PERFORMANCE MATTERS ON IDENTIFICATION OF ORIGIN-DESTINATION MATRIX ON BIG GEOSPATIAL DATA

İ. B. Coşkun [1], A. Çakır[1], B. Anbaroğlu [1]*
[1] Dept. of Geomatics Engineering, Hacettepe University, Turkey – (ihsan.coskun, abdulkadircakir, banbar)@hacettepe.edu.tr

Commission IV, WG IV/7

KEY WORDS: point-in-polygon, origin-destination, database performance, GIS, visualisation

ABSTRACT:

One of the common problems at the intersection of geographical information science and transportation science is the estimation of origin-destination (OD) matrices. The emergence of sensor technologies offers unprecedented opportunities in this regard since massive amounts of traffic data can be collected in an easy way. Researchers and practitioners need to choose a suitable DataBase Management System (DBMS) among alternatives, such that storing and analysing traffic data to estimate the OD matrix is feasible. The aim of this paper is to compare the performance of two such notable DBMSs, PostgreSQL and MongoDB, in the context of OD matrix estimation. The experiments are carried out on New York City's openly available taxi data on two different polygon sets: taxi zones and census blocks. These polygon layers consist of 263 and 38794 features respectively. The results suggest that Postgres outperforms MongoDB by generating the OD matrix instantly. The run time of MongoDB varies depending on the analysed time interval and follows a trip demand curve. As there are more trips involved in the generation of the OD matrix, so does the execution time increases in MongoDB. On the other hand, the query results are the same. Finally, the origin points of the taxi trips are visualised in QGIS using the 'TimeManager' plugin, and results are presented through a web-interface.

1. INTRODUCTION

Origin-Destination (OD) matrix is the representation of the travel demand between different origin and destination pairs of a study region. The OD matrix is a crucial input to a variety of research interests ranging from traffic simulation modelling to understanding mobility patterns and developing effective transportation systems. The cell value of an OD matrix represents the traffic flow between an individual OD pair. The progress in information technology plays a key role to increase the quality of an estimated OD matrix (Munizaga and Palma, 2012).

Using probe vehicles to estimate an OD matrix is an emerging research interest due to the pervasive availability of vehicle tracking technology. In this way, time-stamped location data of a vehicle can be collected. Taxis are considered to be a valuable resource in this context due to their fine granularity in space and time (Garcia et al., 2018). Considering that there can be thousands of taxis in an urban environment, where each one makes tens of journeys in a day, it is easy to imagine the growth of data to analyse. Consequently, practitioners need to identify the correct DataBase Management System (DBMS) to store collected data.

There are two main types of DBMSs: relational and non-relational. Relational databases store data in tables, which is the traditional approach to manage data. Relationships between tables are established during the database design phase, which means that data are logically consistent. Relational databases follow the Atomicity, Consistency, Isolation and Durability, also referred to as ACID, meaning that each transaction is processed reliably (Sveen, 2019). On the other hand, non-relational databases also called NoSQL (Not Only SQL) do not follow a pre-defined schema. NoSQL databases are widely used in recent years due to the pervasive use of web and mobile technologies. Non-relational databases do not establish relationships between tables such as relational databases. Therefore, non-relational databases are suitable when in streaming data, where it might be difficult to establish relationships or define a schema. Since non-relational DBMSs are suitable for real-time and dynamic data, web-based systems usually rely on them (Bugiotti et al., 2014). Managing large volumes of traffic data through the web is also gaining importance due to the real-time estimation of traffic volume.

Traffic volume increases with increasing population every year. Therefore, estimating traffic demand in real-time is an important problem. Furthermore, it is very important for planning and optimization of transportation management. For example, OD taxi demand prediction helps dynamic allocation of resources to meet travel demand and to reduce empty taxis on the streets (Xian et al., 2020). In this way, wasted energy, as well as traffic congestion, can be reduced. Therefore, the correct choice of DBMS becomes even more important in this case. Even though the necessity to rely on a DBMS to store and analyse traffic data to estimate the OD matrix is evident, there is no study comparing the performance of different DBMS.

The aim of this paper is to compare the performance of two commonly used DBMSs, PostgreSQL (Postgres) and MongoDB, regarding the estimation of the OD matrix. The performance is measured in two aspects: spatial accuracy and run-time. The former DBMS has a natural linkage with QGIS due to its relational structure, whereas the latter is non-relational DBMS (NoSQL) that is commonly used in web-based applications (Gebetsroither-Geringer et al., 2018). In order to make the experimental results repeatable and improve their development, the source code of this research is shared on GitHub (Coskun, 2020).

The remainder of this paper is organised as follows. The second section provides the literature review regarding the use of origin-destination matrixes and comparison of spatial databases. The

* Corresponding author

third section provides the methodology of the paper. The fourth section provides the results on a large-scale taxi dataset and visualisation of origin taxi trips. Finally, the fifth section concludes the paper by providing a discussion of the results and their implications for future studies.

2. LITERATURE REVIEW

There are a remarkable number of DBMSs in the market. Specifically, 354 DBMSs are listed and compared based on their popularity on the database ranking site db-engines.com as of March 2020. Therefore, management of data becomes more important and complex. Performance comparison of databases is a prequisite to understand their effectiveness in certain situations.

Recent research evidence compares the performance of relational DBMS and non-relational DBMS, in which MySQL and MongoDB are used respectively (Deari et al., 2018). This research shows that MongoDB is a serious competitive DBMS compared to MySQL and it outperforms relational databases in some situations, especially when the data are not structured and simple to handle. Some researchers have already started investigating the performance of spatial queries of different database management systems. For example, Postgres and MongoDB are compared based on the K-Nearest Neighbour (KNN) query and it is found out that MongoDB is not only faster but also more spatially accurate than Postgres on a large taxi dataset (Coşkun et al., 2019).

Indexing of the data is applied to speed up the query in databases. Spatial data can also be indexed using various data structures including R-trees, kd trees or quadtrees. Indexing method is important as it can significantly improve query performance. One of the widely used spatial index structures is R-trees. An investigation of performance comparison on R-trees methods, such as the Hilbert R-tree and the SR-tree, shows that traditional R-trees may not be the most efficient way of spatial indexing, and one of the special method of R-tree which is '*R-tree CR*' are better in terms of point queries (Ciferri et al., 2003).

Consequently, in order to effectively identify an OD matrix, researchers need to store their historical and real-time traffic data in a DBMS (Lederman and Wynter, 2011). The spatial query that is required to determine the OD matrix is referred to as point-in-polygon (pip) query. Different DBMS is used to estimate the OD matrix. For example, MySQL is used to estimate the OD matrix using a transit passenger trip (Li et al., 2011). Another research that investigated 60K distance computation per second utilised Postgres (Peng et al., 2018). In some other researches that mention the keyword 'database', it might not be clear which DBMS they relied on. For example, research on the estimation of OD matrix using time-dependent traffic information (Cho et al., 2009), and using smartcard and GPS data (Munizaga et al., 2014) have not specified the DBMS they relied on.

Different methods can be used to estimate an OD matrix. The traditional method which is Levenshtein Distance (LD) is a string metric for measuring the difference between two nodes. A study on the OD matrix tries to develop this method and offers Normalised Levenshtein Distance method for OD matrices (NLOD). The sensitivity analysis shows that NLOD gives more robust statistical results according to the LD method (Behara et al., 2020).

Most of the existing research identified the OD matrix on a desktop environment. Recent research efforts; however, also focus on a web-based environment. A web GIS was developed based on the New York taxi data set, which has approximately 170 million taxi trips, investigated point-in-polygon queries. The most important feature of this study was that no database is used, and data were stored in JSON files only. Additionally, point-in-polygon query was applied using JavaScript. However, the experiment relied on only two polygons, and the query time took about 300 milliseconds on the backend side (Zhang et al., 2015). In the results, only the working time of the query in the backend is given, and reporting the total execution times might be equally important.

3. METHODOLOGY

The proposed methodology assesses the performance of Postgres and MongoDB on run-time and comparison of the OD matrices. Three inputs are required from the user: date d, time range Δt, and top K OD pairs. The parameter d is one day within analysis period, and Δt defines the time interval of analysis in minutes. Specifically, the OD matrix is generated for $t = \{1,2,\dots,(24*60)/\Delta t\}$ intervals within a day. Visualisation of all OD pairs which had at least one trip would be difficult to comprehend due to a large number of regions and taxi trips. Therefore, this research is interested in the top-K OD pairs that occurred within the analysed date d for each of the time interval t. Consequently, for an analysed time interval t there are K tuples with the following form: $< O_k^t, D_k^t, y_k^t >$, where $k = \{1,2,\dots,K\}$, and y_k^t denotes the number of trips that occurred from O_k^t to D_k^t at the analysed time interval t. The following relation holds $y_i^t \geq y_j^t, \forall i < j$. The methodology is illustrated in Figure 1.

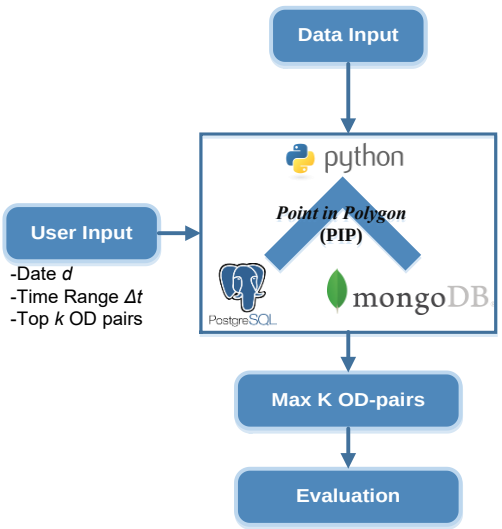


Figure 1. Methodology

Indexing improves data operations on a database table like searching, updating etc. There are lots of methods to index data like B-tree, R-tree etc (Nguyen, 2009). Which method to choose depends on the data type. Generalized search three (gist) is one of the most popular spatial indexes in Postgres. It is not mandatory to create a spatial index in Postgres. On the other hand, indexing is mandatory in MongoDB to run a spatial query. Therefore, '2Dsphere' spatial index is used in MongoDB. The geometries are calculated on a sphere in this method. Consequently, 'gist' and '2Dsphere' are used to index taxi trip start and end locations, for both of the polygon layers (i.e. taxi zones and census blocks).

The OD matrix can be identified in just a single query in Postgres. Two tables are required, which are trips and polygons data. Trips table contains whole trips of the year 2015. On the other hand, 'poly' represents the polygon layers, which might either be taxi zones or census blocks. The PostGIS function that finds the polygon which a point is in is 'st_contains'. In the where condition, *start* and *end* refer to the start time and end time of the analysis, respectively. Different time intervals are analysed in this paper to have a better understanding of its effect on the execution time. Point-in-polygon (pip) query in Postgres is shown in Figure 2.

```
Select z1.gid as origin_zone, z2.gid as destination_zone, count(*)
as total_trips
From trips
Full Join 'poly' z1 on St_Contains (z1.geom, t.l_pickup)
Full Join 'poly' z2 on St_Contains (z2.geom, t.l_dropoff)
Where t.t_pickup >= 'start' and t.t_pickup <= 'end'
Group By z1.gid, z2.gid
Order By total_trips desc
```

Figure 2. Postgres – pip query

In MongoDB, pip query is more complicated than Postgres. It is not possible to obtain the OD matrix in a single query. Specifically, two queries are required to determine the OD matrix. The queries will be used are shown in Figure 3.

```
db.nyc2015.find( {
   $and: [
        $Properties.tpep_pickup_datetime:
        { $gte: 'Start' },
        { $lt: 'End' } ] } )
```

```
db.nyc2015.find( {
   $geometry_pk: {
     $geoIntersects: {
        $geometry: {
          type: "Point",
          coordinates: [p.x, p.y] } } } )
```

x2

Figure 3. MongoDB – pip query

The *nyc2015* is the collection name. First query filters data according to date time. In condition, *start* and *end* refer to the analysis period that the OD matrix is generated. The second query finds point in polygons with the aid of *geoIntersects* function. Additionally, the second query is run twice for pickup and dropoff locations. All of these processes complicate what is a single query in Postgres.

3.1 Visualisation of Data

This section describes how to visualise the results. Queries are executed within a Python 3.7 code written in Spyder Integrated Development Environment (IDE). The results of the OD matrix are exported as a txt file. Visualisation of the txt file will be provided via the QGIS plugin entitled '*Time Manager*'. The steps are illustrated in Figure 4.

Figure 4. Methodology of visualisation

4. RESULTS

The experiments are carried out on openly available taxi dataset of New York City (TLC, 2019). Each taxi trip is defined by 19 attributes including, but not limited to, the pickup and dropoff locations, and start and end time of the trip. In addition, two polygon layers are utilised that correspond to taxi zones and census blocks. An exemplar data is illustrated in Figure 5.

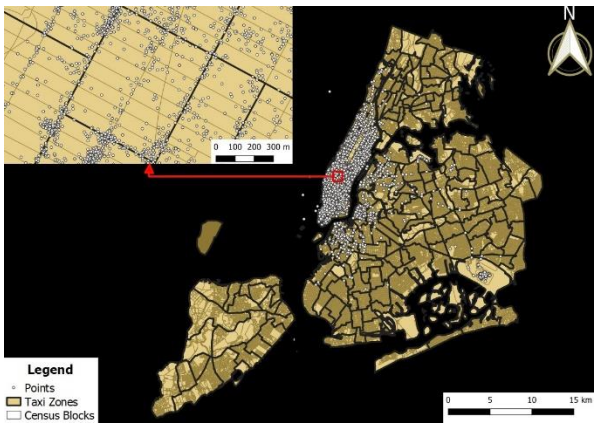


Figure 5. A sample view of pickup locations, taxi zones and census blocks

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

	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	

Table 1. Overall view of the DBMSs

Two types of analysis have been carried out. One day analysis is carried out on a selected random day. The parameters are kept constant to form preliminary ideas. The second part is general analysis describes changing parameters on a selected random

day. The results obtained in this section will add further support to the ones obtained in the former section.

4.1 One Day Analysis

The first experiment is executed with a random day d being 2 February 2015, $\Delta t = 120$ minutes, and $K = 3$. The OD matrix is generated based on the taxi zones consisting of 263 polygons. It should be noted that some pickup or dropoff locations might not fall into a region. In such cases the corresponding O_k^t or D_k^t values will be 'None'. OD matrix generation time of Postgres and MongoDB is illustrated in Figure 6.

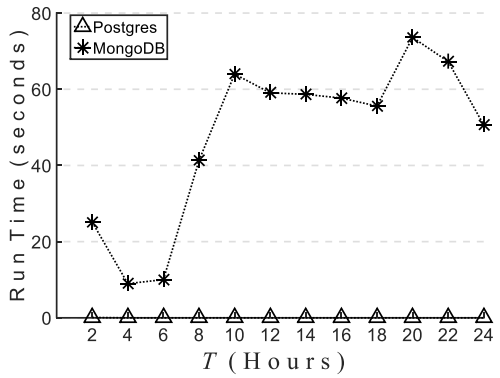


Figure 6. Time comparison of the Query result for taxi zones

Postgres executes almost instantly for taxi zones. On the other hand, MongoDB followed a trip demand curve where the execution lasted longer in morning and afternoon peak periods. As there are more taxi trips occurring at these intervals, so does the execution time last longer in MongoDB.

Additionally, the OD matrix is generated based on the census blocks consisting of 65K polygons. The same parameters are used, and run times are recorded accordingly. Census blocks OD matrix generation time of Postgres and MongoDB is illustrated in Figure 7.

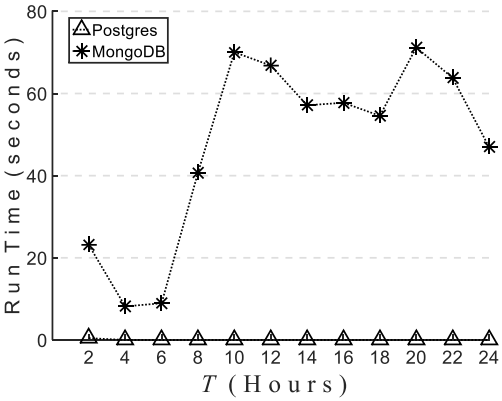


Figure 7. Time comparison of the Query result for census blocks

Consequently, the run time of queries are very close to each other for taxi zones and census blocks. Postgres produced almost instantly again. However, MongoDB execution times vary depending on the analysed time interval. The main reason of this that the query cannot be made directly in MongoDB, and execution time increases with the number of trips occurred in the analysed time interval. Therefore, the execution time follows a similar trend with what we would expect to see in terms of taxi demand. It is also important to highlight that the number of polygons did not affect the execution time of the query.

4.2 General Analysis

Secondly, the tuples $< O_k^t, D_k^t, y_k^t >$ obtained from Postgres are compared with the ones obtained from MongoDB for different time intervals $\Delta t = \{15,30,60,120\}$, and for different $K = \{3,10,100\}$. In addition, taxi zones and census blocks are used in this experiment. All these queries run times and results are saved. As a result, average run times are calculated for each K value separated by time intervals. Average execution times for MongoDB are shown in Figure 8.

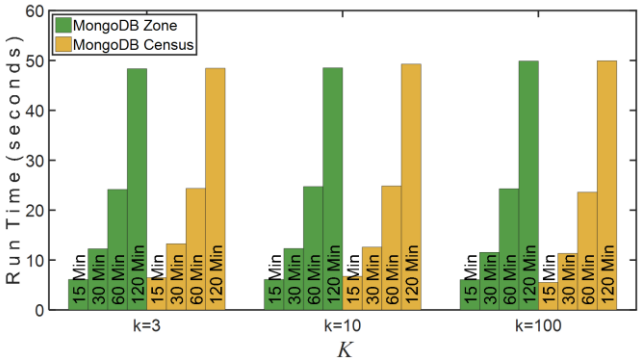


Figure 8. MongoDB run times of queries

As expected, query time increased with a time interval. Sum of the query time is approximately seven seconds when $\Delta t = 15$ minutes. Furthermore, run times are approximately 12, 24 and 48 seconds when $\Delta t = \{30, 60, 120\}$ respectively. On the other hand, results show that query of times is independent from K and number of zones. For example, query time is about 24 seconds when K=3 and $\Delta t = 60$ minutes. It is similar to what had been observed when K= {10,100} with $\Delta t = 60$ minutes. Unexpectedly, number of zones does not change this time. Taxi zone and census results are very similar to each other. The same queries are applied in Postgres and results are shown in Figure 9.

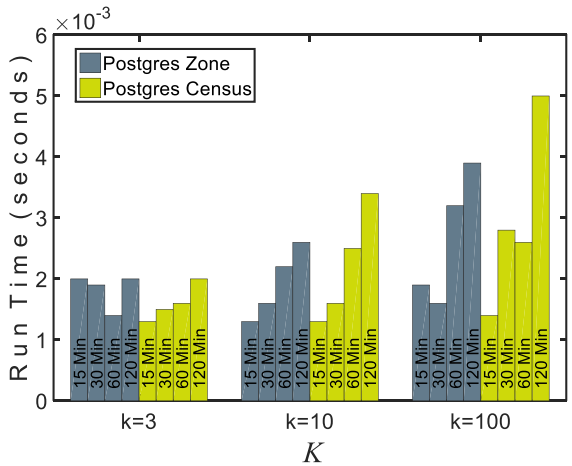


Figure 9. Postgres run times of queries

Postgres query run times very different from MongoDB. The results are found instantly by Postgres. The query of times close to 0.002 seconds when K=3 regardless of time interval and number of zones. The sensitivity analysis demonstrates that the query execution time increases with the K value and time interval although this increase is meaningless with that scale. The maximum query time is 0.005 seconds when K=100 and $\Delta t = 120$ minutes on census data. Consequently, Postgres gave the

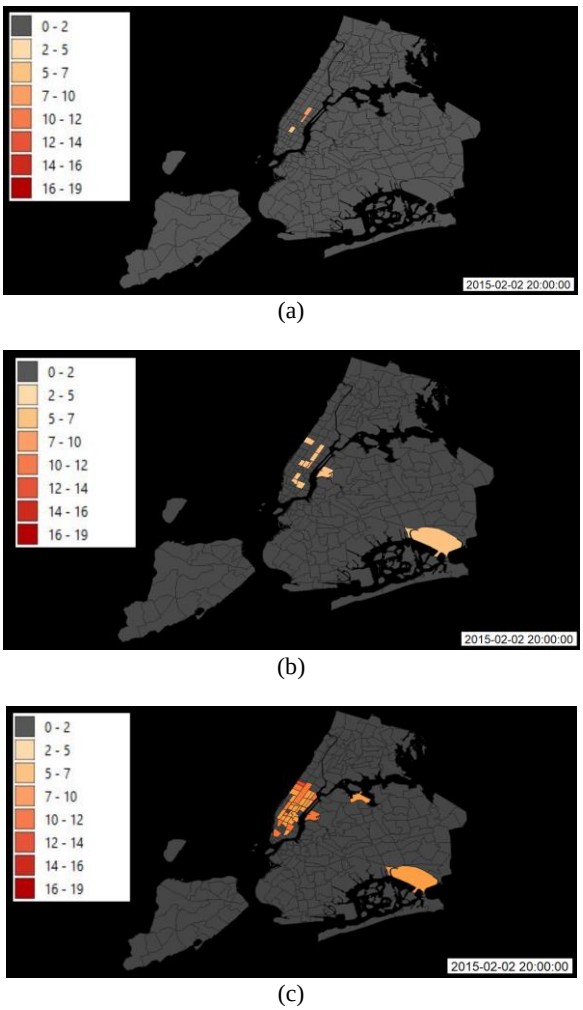
results almost instantly, and it is much faster than MongoDB. After that, the results are compared based on spatial accuracy.

4.3 Spatial Accuracy

The results suggest a match between the results; however, the order of tuples might vary when y_k^t is equal to y_{k+1}^t. Please note that some regions have the same number of different trips. In such cases, although the databases give different travel numbers, they are considered to be the same. Consequently, Postgres and MongoDB gave exactly the same OD matrix.

4.4 Visualisation

An open-source code has been developed to visualise of OD matrix via QGIS time manager plugin. Therefore, the importance of reporting the top *K* OD pairs becomes clear. The results are recorded as a video and shared on YouTube, which demonstrates the clutter once *K* value increases on the pickup of trips (Coskun B, 2020). Examples of screenshots taken when the top-*K* origins are visualised by QGIS time manager for $\Delta t = 15$ when *K*= {3, 10, 100} at 20.00 pm in Figure 10.



(a)

(b)

(c)

Figure 10. Origin taxi trip visualisation on taxi zones $\Delta t = 15$ minutes at 20.00 pm on 2015-02-02 of a) *k*=3 b) *k*=10 c) *k*=100

The first observation is related to the traffic density in the Manhattan region. In addition, it is observed that taxi demand is also intense at JFK airport. Secondly, as seen from the figures, the complexity increases with the increasing number of *K*. Consequently, there is an inverse proportion between the *K* value and legibility. It should also be highlighted that only the top-*K* *origins* are visualised, and once all the OD pairs are visualised the legibility will reduce substantially. Therefore, having a *K* value fit-for-purpose is important. On the other hand, census blocks have much more polygons than taxi zones. Consequently, relying on census blocks might be an ineffective choice.

4.5 Web GIS

The Postgres queries discussed in this paper are transferred to a dedicated website – http://nyc.hacettepe.edu.tr/. It is important to provide a web GIS environment to facilitate queries to detect OD matrix so that results can be investigated easily. In addition, it will be possible to experiment on how the performance varies on a desktop and web environment. An exemplar image of the website is shown in Figure 11.

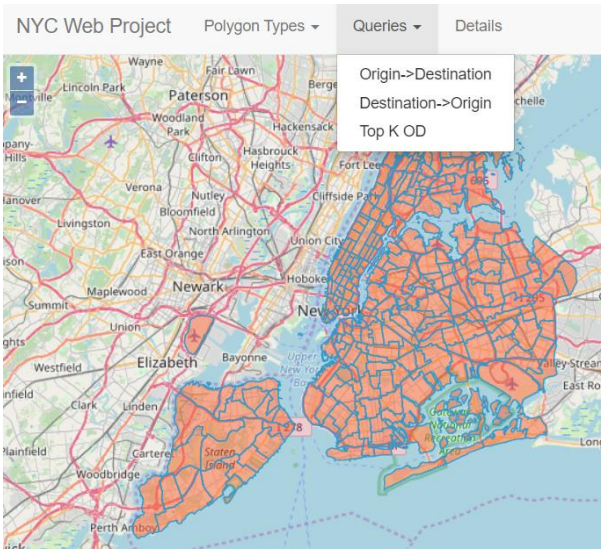


Figure 11. The interactive web interface to conduct OD analysis

There are two types of regions, taxi zones and census blocks under the menu 'Polygon Types'. Three types of queries are present under the 'Queries' tab. Once a polygon is selected the first query, 'Origin->Destination', asks the user to provide a time interval, and the query returns the number of trips that originated from this polygon to other polygons as a CSV file. The second query, 'Destination->Origin' assumes the selected polygon is the destination zone and identifies the number of trips from other polygons that ended up in this polygon. Finally, the top-K OD query determines the top K OD pairs in the provided time interval, which is the same query that is investigated in the previous sections.

5. CONCLUSION AND DISCUSSION

The travel demand between origin and destination polygons can be captured within an OD matrix, which has an important role in various real-world problems including, transportation planning, urban and regional planning. With the growing technology, it becomes more important to do this in real-time. However, performance benchmark tests are required to have an informed decision regarding the correct choice of DBMS to store data. In this way, researchers can have a better understanding of the performance of different DBMS and how they operate.

The experiments carried out in this paper relied on the openly available New York yellow taxi data set obtained in 2015, which

consists of about 144 million trips. Additionally, two different polygon data, taxi zones and census blocks are utilised to identify the OD matrix through a point-in-polygon query. Two different DBMS are analysed: Postgres and MongoDB, a representative of relational and NoSQL DBMS respectively.

Results show that Postgres performed queries much faster than MongoDB. In addition, obtained OD matrices are compared in terms of spatial accuracy, and it is identified that both DBMS provide the same results. Consequently, this paper suggests that Postgres with PostGIS extension is better than MongoDB on point-in-polygon query. In addition, different tools are investigated to visualise the results. First, 'Time Manager' plugin is utilised to demonstrate the top *K* origin polygons in a given time interval. This visualisation demonstrated the importance of relying on the *K* parameter rather than displaying all the possible polygons that generated trips. In addition, an interactive web interface is developed to provide users with an online means of querying data to conduct OD analysis. Future work will focus on improving the performance of web GIS queries. The effectiveness of different OSGeo tools such as GeoMesa or GeoWave can be investigated in this regard.

ACKNOWLEDGEMENTS

This research is supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with the grant number of 118Y282. 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

Behara, K.N.S., Bhaskar, A., Chung, E., 2020. A novel approach for the structural comparison of origin-destination matrices: Levenshtein distance. Transportation Research Part C: 513–530. https://doi.org/10.1016/j.trc.2020.01.005

Bugiotti, F., Cabibbo, L., Atzeni, P., Torlone, R., 2014. Database Design for NoSQL Systems, in: Yu, E., Dobbie, G., Jarke, M., Purao, S. (Eds.), Conceptual Modeling, Lecture Notes in Computer Science. Springer International Publishing, Cham, 223–231. https://doi.org/10.1007/978-3-319-12206-9_18

Cho, H.-J., Jou, Y.-J., Lan, C.-L., 2009. Time Dependent Origin-destination Estimation from Traffic Count without Prior Information. Netw Spat Econ 9, 145–170. https://doi.org/10.1007/s11067-008-9082-7

Ciferri, R.R., Salgado, A.C., Times, V.C., Nascimento, M.A., Magalhaes, G.C., 2003. A performance comparison among the traditional R-trees, the hilbert R-tree and the SR-tree, in: 23rd IEEE, 3–12. https://doi.org/10.1109/SCCC.2003.1245440

Coskun, B, 2020. YouTube-channel Pip-Videos: https://www.youtube.com/channel/UCWuoZtYYv3Vzu7U9rlevmyg (30 May 2020)

Coşkun, İ.B., Sertok, S., Anbaroğlu, B., 2019. K-Nearest Neighbour Query Performance Analyses On a Large Scale Taxi Dataset: Postgresql Vs. Mongodb. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. XLII-2/W13, 1531–1538. https://doi.org/10.5194/isprs-archives-XLII-2-W13-1531-2019

Coskun, (2020) 2020. Pip: https://github.com/bugracoskun/OD-Matrix (30 May 2020)

Deari, R., Zenuni, X., Ajdari, J., Ismaili, F., Raufi, B., 2018. Analysis And Comparision of Document-Based Databases with Relational Databases: MongoDB vs MySQL, in: 2018, IEEE, Varna, 1–4. https://doi.org/10.1109/InfoTech.2018.8510719

Garcia, J.C., Avendaño, A., Vaca, C., 2018. Where to go in Brooklyn: NYC Mobility Patterns from Taxi Rides, in: Rocha, Á., Adeli, H., Reis, L.P., Costanzo, S. (Eds.), 203–212. https://doi.org/10.1007/978-3-319-77703-0_20

Gebetsroither-Geringer, E., Stollnberger, R., Peters-Anders, J., 2018. Interactive Spatial Web-Applications as New Means of Support for Urban Decision-Making Processes, in: ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences. 59–66. https://doi.org/10.5194/isprs-annals-IV-4-W7-59-2018

Lederman, R., Wynter, L., 2011. Real-time traffic estimation using data expansion. Transportation Research Part B: 1062–1079. https://doi.org/10.1016/j.trb.2011.05.024

Li, D., Lin, Y., Zhao, X., Song, H., Zou, N., 2011. Estimating a Transit Passenger Trip Origin-Destination Matrix Using Automatic Fare Collection System, in: Xu, J., Yu, G., Zhou, S., Unland, R. (Eds.), 502–513. https://doi.org/10.1007/978-3-642-20244-5_48

Munizaga, M., Devillaine, F., Navarrete, C., Silva, D., 2014. Validating travel behavior estimated from smartcard data. Transportation Research Part C: Emerging Technologies 44, 70–79. https://doi.org/10.1016/j.trc.2014.03.008

Munizaga, M.A., Palma, C., 2012. Estimation of a disaggregate multimodal public transport Origin–Destination matrix from passive smartcard data from Santiago, Chile. Transportation Research Part C: Emerging Technologies 24, 9–18. https://doi.org/10.1016/j.trc.2012.01.007

Nguyen, T., 2009. Indexing PostGIS databases and spatial Query performance evaluations. International Journal of Geoinformatics 5, 1–9.

Peng, S., Sankaranarayanan, J., Samet, H., 2018. DOS: a spatial system offering extremely high-throughput road distance computations, in: 26th ACM SIGSPATIAL 199–208. https://doi.org/10.1145/3274895.3274898

Sveen, A.F., 2019. Efficient storage of heterogeneous geospatial data in spatial databases. J Big Data 6, 102. https://doi.org/10.1186/s40537-019-0262-8

TLC., 2019.TLC Trip Record Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page (30 May 2020)

Xian, X., Ye, H., Wang, X., Liu, K., 2020. Spatiotemporal Modeling and Real-Time Prediction of Origin-Destination Traffic Demand. Technometrics 1–13. https://doi.org/10.1080/00401706.2019.1704887

Zhang, J., You, S., Xia, Y., 2015. Prototyping A Web-based High-Performance Visual Analytics Platform for Origin-Destination Data: A Case study of NYC Taxi Trip Records, in: Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics - UrbanGIS'15. 16–23. https://doi.org/10.1145/2835022.2835025