UNIVERSITY OF THESSALY
School of Engineering
Electrical & Computer Engineering

Spatio-temporal and Spatio-textual Data Generator

by

Olympia Axelou

Supervising Professor: Michail Vassilakopoulos

Abstract

Human mobility modelling has attracted a lot of interest in recent years. From evaluating GeoSNs algorithms to using them as source data in what-if analysis. In all scenarios, the development of realistic spatio-temporal trajectories of human mobility is of fundamental importance. The need of synthetic datasets derives from the fact that the GeoSNs such as Foursquare and Facebook have privacy restrictions that prohibit the access to such real data in a large scale.

In this project, we present DataGen, a spatio-temporal and textual dataset generator. It operates in two steps: First it creates the profile of the 'virtual' user and then, based on possibilities and statistics, it creates his trajectories for a certain period of time (day, week, month). The dataset generated is stored as a json-per-line file and is open and available for further analysis.

Contents

Chapter 1

Introduction

The goal of generative algorithms of human mobility is to create a population whose mobility patterns are statistically indistinguishable from those of real individuals. Typically, each generative algorithm focuses on just a few properties of human mobility. A class of algorithms is mainly concerned with reproducing the trip distance distribution or the visitation frequency to a set of preferred locations (they mainly focus on realistically representing spatial properties). Another class of algorithms focus on the accurate representation of the time-varying behavior of individuals, relying on details schedules of human activities. Our work is based on the second class of algorithms.

However, the major challenge for generative algorithms is the creation of realistic temporal patterns, including the number and sequence of visited locations together with the time and duration of the visits. Building upon the above findings, many generative algorithms of human mobility have been proposed which try to reproduce the characteristic properties of human mobility trajectories [11] [6].

The two papers that are most close to to our data generator are DITRAS [12] and SPATEN [9]. DITRAS' algorithm operates in two steps: first it generates the mobility diary and then it creates the mobility trajectory. SPATEN's contribution was the Maps API used for the transportation from POI to POI and the static map of the generated daily trajectories.

In this work, we present DataGen, a configurable generator that can produce large amounts of spatio-temporal and textual data based on realistic user behavior. This work is divided into three parts: The first one was the collection of the Points of Interest (POIs) and their efficient storage. The second part was the algorithm for the user's profile creation and the third one was the generation of the dailty trajectories based on the user's profile from the second part and while using the POIs collected in the first part.

For the collection of the POIs, we used YELP [3], an all-purpose open dataset which includes businesses from all over USA and Canada. This dataset was stored in Neo4j [1], a graph database management system suitable for accessing spatial data.

The second part of the generator is to produce the trajectory diary of virtual users. The approach we decided to choose is the following: first it creates the profile of a virtual user (randomly chooses the gender, the age and the educational level) and then according to the chosen profile, it assigns specific probabilities and check-in durations (e.g. The possibility of a user to go to a restaurant and the time spent there). These numbers were extracted from the American Time Use Survey (2019) [4] which include the possibility that an American does a certain daily activity and the average time that he/she spends on it. Finally, it generates the trajectories for a number of days (given as parameter).

Finally, in the third part, the algorithm chooses the categories of activities that a user will do in that specific day (based on the possibilities mentioned in the previous part) and creates the trajectory of this user. It uses Google Directions API to compute the duration to get from one place to another and the Static Maps API to create the final map of the places that the user visited in one day.

Chapter 2

The Generator

2.1 Source Data and Preprocessing

DataGen uses real Points of Interest (POIs) as a source of data. These locations are extracted from the Yelp Dataset [3], a dataset with businesses from all over USA and Canada. The POIs are identified on the map by their geographical coordinates as well as their address. The number of such available points were 200,097.

The database we decided to user in order to store the points is a NoSQL Graph database since information about POIs interconnectivity or topology is as important as each POI itself [5] [13]. One of the main advantages of graph databases as a solution to our problem is that It allows for a more natural modeling of data. Graph structures are visible to the user and they allow a natural way of handling application data like geographic data. The points were stored in a NoSQL Neo4j Database [1], as its capabilities in terms of speed and ease of development are quite outstanding, outperforming many of its competitors [7] [10] [8].

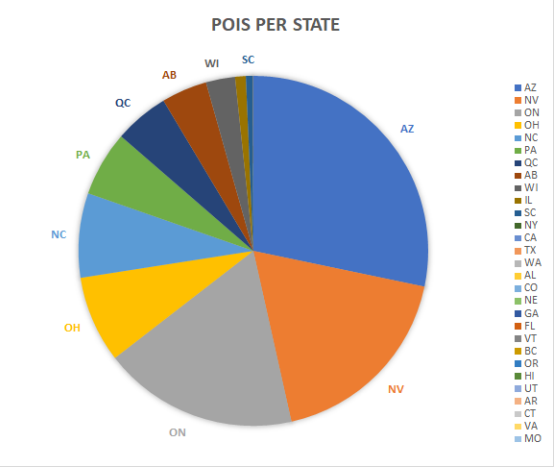

FIGURE 2.1: *States with the most POIs (descending order)*

For further optimization of the response time of the graph database, the POIs were clustered in regions, 8 in particular, separating the most populated states so that the regions are balanced. According to Figure 2.1, the 8 most populated states are Arizona (56592 POIs), Nevada (37452), Ontario (36061), Ohio (16040), North Carolina (15610), Pennsylvania (12042), Quebec (10129) and Alberta (8446). The rest states that form each region are selected by both distance from the nearest most populated state and by the number of POIs per state, so that we can form clusters of similar number of POIs. The final form of the regions is shown in table 2.1 and the visual form on Maps in figure 2.2.

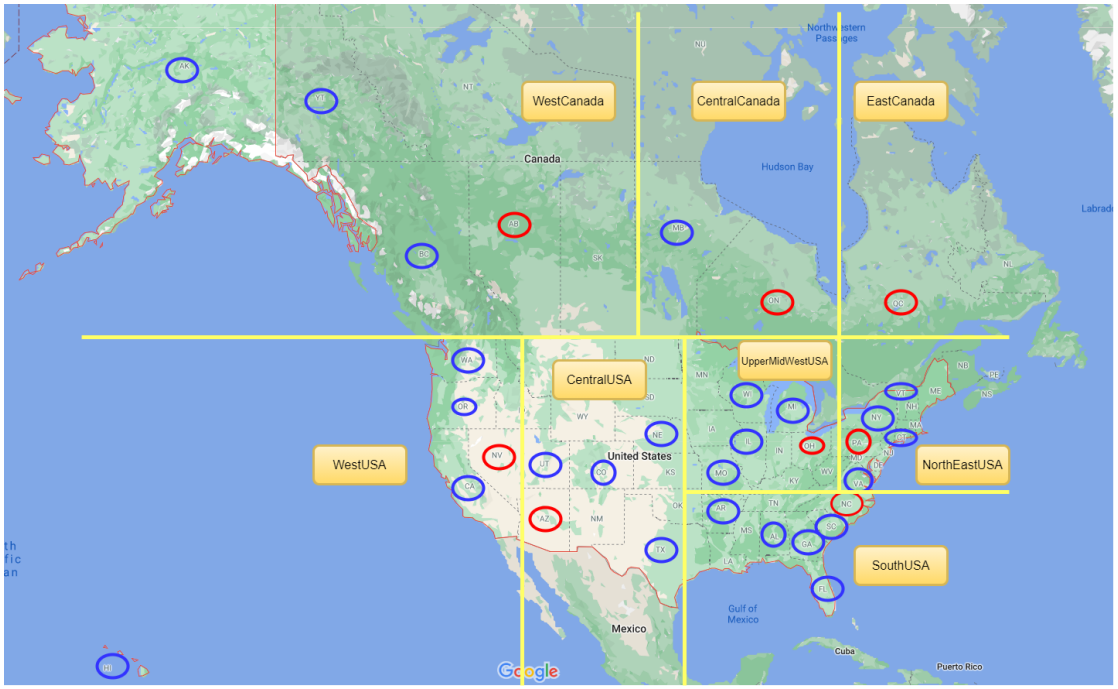

FIGURE 2.2: *Regions and States*

WestUSA		CentralUSA		SouthUSA		NorthEastUSA		UpperMidWestUSA		EastCanada		CentralCanada		WestCanada	
NV	36452	AZ	56592	NC	15610	PA	12042	OH	16040	QC	10129	ON	36061	AB	8446
WA	4	UT	1	SC	1297	NY	21	WI	5393			MB	1	BC	2
OR	1	CO	2	GA	2	VT	2	IL	1969					YT	1
CA	14	TX	6	AL	3	CT	1	MO	1					AK	1
HI	1	NE	2	FL	2	VA	1	MI	1						
				AR	1										
5	36,472	5	56,603	6	16,915	5	12,067	5	23,404	1	10129	2	36061	4	8450

TABLE 2.1: *Regions and States*

2.2 Graph DB and Python driver

After the preprocessing of the dataset has been completed, the data was inserted into the graph database. Graph Databases have some similarities with graphs, in a way that there's also the concept of *node*. A node here is a Point of Interest. In Figure 2.3 we can see all information of the POI stored in the database. Lastly, we created 8 different labels that characterize the nodes, one for each region. Figure 2.4 shows random POIs from different regions. The different labels

in the result of the query are displayed on the top of the screenshot. Some of these regions labels are WestUSA, CentralCanada, EastCanada, etc.

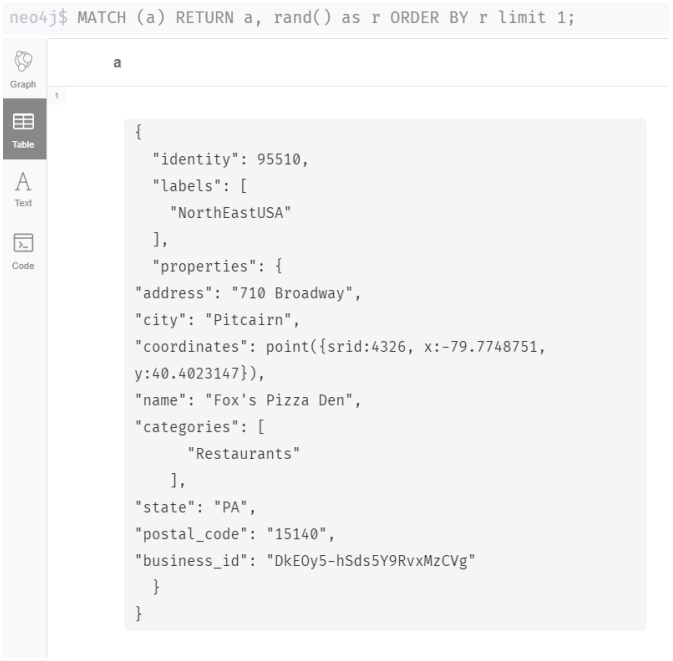

FIGURE 2.3: *POI-node Information*

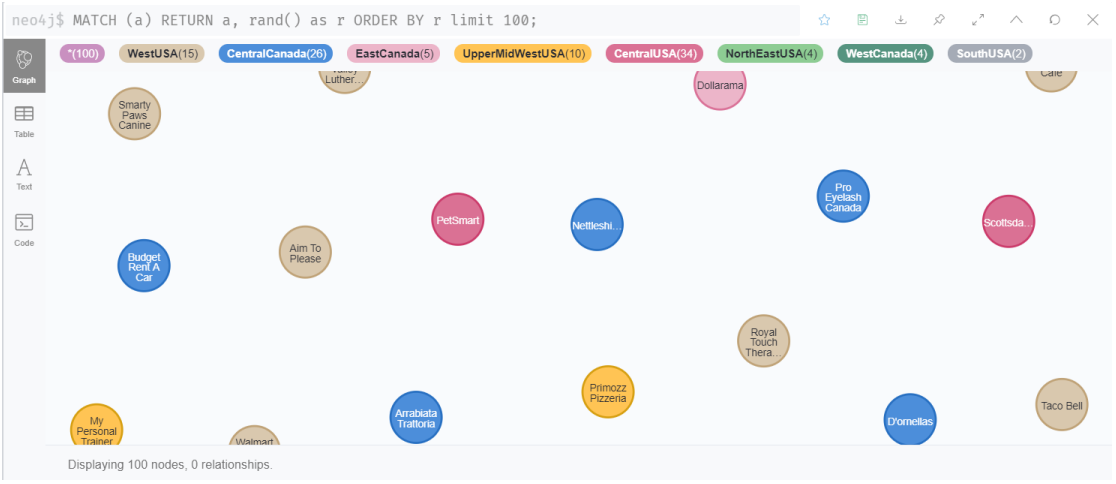

FIGURE 2.4: *Cypher random-nodes query*

As it is going to be explained later on, one of the design attributes of the generator is that there is a *home* POI for every virtual user. In addition to this, every POI that the user can visit must be in the same region as his home. So, with *specificRegion* being the region where *home* is and *maxRadius* a constant value that represents the maximum distance a user can travel from home to visit a POI, the cypher query used to find all POIs that a user can visit in a day is:

MATCH (n: *specificRegion*), (m: *specificRegion*)

WHERE n.businessID=*home['businessID']* AND n $<>$ m

AND DISTANCE(n.coodinates, m.coordinates) $<$ *maxRadius*

RETURN m;

Along with the above query that finds all POIs within the max radius from home, the generator also uses a query to find a random POI in a specific region to set it as *home*. However, because of its simplicity, this query is omitted.[1]

Lastly, since the main program is written in Python, the connection between Neo4j and Python is achieved with the official Neo4j Driver for Python [2].

[1]Code available at: `https://github.com/oaxelou/SpatioTempDataGenerator/`

Chapter 3

The main Algorithm

In this section, we are going to explain the design attributes of the data generator, which includes both parametrizable constansts and constants derived from statistical analysis syrveys. Also, we are going to present the full algorithm and the result data.

3.1 Algorithm's Overview

The generator creates virtual users – based on a randomly set profile – who check into places during the day for a given period of time. Each user is assigned a home. The type of place that the user visits serves as the textual aspect of the generator, the location of the place the spatial and the time and duration of the check-in represents the temporal aspect of the generator.

The decisions that the generator has to make, including the number of POIs visited per day, the duration of each check-in etc., are made using random configurable factors which are going to be explained later on in this section.

Another fundamental part of the generator is the path from one location to the next one. These paths are calculating using Google Directions API. Finally, the generator visualizes each user's daily routes with Google Static Maps API and stores the map as a photo which is added to the final dataset. An example of a Google Static Map image is shown in figure 3.1.

3.2 Design Attributes

The parameters of the generator are divided into two categories, the **constants** which are configurable parameters that are set by the user of the generator (e.g. maximum distance from home, the start time of the day) and the **variables** that are set randomly according to the virtual

FIGURE 3.1: *An example of a Google Static Map image*

user's profile created by the generator, e.g. check-in duration and the mean probability for a user to choose a location to visit. These parameters are going to be described thoroughly later on in this section.

The variable parameters set by the generator for each virtual user:

- **Home**: Every user is assigned a home POI. All the locations that he visits in a day are around his home and within a sensible distance from it. Specifically, a user can drive in a range of *maxDist* meters from his home, which is a constant parameter set by the user of generator.
- **Work**: After randomly setting the profile of the user (age, gender and educational level), the generator decides whether this user works or not based on possibilities according to his profile from the surveys mentioned above. If so, it also assigns an approximate work duration and it chooses the POI that is going to be the workplace. In the case that a person works, his workplace is the first POI that he visits every day.
- **categPossibilities** & **meamCheckInDur**: The first step of the generator is to create the profile of the virtual user, which means to randomly choose the gender, the age and the educational level. Then, according to the chosen profile, it assigns specific probabilities to each different POI and the mean check-in duration, e.g. the possibility of a user to go to a restaurant and the mean time spent there. When the generator has to set the exact duration of the check-in, this number is randomly generated with gaussian distribution, where mean is the *meanCheckInDur* and the standard deviation is the constant parameter *chStdDev*.

The rest of the constant parameters that are not mentioned above are:

- **startTime** & **endTime**: The input parameters that define respectively the time of the day when the first visit to a location begins (this includes the time spent driving) and

the time in the evening when a user must have returned home. The time when the next check-in occurs is the time when the previous one happened, plus the duration of the check-in, plus the duration of driving from the previous location to this one. So, if the time the next check-in exceeds the time set by *endTime* or the number of the next check-in exceeds the maximum check-in number, then the next check-in won't occur and this specific day's check-ins end at that time.

Time in general is represented as time delta in seconds from 00:00 (midnight). For example, the default value of the startTime parameter is 28800 which translates to 8 o'clock in the morning.

- **numberOfUsers** & **timePeriod**: These two input parameters describe the volume of the data generated. In default, the generator runs for 1000 users and in a time period of 60 days (2 months), however these parameters are customizable and can be changed.

Chapter 4

The Output Dataset

4.1 Output Data

The output dataset is divided into two parts: the Google Static Maps figures (png extenstion) and the text file with all users' information & trajectories. In particular, the last one is stored in JSON Lines text file format [1], aka JSON-per-line format. The final form of the dataset text file is as shown in figure 4.1. Because of Google Maps API's free trial limitations, we were only able to generate 2 months (60 days) of daily trajectories for 866 users. The total dataset is 2.31GB [2].

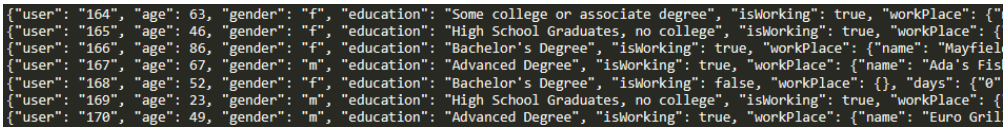

FIGURE 4.1: *Final form of the Dataset - Raw JSONLines file*

Each JSON record contains all data regarding one virtual user like his profile (age, gender, educational level) and work information like work duration and his workplace. It also contains the user's daily trajectories (duration of check-ins, transportation data, etc.) for the period of time set as parameter (default: 60days), as well as details on the POIs he visited (business name, coordinates, etc).

For example, the screenshot in figure 4.2 displays (with json pretty print) user number 602. First, it includes information on her profile like her age, gender and educational level. Then, it has information on her work, like that she works and then her workplace and working duration. After the user's information, there's the daily trajectories of the user for each day. In the particular example, we can see that in the 4th day, the user visited 3 places, first she went to work, then to a restaurant and final to a park.

[1] JSON Lines text file format: `https://jsonlines.org/`

[2] The generated dataset can be found here: `https://drive.google.com/drive/folders/17quHn23EpvuWRogM6XoLTHwkDVPu94Tj?usp=sharing`

As for the durations, she spent 1955 seconds (32 minutes) in transportation from her house to her work and 1999 (33 minutes) from the work to the restaurant. Finally, she went to the restaurant at the timestamp of 55615 seconds (which translates to 15:26:55 o'clock) and she ended the day at 21:03:07 o'clock, in other words at timestamp of 75787 seconds.


```
{
    "user": "602",
    "age": 26,
    "gender": "f",
    "education": "Bachelor's Degree",
    "isWorking": true,
    "workPlace": {
        "name": "Oasis Automotive",
        "business_id": "GlywIf9fWjeQIwYrIlHJjg",
        "state": "NV",
        "postal_code": "89074",
        "city": "Henderson",
        "address": "1251 American Pacific Dr",
        "coordinates": [-115.040901, 36.040616],
        "categories": ["Services"]
    },
    "workingDuration": 28944,
    "days": {
        "0": {"total_pois_visited": 5, "pois_visited": {"0": {"transport_duration": 1955, "transport_distance": 41523, "checkin": 307
        "1": {"total_pois_visited": 4, "pois_visited": {"0": {"transport_duration": 1955, "transport_distance": 41523, "checkin": 307
        "2": {"total_pois_visited": 5, "pois_visited": {"0": {"transport_duration": 1955, "transport_distance": 41523, "checkin": 307
        "3": {"total_pois_visited": 1, "pois_visited": {"0": {"transport_duration": 1955, "transport_distance": 41523, "checkin": 307
        "4": {
            "total_pois_visited": 3,
            "pois_visited": {
                "0": {"transport_duration": 1955, "transport_distance": 41523, "checkin": 30755, "checkout": 53616, "purpose": "Work
                "1": {
                    "transport_duration": 1999,
                    "transport_distance": 47976,
                    "checkin": 55615,
                    "checkout": 60373,
                    "purpose": "Restaurants",
                    "business_details": {
                        "name": "Taco Bell",
                        "business_id": "MTx-Zdl_KcU_z9G832XAjg",
                        "state": "NV",
                        "postal_code": "89131",
                        "city": "Las Vegas",
                        "address": "8033 N Durango Dr",
                        "coordinates": [-115.2790587, 36.3074953],
                        "categories": ["Restaurants"]
                    }
                },
                "2": {"transport_duration": 1712, "transport_distance": 28148, "checkin": 62085, "checkout": 75787, "purpose": "Parks
            }
        },
        "5": {"total_pois_visited": 4, "pois_visited": {"0": {"transport_duration": 1955, "transport_distance": 41523, "checkin": 307
        "6": {"total_pois_visited": 3, "pois_visited": {"0": {"transport_duration": 1955, "transport_distance": 41523, "checkin": 307
```


FIGURE 4.2: *Final form of dataset - Random User*

4.2 Statistical Analysis on the dataset

The generator receives as input the possibility for a user to choose an activity and the mean duration of this activity which values have been extracted from surveys. Along with that, the generator also receives as parameter the standard deviation of the duration of each check-in, which is fixed for every type of activity and the maximum number of check-ins per day which is 10 by default.

As a result, the number of check-ins per day differs depending on the person's profile. As observed by a sample of 120,000 days (200 Users x 60 Days), the number of check-ins per day follows a normal distribution with mean value 4.7 check-ins/day and standard deviation 2.05. Figure 4.3 shows the normal distribution of the number of check-ins.

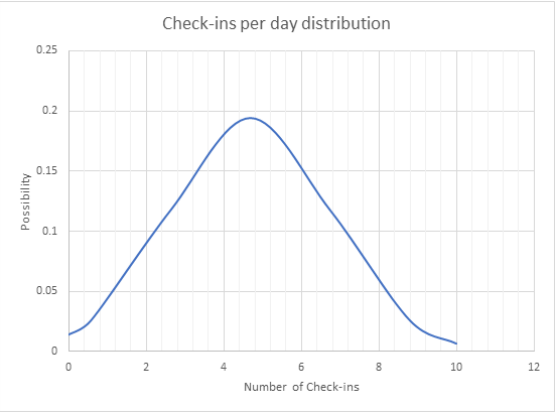

FIGURE 4.3: *Number of Check-ins per day*

Another statistical analysis on the final dataset worth mentioning is the duration of the check-ins. As it is mentioned before, the duration of each activity defers from activity to activity and also depends on the user's profile. There may be large differences in the duration, as observed in the statistical analysis on the duration of check-ins. It was found that it follows a normal distribution with mean duration of a check-in being 2.64 hours and the standard deviation 2.05 hours. The distribution of the check-in's duration as shown in figure 4.4.

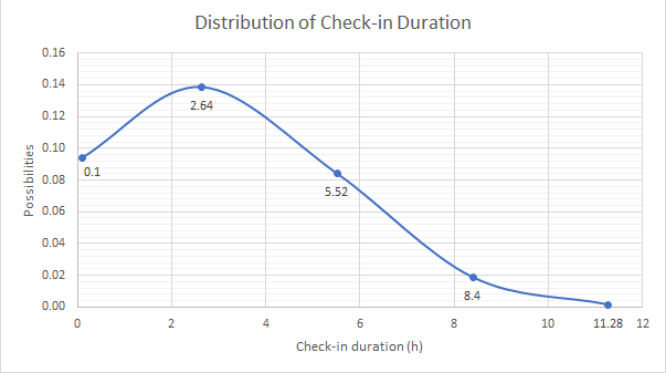

FIGURE 4.4: *Duration of Check-ins*

4.3 Generator's time performance

We performed a scalability evaluation by testing the generator with different number of users while all other parameters of the generator remained unchanged. In particular, for 10, 100, 1000 and 10000 users, which means that the growth of the number of users is exponential. As we can see in figure 4.5, the execution time is proportional to the exponential increase of the number of users. This $O(n)$ complexity is very satisfactory; however, time optimizations can be applied for better performance.

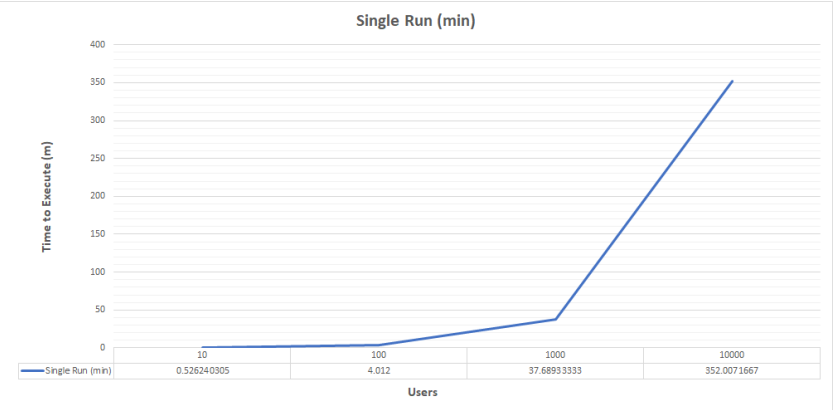

FIGURE 4.5: *Execution Time*

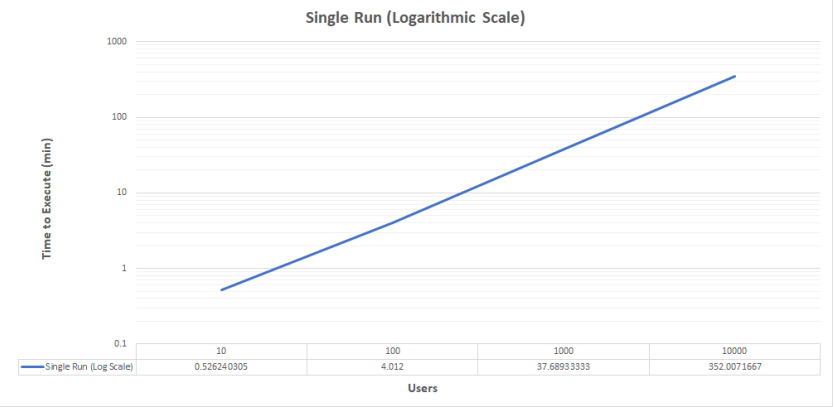

FIGURE 4.6: *Execution Time - Logarithmic Scale*

As mentioned above, the generator ran for approximately 1,000 users. Since the execution time for this number of users was approximately 3.5 hours[3], we ran some experiments with multiple processes sharing the load. According to figure 4.7, the execution time begins to converge to approximately 0.3 hours when the data generator is executed by 5 processes in parallel. So, the suggested number of processes to execute the generator's code is 5.

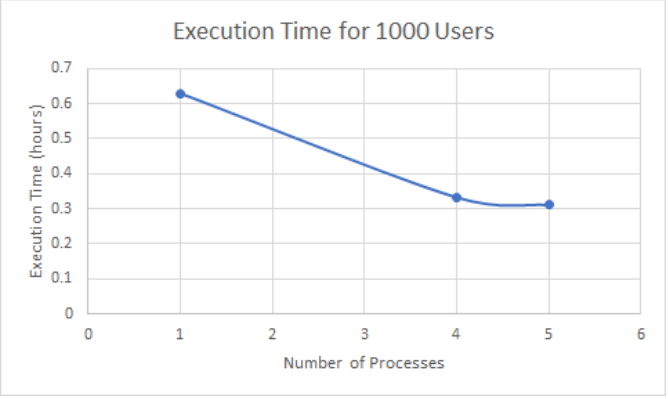

FIGURE 4.7: *Execution Time - Multiple Processes*

[3]PC's specifications: Intel Core i7-7500U 2.70GHz, 8GB RAM, NVIDIA GeForce 940MX GPU

Chapter 5

Conclusions

We proposed DataGen, a configurable spatio-textual and spatio-temporal data generator. First, we collected POIs from the YELP dataset, a dataset about businesses in the USA and Canada, and inserted them in a Neo4j Graph Database. Then we researched and collected data from surveys regarding the tendencies of people. We processed the data and got the possibilities of doing an activity and the duration of this activity for different profiles of users. Then, the data generator created randomly (in a uniform way) the users' profiles and produced the trajectories.

The resulted dataset is our contribution to the scientific society. Since, real big data is hard to find because of the privacy restrictions of GeoSocial Networks, the synthetic datasets are a good alternative. Our open dataset can be used in Data Science, like Machine Learning that needs large datasets for training and testing and for spatio-textual and spatio-temporal algorithms' evaluation.

As long as future work is concerned, the instantiation of the data generator we propose can be further improved in several directions. First, the selection of the next place for the user to visit, instead of choosing it randomly (with the gaussian distribution), it can be desgined using Markov Models.

Second, another nice addition, inspired by Spaten Dataset [9], would be to consider travelling for the user. In this scenario, the user's home would be set to a 'travel home', such as a hotel etc., and the rest after that would be the same as the existing code. The generator would then have to make a choice whether the user will travel or not and how many days this travel will last.

Finally, the usage of Google Directions API could be improved in some ways. Route handling could be added so that the final dataset would also have the detailed route that the virtual user did from a place to another. Also, the feature of the multiple ways of transporting that the Google Directions API offer could be used.

Bibliography

[1] 2010. Neo4j: Graph Database System Management. https://neo4j.com.

[2] 2010. The Official Neo4j Driver for Python. https://neo4j.com/docs/api/python-driver/current/.

[3] 2014. YELP database. https://www.yelp.com/dataset.

[4] 2019. American Time Use Survey of 2019. https://www.bls.gov/news.release/pdf/atus.pdf.

[5] Renzo Angles and Claudio Gutierrez. 2008. Survey of Graph Database Models. *ACM Comput. Surv.* 40, 1, Article 1 (Feb. 2008), 39 pages. https://doi.org/10.1145/1322432.1322433

[6] Hugo Barbosa, Marc Barthelemy, Gourab Ghoshal, Charlotte R. James, Maxime Lenormand, Thomas Louail, Ronaldo Menezes, José J. Ramasco, Filippo Simini, and Marcello Tomasini. 2018. Human mobility: Models and applications. *Physics Reports* 734 (2018), 1 – 74. https://doi.org/10.1016/j.physrep.2018.01.001 Human mobility: Models and applications.

[7] Sotirios Beis, Symeon Papadopoulos, and Ioannis Kompatsiaris. 2015. *Benchmarking Graph Databases on the Problem of Community Detection.* Vol. 312. 3–14. https://doi.org/10.1007/978-3-319-10518-51

[8] Felix Dietze, Johannes Karoff, André Calero Valdez, Martina Ziefle, Christoph Greven, and Ulrik Schroeder. 2016. An Open-Source Object-Graph-Mapping Framework for Neo4j and Scala: Renesca, Vol. 9817. 204–218. https://doi.org/10.1007/978-3-319-45507-514

[9] Thaleia Doudali, Ioannis Konstantinou, and Nectarios Koziris. 2017. Spaten: A spatio-temporal and textual big data generator. 3416–3421. https://doi.org/10.1109/BigData.2017.8258327

[10] S. Jouili and V. Vansteenberghe. 2013. An Empirical Comparison of Graph Databases. In *2013 International Conference on Social Computing*. 708–715. https://doi.org/10.1109/SocialCom.2013.106

[11] D. Karamshuk, C. Boldrini, M. Conti, and A. Passarella. 2011. Human mobility models for opportunistic networks. *IEEE Communications Magazine* 49, 12 (2011), 157–165. https://doi.org/10.1109/MCOM.2011.6094021

[12] Luca Pappalardo and Filippo Simini. 2018. Data-driven generation of spatio-temporal routines in human mobility. *Data Mining and Knowledge Discovery* 32 (05 2018). https://doi.org/10.1007/s10618-017-0548-4

[13] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. 2010. A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective. In *Proceedings of the 48th Annual Southeast Regional Conference* (Oxford, Mississippi) *(ACM SE '10)*. Association for Computing Machinery, New York, NY, USA, Article 42, 6 pages. https://doi.org/10.1145/1900008.1900067