PoDiGG: A Public Transport RDF Dataset Generator

Ruben Taelman Ruben Verborgh
Tom De Nies Erik Mannens

Ghent University – imec – IDLab, Belgium
{firstname.lastname}@ugent.be

ABSTRACT

A large amount of public transport data is made available by many different providers, which makes RDF a great method for integrating these datasets. Furthermore, this type of data provides a great source of information that combines both geospatial and temporal data. These aspects are currently undertested in RDF data management systems, because of the limited availability of realistic input datasets. In order to bring public transport data to the world of benchmarking, we need to be able to create synthetic variants of this data. In this paper, we introduce a dataset generator with the capability to create realistic public transport data. This dataset generator, and the ability to configure it on different levels, makes it easier to use public transport data for benchmarking with great flexibility.

Keywords

Public Transport, Dataset Generator, Benchmark, Linked Data, RDF

1. INTRODUCTION

The effectiveness and efficiency of Linked Data and RDF data management systems are mostly measurement using benchmarks. Benchmarks usually require a real-world or synthetic input dataset to see how well systems handle and process this data. A major drawback of individual real-world datasets is their limited variety. In order to properly test the limitations of systems, it must be possible to introduce variations into the dataset, for example, making the dataset ten times larger. These variations must however still result in realistic datasets, which share characteristics with real data [6].

Public transport data is an especially interesting source of data, since it is made available by many different data providers, and it contains characteristics on several levels such as geospatial stops and temporal connections. These properties can uncover strengths or weaknesses of certain systems through benchmarking. Furthermore, the interlinked, multi-leveled nature of public transport data makes it particularly interesting for its representation as Linked Data.

In order to be able to measure the performance of RDF data management systems in the public transport domain, we introduce a mimicking algorithm that is able to generate public transport data. This generator is inspired by real-world transit network design and scheduling methodologies, in order to mimic realistic public transport data at different levels that can be configured using various parameters.

2. RELATED WORK

The de-facto standard for public transport schedule data is GTFS[1], in which transit feeds are represented by a series of CSV files with specific columns contained in a ZIP file. GTFS uses the following terminology for transit schedules:

Stops are geospatial locations where vehicles halt and passengers can get on or off.

Routes contain a sequence of stops.

Trips are serviced routes that are instantiated by time.

In order to make the publication of such data in RDF scalable for public route planning, *Linked Connections* [4] was introduced, where a *connection* is a hop from one stop to another in a trip, and is the primary element to represent public transport schedules. We use the Linked Connections methodology to serialize our generated datasets, because of its inherently linked structure, which makes it ideal for RDF datasets.

The process of public transit network planning is typically categorized in five sequential steps [2, 7]: (1) design of routes (2) setting of frequencies (3) development of timetables (4) vehicle scheduling and (5) driver scheduling. The first step is usually referred to as the *strategic* step of transit network planning, while steps 2 and 3 are referred to as the *tactical* steps. In this work, we are only concerned about the first three steps because vehicle and driver scheduling information is not needed for travelers.

3. GENERATOR

We assume a correlation between the population distribution of a region and its transit network design and scheduling: we expect that more populated areas will have easier access to public transport, and that those areas will also have more frequent connections to other places. In reality, the public demand for populated areas is a factor that is always taken into account in transit network design methodologies [7]. In fact, when we look at the population distribution of Belgium and the Netherlands and compare this with the distribution of train stops, we measure a positive correlation[2] of 0.439 for Belgium and 0.442 for the Netherlands.

We use the population distribution of a region as input to our generator, and create a transit network and its scheduling in four steps: (1) the placement of stops, (2) connecting stops using edges, (3) generation of routes and (4) scheduling of timely trips over

[1] https://developers.google.com/transit/gtfs/
[2] p-values in both cases < 0.00001

routes Finally, all this data is serialized as RDF using the Linked Connections[3] and GTFS[4] ontologies.

While real-world public transit planning methodologies assume stops and paths between them to be part of the input to the process, our algorithm also includes these as part of the generation process. Furthermore, our algorithm does currently not consider the vehicle and driver scheduling to be part of the generation process.

3.1 Stops Generator

The goal of the first step is to generate a realistic placement of stops in a two-dimensional area subdivided in discrete cells of equal size. For a preconfigured number of stops, we iteratively tag random cells as stops, where each stop is given a certain size based on its population value. This random selection is based on a Zipf-distribution where cells with a higher population value have a higher chance to be tagged as a stop than cells with a low population value. This distribution can be scaled to select minimum and maximum population values at which to tag stops.

3.2 Edges Generator

After stops have been generated, edges are created in order to form paths between stops. This generator will always create one connected transit network graph, where all stops are reachable from all other stops by any given path consisting of edges. In order to do this, the generator consists of two clustering phases and one so-called "loose stops"-phase.

The first agglomerative hierarchical clustering phase first considers all stops to be part of a different cluster. Each cluster always has a *center* point representing the average location of all stops in that cluster. This phase loops until all clusters have an inter-cluster distance larger than a preconfigured value. These distances are always calculated using Euclidian distances of the cells in their two-dimensional area.

After this first step, we have clusters of stops that lie close to each other. In the next step, we consider border stops, similar to *border stations* [1], as the only stops in a cluster that can be used to transfer to (border) stops in other clusters. This step ensures that all clusters eventually form one connected network.

The final step aims to resolve the problem of *loose stops*, which means that the two previous steps result in a significant amount of stops that are connected with only one other stop, i.e., they have a degree of one, which does not occur frequently in reality. This step aims to resolve this problem by detecting these loose stops, searching for other stops in the opposite direction of the single present edge, and adding an edge for the first stop that can be found in a given area.

3.3 Routes Generator

Once we have a network of stops connected by edges, we can start the placement of routes over this network. In order to simulate long and short distance routes, this generation phase is divided into two steps: first we create long routes connecting large stops, and after that, we create smaller routes connecting smaller stops.

For the first step, we create a list of the largest stops, where the size of this list is configurable. For each of these largest stops, the shortest path in the network to all other largest stops is calculated using the A* search algorithm and instantiated as a route. All sufficiently large stops on that route are also included as stops for that route.

Next, smaller routes are generated by iteratively selecting two random large stations, connecting them through a heuristically-determined shortest path, and including all passed stops in the route. This is done until a preconfigured number of routes is generated.

[3]http://semweb.mmlab.be/ns/linkedconnections

[4]http://vocab.gtfs.org/terms

3.4 Trips Generator

Finally, we generate trips based on the created routes. This is done by continuously picking a random route, with a larger chance on longer routes, and instantiating that route in time as a trip. This is done until a pre-configured number of trips has been created.

The instantiation of a trip is done by choosing a random starting time of the trip, based on a preconfigured time distribution. The default time distribution is based on the logs of the route planning API (iRail[5]) in Belgium [3]. The stop times are calculated for each stop in the trip by estimating the time it takes for a train to go from one stop to the next, including preconfigured speedup times, maximum vehicle speeds and required waiting times at stops.

4. CONCLUSIONS

By taking into account the strategic and tactical steps within transit network planning, and combining it with the population distribution of an area, we were able to create a realistic public transport dataset generator. PODIGG is implementation[6] of this algorithm, and is available under an open license.

In order to determine the realism of our mimicking algorithm, we aim to measure and improve the realism [6] of generated datasets in future work. Furthermore, this generator provides a basis for more extensive public transport data generation, for example mimicking the vehicle and driver scheduling in transport networks. Finally, this system can also be extended to serve as input to RDF stream processing systems [5], where the original transit schedules are used as static background knowledge, and real-time vehicle delays can be represented as RDF data streams.

The proposed dataset generator, and its flexible configurability, makes it easier for benchmarking RDF data management systems with realistic input datasets of different sizes at different interlinked levels, including the temporal and spatial dimension.

Acknowledgements

This work was funded by the H2020 project HOBBIT (#688227).

5. REFERENCES

[1] H. Bast, M. Hertel, and S. Storandt. Scalable transfer patterns.

[2] A. Ceder and N. H. Wilson. Bus network design. *Transportation Research Part B: Methodological*, 20(4):331–344, 1986.

[3] P. Colpaert, A. Chua, R. Verborgh, E. Mannens, R. Van de Walle, and A. Vande Moere. What public transit API logs tell us about travel flows. In *Proceedings of the 6th USEWOD Workshop on Usage Analysis and the Web of Data*, pages 873–878, Apr. 2016.

[4] P. Colpaert, A. Llaves, R. Verborgh, O. Corcho, E. Mannens, and R. Van de Walle. Intermodal public transit routing using Linked Connections. In *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*, 2015.

[5] E. Della Valle, S. Ceri, F. van Harmelen, and D. Fensel. It's a streaming world! Reasoning upon rapidly changing information. *Intelligent Systems, IEEE*, 24(6), Nov 2009.

[6] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 145–156. ACM, 2011.

[7] V. Guihaire and J.-K. Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, 2008.

[5]https://hello.irail.be

[6]https://github.com/PoDiGG/podigg-lc