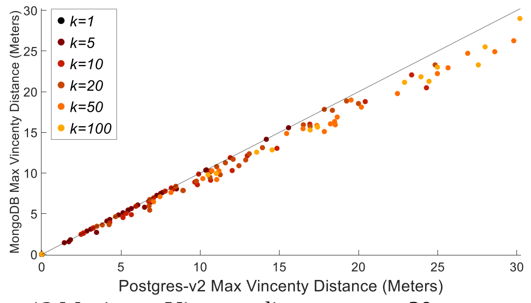


Figure 12 Maximum Vincenty distances up to 30 meters

Out of the 180 values, 68 of them are on the x=y line and 112 of them are below the line supporting the previous finding. Results show that MongoDB provides more accurate and faster results compared to Postgres-v2 regarding the detection of kNN of a query point regardless the value of k.

4.3 POI Based Analysis

This subsection provides an in-depth analysis of the two POIs illustrated in Figure 11. The lowest match percentage between Postgres and MongoDB has occurred on POI-1. Only 75% of the detected neighbours matched out of the k=100 neighbours. Even though the difference $\mathrm{V}(p, x_k^P) - \mathrm{V}(p, x_k^M)$ is only nine meters, it is still important to highlight the success of MongoDB. The POI and its neighbours detected by Postgres-v2 and MongoDB are illustrated in Figure 13.

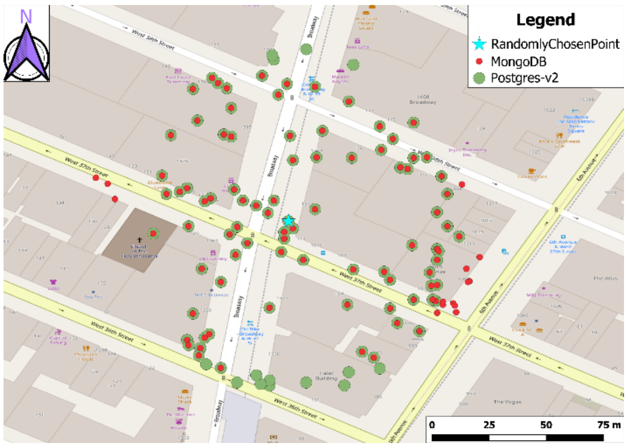


Figure 13 POI-1 – the largest Vincenty difference Postgres-v2 and MongoDB

Another interesting query point is POI-2, where the largest $\mathrm{V}(p, x_k^P)$ and $\mathrm{V}(p, x_k^M)$ values are observed, which is approximately 240 meters. The difference between the maximum Vincenty distances of Postgres-v2 and MongoDB is about seven meters. The POI-2 and its 100 nearest neighbours are illustrated in Figure 14.

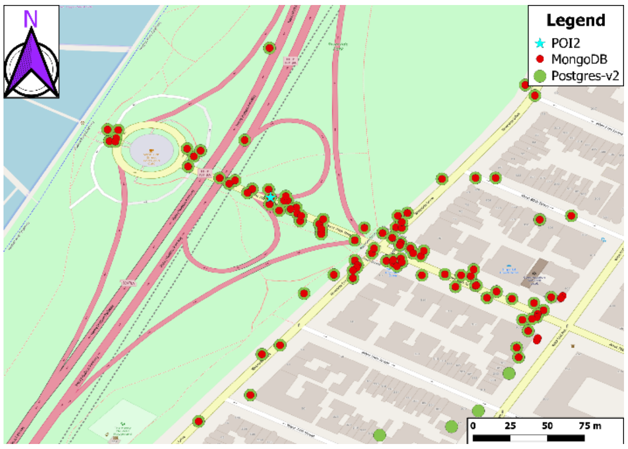


Figure 14 The biggest maximum Vincenty distance visualization

As expected, this point occurred on the suburbs of New York City, which is close to a highway near the Hudson River. First, due to topography of the area, neighbours cannot spread all directions. In addition, the area is not as densely populated as financial district Manhattan where the majority of taxi trips take place.

5. CONCLUSION

The need to choose the correct DBMS to store spatial data is eminent. Traditional way is to rely on relational DBMSs, such as Postgres, due to their natural linkage with GIS. However, the emergence of web based systems, which are dynamic in nature, prohibit defining a data schema. Therefore, researchers developed non-relational DBMSs such as MongoDB, which also support spatial data types.

Extensive test scenarios need to be carried out to determine the performance of DBMS for a given scenario. In this paper, the performance of a common spatial query –detecting the *k*NN of a query point- is investigated for the aforementioned DBMSs. The experiments are carried out on the openly available taxi dataset of New York City. Results favour MongoDB as it is faster and more accurate compared to Postgres.

While importing the data to both of the DBMS as well as creating the spatial indices, default parameter settings are used. In this way, a spatial analyst, who might not have the technical skills to tune the relevant parameters, could easily rely on the findings of this paper.

The future work will focus on different spatial queries as well as those that concern different spatial data types including lines and polygons. In addition, the effectiveness of the parallel query or partitioning features of Postgres could be investigated. Last, other DBMS that support spatial data types could be included in the developed library.

ACKNOWLEDGEMENTS

This research is supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with the grant number of 118Y282. The authors are also thankful to the online communities, StackOverflow and GIS StackExchange. The contents of this paper reflect the views of the authors, who are responsible for the facts and the accuracy of the results presented herein.

REFERENCES

Agarwal, S., K. S. R., 2016. 'Performance Analysis of MongoDB versus PostGIS/PostGreSQL Databases for Line Intersection and Point Containment Spatial Queries'. *Spatial Information Research* 24 (6): 671–77.

Alamri, S., 2018. 'Spatial Data Managements in Indoor Environments: Current Trends, Limitations and Future Challenges'. *International Journal of Web Information Systems* 14 (4): 402–22.

Cao, Y., Das, G.C., Chan, C.Y., Tan, K.L., 2008. 'Optimizing Complex Queries with Multiple Relation Instances'. In *ACM SIGMOD International Conference on Management of Data*, 525–538.

Chen, Z., Shen, H.T., Zhou, X., Zheng, Y., Xie X., 2010. 'Searching Trajectories by Locations: An Efficiency Study'. In *ACM SIGMOD International Conference on Management of Data*, 255–266.

Coşkun, B., (2019) 2019. *K-NN: Postgres vs MongoDB*. Python. https://github.com/bugracoskun/K-NN.

Cover, T., Hart, P., 1967. 'Nearest Neighbor Pattern Classification'. *IEEE Transactions on Information Theory* 13 (1): 21–27.

Donovan, B., Work, D., 2014. 'New York City Taxi Trip Data (2010-2013)'. http://dx.doi.org/10.13012/J8PN93H8.

Donovan, B., Work, D., 2017. 'Empirically Quantifying City-Scale Transportation System Resilience to Extreme Events'. *Transportation Research Part C: Emerging Technologies* 79: 333–346.

Dritsas, E., Trigka, M., Gerolymatos, P., Sioutas S., 2018. 'Trajectory Clustering and K-NN for Robust Privacy Preserving Spatiotemporal Databases'. *Algorithms* 11 (12): 207.

Freire, S.M., Teodoro, D., Wei-Kleiner, F., Sundvall, E., Karlsson, D., Lambrix, P., 2016. 'Comparing the Performance of NoSQL Approaches for Managing Archetype-Based Electronic Health Record Data'. *PLOS ONE* 11 (3): e0150069.

Hu, L., He, S., Han, Z., Xiao, H., Su, S., Weng, M., Cai, Z., 2019. 'Monitoring Housing Rental Prices Based on Social Media:An Integrated Approach of Machine-Learning Algorithms and Hedonic Modeling to Inform Equitable Housing Policies'. *Land Use Policy* 82: 657–73.

Mahmoud, H., Akkari, N., 2016. 'Shortest Path Calculation: A Comparative Study for Location-Based Recommender System'. In *2016 World Symposium on Computer Applications Research (WSCAR)*, 1–5.

Makris, A., Tserpes, K., Spiliopoulos, G., Anagnostopoulos D., 2019. 'Performance Evaluation of MongoDB and PostgreSQL for Spatio-Temporal Data'. In . Vol. 2322.

Matuszka, T., Kiss, A., 2014. 'Experimental Evaluation of Some Geodata Management Systems'. In *2014 9th International Conference on Computer Engineering Systems (ICCES)*, 92–97.

Mehta, P., Dorkenwald, S., Zhao, D., Kaftan, T., Cheung, A., Balazinska, M., Rokem, A., Connolly, A., Vanderplas, J., AlSayyad, Y., 2017. 'Comparative Evaluation of Big-Data Systems on Scientific Image Analytics Workloads'. *Proc. VLDB Endow.* 10 (11): 1226–1237.

Nguyen, T. T. T., 2009. 'Indexing PostGIS Databases and Spatial Query Performance Evaluations'. *International Journal of Geoinformatics* 5 (3).

Pereira, D.A., Wagner Ourique de Morais, Edison Pignaton de Freitas. 2018. 'NoSQL Real-Time Database Performance Comparison'. *International Journal of Parallel, Emergent and Distributed Systems* 33 (2): 144–156.

Schmid, S., Galicz, E., Reinhardt, W., 2015. 'WMS Performance of Selected SQL and NoSQL Databases'. In *International Conference on Military Technologies (ICMT) 2015*, 1–6.

Veenendaal, B., Brovelli, M.A., Li S., 2017. 'Review of Web Mapping: Eras, Trends and Directions'. *ISPRS International Journal of Geo-Information* 6 (10): 317.

Veness, C., 2019. 'Vincenty Solutions of Geodesics on the Ellipsoid'.https://www.movabletype.co.uk/scripts/latlong-vincenty.html.

Vincenty, T., 1975. 'Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations'. *Survey Review* 23 (176): 88–93.

Zhong, R., Li, G., Tan, K.L., Zhou L., 2013. 'G-Tree: An Efficient Index for KNN Search on Road Networks'. In *Proceedings of the 22. ACM International Conference on Information & Knowledge Management*, 39–48.