# Concerns on Design and Performance of a Local and Global Dynamic Map

Itziar Urbieta, Irati Mendikute, Harbil Arregui, Oihana Otaegui

Vicomtech, Mikeletegi 57, 20009 Donostia (Spain)

**Abstract.** Current real-time data collection systems for urban transportation and mobility allow enhancing digital maps with up-to-date situational information. This information is of great interest for short-term navigation and route planning as well as for medium- to long-term mobility data analysis, as it provides a finer time-varying detail of the urban movement infrastructure. In this work, we present our ongoing work to design a representation of a unique urban movement space graph as a local and global dynamic map approach. We address the concerns that must be considered when handling different scales of geographic areas inside a city, according to the application.

**Keywords.** Dynamic map, Spatial, Graph databases

## 1. Introduction

Digital representations of both transport networks and movement data collected by any sensing technique need to be stored in efficient electronic formats if we aim to use them for navigation or short-term mobility planning purposes. How they are stored and indexed, in fact, will impact the read and write access times, as well as the performance of spatial or temporal joins of any kind.

Digital maps of several levels of detail are used to represent the city infrastructure devoted to transportation purposes. Current extended sensing and data collection techniques allow measuring and reporting the real demand of this infrastructure and what is happening at large scale. These digital maps can be of interest from a first-person view, locally (the moving vehicle, passenger or citizen), or from a third person view, globally (a city mobility manager or a transportation planner authority).

Published in "Adjunct Proceedings of the 15th International Conference on Location Based Services (LBS 2019)", edited by Georg Gartner and Haosheng Huang, LBS 2019, 11–13 November 2019, Vienna, Austria.

This contribution underwent double-blind peer review based on the paper. https://doi.org/10.34726/lbs2019.42 | © Authors 2019. CC BY 4.0 License.

Several works have addressed the computational challenges of storing and querying records that represent locations, movements and trajectories, like Evans et al. (2013). Research on the concept of a Local Dynamic Map (LDM) has been mainly oriented to automotive navigational purposes, e.g., Shimada et al. (2015) and Eiter et al. (2019). The purpose of LDM is to provide the nearest location information to moving agents as they move. The most known approach consists of four different layers, from the most static to the most dynamic. To our knowledge, no other authors have tried to use the same map for a Global use. Thus, we present the concept of the Local and Global Dynamic Map (LGDM).

## 2. Problem statement and research objectives

We envision that the extension of the same Dynamic Map to Global applications may feed high level transportation planning purposes. The main challenge that arises when considering the LGDM is scalability. Thus, we find the need to evaluate the response time and graph behavior when it comes to perform basic operations both locally and globally. For evaluation, we choose the geometric map-matching as one of the most frequently used basic operations. This operation associates a vehicle position to an element on the digital transportation infrastructure map. Being aware of the limitations will allow creating a graph model with optimum performance for global and local analysis.

The main research objectives of the present on-going work are two-fold. First, we aim to define a consistent graph model to represent the urban road infrastructure as a time-varying dynamic map for local and global use. Secondly, we seek to evaluate the implications of the size of the graph (in terms of number of nodes) when computing the needed operations so as to develop the most suitable graph partitioning method. These main objectives are built upon some partial objectives: the study of options to build the network from raw OpenStreetMap (OSM) data (OpenStreetMap contributors 2018) and the integration between a real-time feed of vehicle locations and the network map.

## 3. Methodology and preliminary results

In our research, we have chosen Neo4j as a graph database management system for its lower learning curve. For an overview of different graph data base frameworks, we refer the reader to work published by Kumar Kaliyar (2015). We have chosen OSM as the transportation network data source, and the SUMO simulation framework, (Lopez et al. 2018), for creating synthetic vehicle movement data.

The transportation infrastructure network is a directed graph, represented by a set of nodes and vertices, associated by a collection of numerical and categorical properties. To import this data from OSM, we have evaluated different alternatives.

- **Osm2po**, (Moeller 2018), is a converter and routing engine. It allows filtering from OSM attributes to build a routable network. Then, we import its output manually into Neo4j. We have tested two variants: representing ways as nodes or as relations. The first variant adds complexity to the network since the number of nodes is significantly higher, but it is more flexible when querying the graph.
- **Spatial plugin for Neo4j**. This option avoids the need to pre-process the OSM file. The spatial functions it provides make the process easier and faster. However, the way of representing the information has been found less intuitive. In addition, intersections are not easily identified, thus limiting some route analysis.
- Additional tools such as https://github.com/neo4j-contrib/osm

In order to simulate the generation of vehicle positioning traces that can be matched onto the network we use the SUMO traffic microsimulation tool. The output of SUMO is a Floating Car Data (FCD) xml file. The TRaCI interface enables the access to a running simulation and allows reading all variables describing vehicle behavior online from our Python code. The most valuable information obtained is the location of a given vehicle at each simulation step.

Afterwards, we need to store this data into the Neo4j graph database where the OSM network has been previously loaded. After evaluating some different drivers to interact with Neo4j from our Python code, we chose the official Neo4j Python driver. At each simulation step, a reference vehicle is taken as the *ego-vehicle*. Step by step, a new node is created at each time instant in Neo4j, adding all the attributes that the TRaCI commands extract. Then, we apply a geometric map-matching, which consists of assigning the network node closest to the vehicle position by creating a new relation (*LOCATED_IN*) between the nodes in Neo4j.

TraCI slows down the simulation speed and there are several factors that condition its performance such as 1) the number of TraCI function calls per simulation step; 2) the types of TraCI functions being called; 3) the computation within the TraCI script; and 4) client language. Moreover, there are also other estimations related to the response time and the client language already given in the documentation.

To make the evaluation and compare different cases of the complexity of the matching process according to the size of the network, we record: the initial, final and middle simulation step times and the average step duration, as well as the total duration of the simulation and the number of elements in the database. Two simulation-related figures are also given: real-time factor and updates per second (UPS). Real time factor represents the relation *simulated time / computation time*. UPS (updates per second) denotes the number of vehicles that were simulated on average per second of computation time. Two different cases have been evaluated simulating 100 steps with a step-length of 0.1s, in 7 executions each, linearly increasing the number of nodes in the database, thus enlarging the geographical area considered:

**Case 1: Response time of the simulation with map-matching of vehicle.** In this case, for the evaluation of the response time of the simulation, the first approach has been simulating several times the script with the complete simulation. This way, the number of elements in the database increased linearly since the number of nodes was every time the same (Table 1).

| Trial | Nodes | Relations | Duration [ms] | Real time factor | UPS | Average step [s] | Final step [s] | Initial step [s] | Middle step [s] |
|---|---|---|---|---|---|---|---|---|---|
| A | 263447 | 284412 | 65760 | 1.55 | 74.55 | 0.64 | 0.68 | 0.78 | 0.65 |
| B | 263543 | 284415 | 111684 | 0.91 | 43.90 | 1.11 | 1.12 | 1.34 | 1.08 |
| C | 263639 | 284418 | 152837 | 0.66 | 32.07 | 1.54 | 1.58 | 1.70 | 1.60 |
| D | 263735 | 284421 | 196572 | 0.51 | 24.94 | 2.00 | 2.02 | 2.11 | 2.00 |
| E | 263831 | 284424 | 240646 | 0.42 | 20.37 | 2.46 | 2.47 | 2.58 | 2.50 |
| F | 263927 | 284427 | 283675 | 0.35 | 17.28 | 2.91 | 2.96 | 2.94 | 2.88 |
| G | 264023 | 284430 | 330335 | 0.30 | 14.84 | 3.40 | 3.38 | 3.52 | 3.42 |
| H | 264119 | 284433 | 14779 | 6.90 | 331.75 | 0.12 | 0.11 | 0.19 | 0.11 |

*H = without map-matching

**Table 1.** Measurements obtained in case 1 with 100 steps simulation and step-length of 0.1s.

**Case 2: Response time evaluation with map-matching and first approach to build an egocentric network.** The next evaluated case has been the simulation with the map-matching considering the distance between nodes and the first approach defined to build an egocentric network. Once again, the simulation script has been simulated several times to increase gradually the elements in the database. As an initial method to represent an ego-centered graph, the vehicle with 'id=0' has been chosen to characterize the Ego-car and new relations have been generated. First, the *ego-vehicle* has been related to the cartography (*EGO_LOCATED*) with the same map-matching approach. Then, to analyze the vehicles that are around the ego-car a new *NEAR* relation has been created considering the distance that exists between them in each time step (Table 2).

| Trial | Nodes | Relations | Duration [ms] | Real time factor | UPS | Average step [s] | Final step [s] | Initial step [s] | Middle step [s] |
|---|---|---|---|---|---|---|---|---|---|
| A | 263447 | 284412 | 131822 | 0.77 | 37.19 | 1.34 | 1.29 | 1.41 | 1.32 |
| B | 263639 | 284482 | 219489 | 0.46 | 22.33 | 2.25 | 2.24 | 2.31 | 2.28 |
| C | 263831 | 284552 | 305190 | 0.33 | 16.06 | 3.14 | 3.13 | 3.24 | 3.14 |
| D | 264023 | 284622 | 395011 | 0.25 | 12.41 | 4.08 | 4.37 | 4.21 | 4.06 |
| E | 264215 | 284692 | 481976 | 0.21 | 10.17 | 4.99 | 5.01 | 5.13 | 4.98 |
| F | 264407 | 284762 | 562707 | 0.18 | 8.71 | 5.81 | 5.83 | 5.93 | 5.77 |
| G | 264599 | 284832 | 660792 | 0.15 | 7.41 | 6.85 | 6.66 | 6.78 | 6.75 |

**Table 2.** Measurements obtained in case 2 with 100 steps simulation and step-length 0.1s.

## 4. Conclusions

Several conclusions have been obtained so far: on the one hand, the time needed for each step increases with the number of elements in the database. On the other hand, the initial step duration is, in general, bigger than the others. Also, in the first case, the simulation has been executed an extra time but without the map-matching approach (just creating a new node for each step). So, this was the case where the biggest number of elements was in the database. However, the response time decreased considerably in both cases. For instance, comparing the values obtained for this execution case (Trial H) and the first one (Trial A), can be seen that the time values are in general 5 times smaller. Besides, it should be highlighted that this is the best case since, if compared with the other trials, this difference increases. Considering how the map-matching method affects the response time of the simulation, it is still pending to make tests with other methods and compare the response times.

Finally, considering the second simulation case evaluated, as expected, the response time in general increased. This is both affected by the increase in the number of elements and the extra queries added to create the relations.

As a future work, both simulation cases will be tested with different geospatial databases to analyze and compare the graph considering the size and performance of the system. These preliminary conclusions will be extended to decide optimal network partition strategies according to response-time requirements for real-time applications of the LGDM.

## References

Eiter, T., Fureder, H., Kasslatter, F., Parreira, J. X., and Schneider, P. (2019). Towards a semantically enriched local dynamic map. *International Journal of Intelligent Transportation Systems Research*, 17(1):32–48.

Evans, M. R., Oliver, D., Shekhar, S., and Harvey, F. (2013). Fast and exact network trajectory similarity computation: A case-study on bicycle corridor planning. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing*, UrbComp '13, pages 9:1–9:8, New York, NY, USA. ACM.

Kumar Kaliyar, R. (2015). Graph databases: A survey. *In International Conference on Computing, Communication Automation*, pages 785–790.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flotterod, Y.-P., Hilbrich, R., Lucken, L., Rummel, J., Wagner, P., and Wießner, E. (2018). Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE.

Moeller, C. (2018). Osm2po: OpenStreetMap converter and routing engine for java. *URL: http://osm2po.de.*

OpenStreetMap Contributors (2018). Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org

Shimada, H., Yamaguchi, A., Takada, H., and Sato, K. (2015). Implementation and evaluation of local dynamic map in safety driving systems. *Journal of Transportation Technologies*, 5(02):102.