Generating Public Transport Data for the Web based on Population Distributions

Ruben Taelman, Pieter Colpaert, Ruben Verborgh and Erik Mannens
imec - Ghent University - IDLab, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium
E-mail: ruben.taelman@ugent.be

Abstract
Applying Linked Data technologies to geospatial and temporal data introduces many new challenges, such as Web-scale storage, management, and the transmission of potentially large amounts of data. Several benchmarks have been introduced to evaluate the efficiency of systems that aim to solve such problems. Unfortunately, the synthetic data many of these benchmarks work with have only limited realism, raising questions about the generalizability of benchmark results to real-world scenarios. On the other hand, real-world datasets cannot be configured as freely, and often cover only certain aspects. In order to benchmark geospatial and temporal RDF data management systems with sufficient external validity and depth, we designed poDiGG, a highly configurable generation algorithm for synthetic datasets with realistic geospatial and temporal characteristics comparable to those of their real-world variants. The algorithm is inspired by real-world public transit network design and scheduling methodologies. This article discusses the design and implementation of poDiGG and validates the properties of its generated datasets. Our findings show that the generator achieves a sufficient level of realism, based on the existing coherence metric and new metrics we introduce specifically for the public transport domain. Thereby, poDiGG provides a flexible foundation for benchmarking RDF data management systems with geospatial and temporal data.

Keywords: Public Transport, Dataset Generator, Benchmarking, RDF, Linked Data

1. Introduction

The Resource Description Framework (RDF) and Linked Data technologies enable distributed use and management of semantic data models [9,4]. Datasets with an interoperable domain model can be stored and queried by different data owners in different ways. In order to discover the strengths and weaknesses of different storage and querying possibilities, data-driven benchmarks with different sizes of datasets and varying characteristics can be used.

Regardless of whether existing data-driven benchmarks use real or synthetic datasets, the external validity of their results can be too limited, which makes a generalization to other datasets difficult. Real datasets, on the one hand, are often only scarcely available for testing, and only cover very specific scenarios, such that not all aspects of systems can be assessed. Synthetic datasets, on the other hand, are typically generated by mimicking algorithms [5,18,25,26], which are not always sufficiently realistic [4]. Features that are relevant for real-world datasets may not be tested. As such, conclusions drawn from existing benchmarks do not always apply to the envisioned real-world scenarios. One way to get the best of both worlds is to design mimicking algorithms that generate realistic synthetic datasets.

The area of geospatial [15] and temporal [11,9] benchmarking for RDF storage and querying systems provides no mimicking algorithms for generating realistic synthetic dataset with geospatial and temporal characteristics. There is a need for benchmarks that evaluate systems handling datasets with geospatial and temporal characteristics, but it is important that the datasets they use are realistic.

The public transport domain provides data with both geospatial and temporal properties, which makes this an especially interesting source of data for benchmarking. Its representation as Linked Data is valuable because i) of the many shared entities, such as stops,
routes and trips, across different existing datasets on the Web. ii) These entities can be distributed over different datasets and iii) benefit from interlinking for the improvement of discoverability. Synthetic public transport datasets are particularly important and needed in cases where for example geospatial systems or public transport route planning algorithms are evaluated. The Linked Connections framework [8] and Connection Scan Algorithm [12] are examples of such public transport route planning systems. Because of the limited availability of real-world datasets with desired properties, these systems were evaluated with only a very low number of datasets, respectively one and three datasets. A synthetic public transport dataset generator would make it easier for researchers to include a higher number of realistic datasets with various properties in their evaluations, which would be beneficial to the discovery of new insights from the evaluations. Network size, network sparsity and temporal range are examples of such properties, and different combinations of them may not always be available in real datasets, which motivates the need for generating synthetic, but realistic datasets with these properties.

In order to solve the need of realistic synthetic datasets with geospatial and temporal characteristics, we introduce a mimicking algorithm for generating realistic public transport data, which is the main contribution of this work. We observed a significant correlation between transport networks and the population distributions of their geographical areas, which is why population distributions are the driving factor within our algorithm. The cause of this correlation is obvious, considering transport animals are frequently used to transport people, but other – possibly independent – factors exist that influence transport networks as well, like certain points of interest such as tourist attractions and shopping areas. Our algorithm is subdivided into five sequential steps, inspired by existing methodologies from the domains of public transit planning [17] as a means to improve the realism of the algorithm’s output data. These steps include the creation of a geospatial region, the placement of stops, edges and routes, and the scheduling of trips. We provide an implementation of this algorithm, with different parameters to configure the algorithm. Finally, we confirm the realism of datasets that are generated by this algorithm using the existing generic structuredness metric [14] and new metrics that we introduce ourselves, which are specific to the public transport domain. The notable difference of this work compared to other synthetic dataset generators is that our generation algorithm specializes in generating public transit networks, while other generators either focus on other domains, or aim to be more general-purpose. For this, we provide an implementation and evaluation. Furthermore, our algorithm is based on population distributions and existing methodologies from public transit network design.

In the next section, we introduce the related work on dataset generation, followed by the background on public transit network design, and transit feed formats in Section 3. In Section 4, we introduce the main research question and hypothesis of this work. Next, our algorithm is presented in Section 5, followed by its implementation in Section 6. In Section 7, we present the evaluation of our implementation, followed by the a discussion and conclusion in Section 8 and Section 9.

2. Related Work

In this section, we present the related work on spatiotemporal and rdf dataset generation.

Spatiotemporal database systems store instances that are described using an identifier, a spatial location and a timestamp. In order to evaluate spatiotemporal indexing and querying techniques with datasets, automatic means exist to generate such datasets with predictable characteristics [22].

Brinkhoff [8] argues that moving objects tend to follow a predefined network. Using this and other statements, he introduces a spatiotemporal dataset generator. Such a network can be anything over which certain objects can move, ranging from railway networks to air traffic connections. The proposed parameter-based generator restricts the existence of the spatiotemporal objects to a predefined time period \([t_{\text{min}}, t_{\text{max}}]\). It is assumed that each edge in the network has a maximum allowed speed and capacity over which objects can move at a certain speed. The eventual speed of each object is defined by the maximum speed of its class, the maximum allowed speed of the edge, and the congestion of the edge based on its capacity. Furthermore, external events that can impact the movement of the objects, such as weather conditions, are represented as temporal grids over the network, which apply a decreasing factor on the maximum speed of the objects in certain areas. The existence of each object that is generated starts at a certain timestamp, which is determined by a certain function, and dies when it arrives at its destination. The starting node of an object can be chosen based on three approaches:
dataspace-oriented approaches Selecting the nearest node to a position picked from a two-dimensional distribution function that maps positions to nodes.

region-based approaches Improvement of the dataspace oriented approach where the data space is represented as a collection of cells, each having a certain chance of being the place of a starting node.

network-based approaches Selection of a network node based on a one-dimensional distribution function that maps each node to a certain chance.

Determining the destination node using one of these approaches leads to non-satisfying results. Instead, the destination is derived from the preferred length of a route. Each route is determined as the fastest path to a destination, weighed by the external events. Finally, the results are reported as either textual output, insertion into a database or a figure of the generated objects.

In order to improve the testability of Information Discovery Systems, a generic synthetic dataset generator [21] was developed that is able to generate synthetic data based on declarative semantic graph definitions. This graph is based on objects, attributes and relationships between them. The authors propose to generate new instances, such as people, based on a set of dependency rules. They introduce three types of dependencies for the generation of instances:

- **independent** Attribute values that are independent of other instances and attributes.
- **intra-record (horizontal) dependencies** Attribute values depending on other values of the same instance.
- **inter-record (vertical) dependencies** Relationships between different instances.

Their engine is able to accept such dependencies as part of a semantic graph definition, and iteratively create new instances to form a synthetic dataset.

The need for benchmarking RDF data management systems is illustrated by the existence of the Linked Data Benchmark Council [2] and the H2020 EU project for benchmarking of Big Linked Data. RDF benchmarks are typically based on certain datasets that are used as input to the tested systems. Many of these datasets are not always very closely related to real datasets [14], which may result in conclusions drawn from benchmarking results that do not translate to system behaviours in realistic settings.

Duan et al. [14] argue that the realism of an RDF dataset can be measured by comparing the structuredness of that dataset with a realistic equivalent. The authors show that real-world datasets are typically less structured than their synthetic counterparts, which can result in significantly different benchmarking results, since this level of structuredness can have an impact on how certain data is stored in RDF data management systems. This is because these systems may behave differently on datasets with different levels of structuredness, as they can have certain optimizations for some cases. In order to measure this structuredness, the authors introduce the coherence metric of a dataset \( D \) with a type system \( T \) that can be calculated as follows:

\[
CH(T, D) = \sum_{T \in T} WT(CV(T, D)) \times CV(T, D)
\]  

The type system \( T \) contains all the RDF types that are present in a dataset. \( CV(T, D) \) represents the coverage of a type \( T \) in a dataset \( D \), and is calculated as the fraction of type instances that set a value for all its properties. The factor \( WT(CV(T, D)) \) is used to weight this sum, so that the coherence is always a value between 0 and 1, with 1 representing a perfect structuredness. A maximal coherence means that all instances in the dataset have values for all possible properties in the type system, which is for example the case in relational databases without optional values. Based on this metric, the authors introduce a generic method for creating variants of real datasets with different sizes while maintaining a similar structuredness. The authors describe a method to calculate the coverage value of this dataset, which has been implemented as a procedure in the Virtuoso RDF store [26].

3. Public Transit Background

In this section, we present background on public transit planning that is essential to this work. We discuss existing public transit network planning methodologies and formats for exchanging transit feeds.

3.1. Public Transit Planning

The domain of public transit planning entails the design of public transit networks, rostering of crews, and all the required steps inbetween. The goal is to maximize the quality of service for passengers while minimizing the costs for the operator. Given a public demand and a topological area, this planning process aims to obtain routes, timetables and vehicle and crew...
assignment. A survey about 69 existing public transit planning approaches shows that these processes are typically subdivided into five sequential steps [17]:

1. **route design**, the placement of transit routes over an existing network.
2. **frequencies setting**, the temporal instantiation of routes based on the available vehicles and estimated demand.
3. **timetabling**, the calculation of arrival and departure times at each stop based on estimated demand.
4. **vehicle scheduling**, vehicle assignment to trips.
5. **crew scheduling and rostering**, the assignment of drivers and additional crew to trips.

In this paper, we only consider the first three steps, which lead to all the required information that is of importance to passengers in a public transit schedule. We present the three steps from this survey in more detail hereafter.

The first step, route design, requires the topology of an area and public demand as input. This topology describes the network in an area, which contains possible stops and edges between these stops. Public demand is typically represented as **origin-destination (OD)** matrices, which contain the number of passengers willing to go from origin stops to destination stops. Given this input, routes are designed based on the following objectives [17]:

- **area coverage** The percentage of public demand that can be served.
- **route and trip directness** A metric that indicates how much the actual trips from passengers deviate from the shortest path.
- **demand satisfaction** How many stops are close enough to all origin and destination points.
- **total route length** The total distance of all routes, which is typically minimized by operators.
- **operator-specific objectives** Any other constraints the operator has, for example the shape of the network.
- **historical background** Existing routes may influence the new design.

The next step is the setting of frequencies, which is based on the routes from the previous step, public demand and vehicle availability. The main objectives in this step are based on the following metrics [17]:

- **demand satisfaction** How many stops are serviced frequently enough to avoid overcrowding and long waiting times.
- **number of line runs** How many times each line is serviced – a trade-off between the operator’s aim for minimization and the public demand for maximization.

- **waiting time bounds** Regulation may put restrictions on minimum and maximum waiting times between line runs.
- **historical background** Existing frequencies may influence the new design.

The last important step for this work is timetabling, which takes the output from the previous steps as input, together with the public demand. The objectives for this step are the following:

- **demand satisfaction** Total travel time for passengers should be minimized.
- **transfer coordination** Transfers from one line to another at a certain stop should be taken into account during stop waiting times, including how many passengers are expected to transfer.
- **fleet size** The total amount of available vehicles and their usage will influence the timetabling possibilities.
- **historical background** Existing timetables may influence the new design.

### 3.2. Transit Feed Formats

The de-facto standard for public transport time schedules is the General Transit Feed Specification (gTFS) [gtfs](https://developers.google.com/transit/gtfs/). gTFS is an exchange format for transit feeds, using a series of csv files contained in a zip file. The specification uses the following terminology to define the rules for a public transit system:

- **Stop** is a geospatial location where vehicles stop and passengers can get on or off, such as platform 3 in the train station of Brussels.
- **Stop time** indicates a scheduled arrival and departure time at a certain stop.
- **Route** is a time-independent collection of stops, describing the sequence of stops a certain vehicle follows in a certain public transit line. For example the train route from Brussels to Ghent.
- **Trip** is a collection of stops with their respective stop times, such as the route from Brussels to Ghent at a certain time.

The zip file is put online by a public transit operator, to be downloaded by route planning [11] software. Two models are commonly used to then extract these rules into a graph [24]. In a **time-expanded model**, a large
4. Research Question

In order to generate public transport networks and schedules, we start from the hypothesis that both are correlated with the population distribution within the same area. More populated areas are expected to have more nearby and more frequent access to public transport, corresponding to the recurring demand satisfaction objective in public transit planning [17]. When we calculate the correlation between the distribution of stops in an area and its population distribution, we discover a positive correlation\(^3\) of 0.439 for Belgium and 0.459 for the Netherlands, thereby validating our hypothesis with a confidence of 99%. Because of the continuous population variable and the binary variable indicating whether or not there is a stop, the correlation is calculated using the point-biserial correlation coefficient\(^4\).

For the calculation of these correlations, we ignored the population value outliers. Following this conclusion, our mimicking algorithm will use such population distributions as input, and derive public transport networks and trip instances.

The main objective of a mimicking algorithm is to create realistic data. We will measure dataset realism by comparing the levels of structuredness of real-world datasets and their synthetic variants using the coherence metric introduced by Duan et al. \([13]\). Furthermore, we will measure the realism of different characteristics within public transport datasets, such as the location of stops, density of the network of stops, length of routes or the frequency of connections. We will quantify these aspects by measuring the distance of each aspect between real and synthetic datasets. This generic coherence metric together with domain-specific metrics will provide a way to evaluate dataset realism.

Based on this, we introduce the following research question for this work: “Can population distribution data be used to generate realistic synthetic public transport networks and scheduling?”

5. Method

In order to formulate an answer to our research question, we designed a mimicking algorithm that generates realistic synthetic public transit feeds. We based it on techniques from the domains of public transit planning, spatiotemporal and \(\text{KDF}\) dataset generation. We reuse the route design, frequencies setting and timetabling steps from the domain public transit planning, but prepend this with a network generation phase.

Figure 1 shows the model of the generated public transit feeds, with connections being the primary data element. We consider different properties in this model based on the independent, intra-record or inter-record dependency rules \([21]\), as discussed in Section 2. The arrival time in a connection can be represented as a fully intra-record dependency, because it depends on the time it departed and the stops it goes between. The departure time in a connection is both an intra-record and inter-record dependency, because it depends on the stop at which it departs, but also on the arrival time of the connection before it in the trip. Furthermore, the delay value can be seen as an inter-record dependency, because it is influenced by the delay value of the previous connection in the trip. Finally, the geospatial location of a stop depends on the location of its parent station, so this is also an inter-record dependency. All other unmentioned properties are independent.

In order to generate data based on these dependency rules, our algorithm is subdivided in five steps:

1. **Region**: Creation of a two-dimensional area of cells annotated with population density information.
2. **Stops**: Placement of stops in the area.
3. **Edges**: Connecting stops using edges.
4. **Routes**: Generation of routes between stops by combining edges.
5. **Trips**: Scheduling of timely trips over routes by instantiating connections. These steps are not fully sequential, since stop generation is partially executed before and after edge generation. The first three steps are required to generate a network, step 4 corresponds to the route design step in public transit planning and step 5 corresponds to both the frequencies setting and timetabling. These steps are explained in the following subsections.

5.1. **Region**

In order to create networks, we sample geographic regions in which such networks exist as two-dimensional matrices. The resolution is defined as a configurable number of cells per square of one latitude by one longitude. Network edges are then represented as links between these cells. Because our algorithm is population distribution-based, each cell contains a population density. These values can either be based on real population information from countries, or this can be generated based on certain statistical distributions. For the remainder of this paper, we will reuse the population distribution from Belgium as a running example, as illustrated in Figure 2.

5.2. **Stops**

Stop generation is divided into two steps. First, stops are placed based on population values, then the edge generation step is initiated after which the second phase of stop generation is executed where additional stops are created based on the generated edges.
clearly shows line structures. Stop placement after the first generation phase results can be seen in Figure 3b, which does not show these line structures. After the second generation phase, these line structures become more apparent as can be seen in Figure 3c. In this second generation phase, edges are modified so that sufficiently populated areas will be included in paths formed by edges, as illustrated by Figure 4. Random edges will iteratively be selected, weighted by the edge length measured as Euclidean distance. On each edge, a random cell is selected weighed by the population value in the cell. Next, a weighed random point in a certain area around this point is selected. This selected point is marked as a stop, the original edge is removed and two new edges are added, marking the path between the two original edge nodes and the newly selected node.

5.3. Edges

The next phase in public transit network generation connects the stops that were generated in the previous phase with edges. In order to simulate real transit network structures, we split up this generation phase into three sequential steps. In the first step, clusters of nearby stops are formed, to lay the foundation for short-distance routes. Next, these local clusters are connected with each other, to be able to form long-distance routes. Finally, a cleanup step is in place to avoid abnormal edge structures in the network.

---

5 The Euclidian distance based on geographical coordinates is always used to calculate distances in this work.
are connected with at least more than one other stations. The two earlier generation steps however generate a significant number of loose stops, which are connected with only a single other stop with a direct edge. In this step, these loose stops are identified, and an attempt is made to connect them to other nearby stops as shown in Algorithm 1. For each loose stop, this is done by first identifying the direction of the single edge of the loose stop on line 8. This direction is scaled by the radius in which to look for stops, and defines the stepsize for the loop that starts on line 10. This loop starts from the loose stop and iteratively moves the search position in the defined direction, until it finds a random stop in the radius, or the search distance exceeds the average distance of between the stops in the neighbourhood of this loose stop. This random stop from line 12 can be determined by finding all stations that have a distance to the search point that is below the radius, and picking a random stop from this collection. If such a stop is found, an edge is added from our loose stop to this stop.

Figure 5 shows an example of these three steps. After this phase, a network with stops and edges is available, and the actual transit planning can commence.

**Generator Objectives** The main guaranteed objective of the edge generator is that the stops form a single connected transit network graph. This is to ensure that all stops in the network can be reached from any other stop using at least one path through the network.

### 5.4. Routes

Given a network of stops and edges, this phase generates routes over the network. This is done by creating short and long distance routes in two sequential steps.

**Short-distance** The goal of the first step is to create short routes where vehicles deliver each passed stop. This step makes sure that all edges are used in at least one route, this ensures that each stop can at least be reached from each other stop with one or more transfers to another line. The algorithm does this by first determining a subset of the largest stops in the network, based on the population value. The shortest path from each large stop to each other large stop through the network is determined. if this shortest path is shorter than a predetermined value in terms of the number of edges, then this path is stored as a route, in which all passed stops are considered as actual stops in the route. For each edge that has not yet been passed after this, a route is created by iteratively adding unpassed edges to the route that are connected to the edge until an edge is found that has already been passed.

**Long-distance** In the next step, longer routes are created, where the transport vehicle not necessarily halts at each passed stop. This is done by iteratively picking two stops from the list of largest stops using the network-based method with each stop having an equal chance to be selected. A heuristic shortest path algorithm is used to determine a route between these stops. This algorithm searches for edges in the geographical direction of the target stop. This is done to limit the complexity of finding long paths through potentially large networks. A random amount of the largest stops on the path are selected, where the amount is a value between a minimum and maximum preconfigured route length. This iteration ends when a predetermined number of routes are generated.

**Generator Objectives** This algorithm takes into account the objectives of route design, as discussed in Section 2. More specifically, by first focusing on the largest stops, a minimal level of area coverage and demand satisfaction is achieved, because the largest stops correspond to highly populated areas, which therefore satisfies at least a large part of the population. By determining the shortest path between these largest stops, the route and trip directness between these stops is optimal. Finally, by not instantiating all possible routes over the network, the total route length is limited to a reasonable level.
Function RemoveLooseStops($S, E, N, \theta, \vartheta$)

Input: Set of stops $S$; Set of edges $E$ between the stops from $S$; Maximum number $N$ of closest stations to consider; Maximum average distance $\theta$ around a stop to be considered a loose station; Radius $\vartheta$ in which to look for stops.

1. foreach $s \in S$ with degree of 1 w.r.t. $E$ do
2.   $s_x = x$ coordinate of $s$; $s_y = y$ coordinate of $s$;
3.   $\Lambda = N$ closest stations to $s$ in $S$ excluding $s$;
4.   $\lambda = \text{closest station to } s \text{ in } S \text{ excluding } s$; $\lambda_x = x$ coordinate of $\lambda$; $\lambda_y = y$ coordinate of $\lambda$;
5.   $\delta = \text{average distance between each pair of stops in } \Lambda$;
6.   if $\delta < \theta$ and $\Lambda$ not empty then
7.     $\phi_x = (s_x - \lambda_x) \times \vartheta; \phi_y = (s_y - \lambda_y) \times \vartheta$;
8.     $o_x = s_x; o_y = s_y$;
9.     while distance between $o$ and $s < \delta$ do
10.    $o_x += \phi_x; o_y += \phi_y$;
11.    $s' = \text{random station around } o \text{ with radius } \delta \times \vartheta$;
12.    if $s'$ exists, add edge between $s$ and $s'$ to $E$ and continue next for-loop iteration;

Algorithm 1: Reduce the number of loose stops by adding additional edges.

5.5. Trips

A time-agnostic transit network with routes has been generated in the previous steps. In this final phase, we temporally instantiate routes by first determining starting times for trips, after which the following stop times can be calculated based on route distances. Instead of generating explicit timetables, as is done in typical transit scheduling methodologies, we create fictional rides of vehicles. In order to achieve realistic trip times, we approximate real trip time distributions, with the possibility to encounter delays.

As mentioned before in Section 2, each consecutive pair of start and stop time in a trip over an edge corresponds to a connection. A connection can therefore be represented as a pair of timestamps, a link to the edge representing the departure and arrival stop, a link to the trip it is part of, and its index within this trip.

Trip Starting Times The trips generator iteratively creates new connections until a predefined number is reached. For each connection, a random route is selected with a larger chance of picking a long route. Next, a random start time of the connection is determined. This is done by first picking a random day within a certain range. After that, a random hour of the day is determined using a preconfigured distribution. This distribution is derived from the public logs of iRail API in Belgium. A separate hourly distribution is used for weekdays and weekends, which is chosen depending on the random day that was determined.

Stop Times Once the route and the starting time have been determined, different stop times across the trip can be calculated. For this, we take into account the following factors:

- Maximum vehicle speed $\omega$, preconfigured constant.
- Vehicle acceleration $\varsigma$, preconfigured constant.
- Connection distance $\delta$, Euclidian distance between stops in network.
- Stop size $\sigma$, derived from population value.

For each connection in the trip, the time it takes for a vehicle to move between the two stops over a certain distance is calculated using the formula in Equation 4. Equation 2 calculates the required time to reach maximum speed and Equation 3 calculates the required distance to reach maximum speed. This formula simulates the vehicle speeding up until its maximum speed, and slowing down again until it reaches its destination. When the distance is too short, the vehicle will not reach its maximum speed, and just speeds up as long as...
possible until is has to slow down again to stop in time.

\[
T_{\omega} = \frac{\omega}{\varsigma} \tag{2}
\]

\[
\delta_{\omega} = T_{\omega}^2 \cdot \varsigma \tag{3}
\]

\[
\text{duration} = \begin{cases} 
2T_{\omega} + (\delta - 2\delta_{\omega})/\omega & \text{if } \delta_{\omega} < \delta/2 \\
\sqrt{2\delta/\varsigma} & \text{otherwise}
\end{cases} \tag{4}
\]

Not only the connection duration, but also the waiting times of the vehicle at each stop are important for determining the stop times. These are calculated as a constant minimum waiting time together with a waiting time that increases for larger stop sizes \(\sigma\), this increase is determined by a predefined growth factor.

### Delays

Finally, each connection in the trip will have a certain chance to encounter a delay. When a delay is applicable, this delay is randomly chosen within a certain range. Next to this, also a cause of the delay is determined from a preconfigured list. These causes are based on the Traffic Element Events from the Transport Disruption ontology, which contains a number of events that are not planned by the network operator such as strikes, bad weather or animal collisions. Different types of delays can have a different impact factor of the delay value, for instance, simple delays caused by rush hour would have a lower impact factor than a major train defect. Delays are carried over to next connections in the trip, with again a chance of encountering additional delay. Furthermore, these delay values can also be reduced when carried over to the next connection by a certain predetermined factor, which simulates the attempt to reduce delays by letting vehicles drive faster.

### Generator Objectives

For trip generation, we take into account several objectives from the setting of frequencies and timetabling from transit planning \[17\]. By instantiating more long distance routes, we aim to increase demand satisfaction as much as possible, because these routes deliver busy and populated areas, and the goal is to deliver these more frequently. Furthermore, by taking into account realistic time distributions for trip instantiation, we also adhere to this objective. Secondly, by ensuring waiting times at each stop that are longer for larger stations, the transfer coordination objective is taken into account to some extent.

6. **Implementation**

In this section, we discuss the implementation details of podigg, based on the generator algorithm introduced in Section 5. Podigg is split up into two parts: the main podigg generator, which outputs \textsc{gtfs} data, and podigg-lc, which depends on the main generator to output \textsc{rdf} data. Serialization in \textsc{rdf} using existing ontologies, such as the \textsc{gtfs} and Linked Connections ontologies, allows this inherently linked data to be used within \textsc{rdf} data management systems, where it can for instance be used for benchmarking purposes. Providing output in \textsc{gtfs} will allow this data to be used directly within all systems that are able to handle transit feeds, such as route planning systems. The two generator parts will be explained hereafter, followed by a section on how the generator can be configured using various parameters.

6.1. **podigg**

The main requirement of our system is the ability to generate realistic public transport datasets using the mimicking algorithm that was introduced in Section 5. This means that given a population distribution of a certain region, the system must be able to design a network of routes, and determine timely trips over this network.

Podigg is implemented to achieve this goal. It is written in JavaScript using Node.js, and is available under an open license on GitHub [10]. In order to make installation and usage more convenient, podigg is available as a Node module on the npm [11] package manager and as a Docker image on Docker Hub [12] to easily run on any platform. Every sub-generator that was explained in Section 5 is implemented as a separate module. This makes podigg highly modifiable and composable, because different implementations of sub-generators can easily be added and removed. Furthermore, this flexible composition makes it possible to use real data instead of certain sub-generators. This can be useful for instance when a certain public transport network is already available, and only the trips and connections need to be generated.

We designed podigg to be highly configurable to adjust the characteristics of the generated output across

---

8. [http://vocab.gtfs.org/terms](http://vocab.gtfs.org/terms)
10. [https://github.com/PoDiGG/podigg](https://github.com/PoDiGG/podigg)
12. [https://hub.docker.com/r/podigg/podigg](https://hub.docker.com/r/podigg/podigg)
The **gtfs** files that are written by **podigg**, with their corresponding sub-generators that are responsible for generating the required data. The files in bold refer to files that are required by the **gtfs** specification.

<table>
<thead>
<tr>
<th>File</th>
<th>Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>agency.txt</td>
<td>Constant</td>
</tr>
<tr>
<td>stops.txt</td>
<td>Stops</td>
</tr>
<tr>
<td>routes.txt</td>
<td>Routes</td>
</tr>
<tr>
<td>trips.txt</td>
<td>Trips</td>
</tr>
<tr>
<td>stop_times.txt</td>
<td>Trips</td>
</tr>
<tr>
<td>calendar.txt</td>
<td>Trips</td>
</tr>
<tr>
<td>calendar_dates.txt</td>
<td>Trips</td>
</tr>
<tr>
<td>delays.txt</td>
<td>Trips</td>
</tr>
</tbody>
</table>

Table 1

The **gtfs** files that are generated by the **podigg** modules. This table does not contain references to the **region** and **edges** generator, because they are only used internally as prerequisites to the later steps. All required files are created to have a valid **gtfs** dataset. Next to that, the optional file for exceptional service dates is created. Furthermore, **delays.txt** is created, which is not part of the **gtfs** specification. It is an extension we provide in order to serialize delay information about each connection in a trip. These delays are represented in a **csv** file containing columns for referring to a connection in a trip, and contains delay values in milliseconds and a certain reason per connection arrival and departure, as shown in Listing 1.

In order to easily observe the network structure in the generated datasets, **podigg** will always produce a figure accompanying the **gtfs** dataset. Figure 6 shows an example of such a visualization.

Because the generation of large datasets can take a long time depending on the used parameters, **podigg** has a logging mechanism, which provides continuous feedback to the user about the current status and progress of the generator.

Finally, **podigg** provides the option to derive realistic public transit queries over the generated network, aimed at testing the load of route planning systems. This is done by iteratively selecting two random stops weighed by their size and choosing a random starting time based on the same time distribution as discussed in Section 5.5.

This is serialized to a **json** format that was introduced for benchmarking the **Linked Connections** route planner.

6.2. **podigg-lc**

**podigg-lc** is an extension of **podigg**, that outputs data in Turtle/RDF using the ontologies shown in Figure 1. It is also implemented in JavaScript using Node.js, and available under an open license on GitHub[^14]. **PODiGG-lc** is also available as a Node module on npm[^15] and as a Docker image on Docker Hub[^16]. For this, we extended the **GTFS2LC** tool[^17] that is able to convert **gtfs** datasets to RDF using the **Linked Connections** and **gtfs** ontologies. The original tool serializes a minimal subset of the **gtfs** data, aimed at being used for **Linked Connections** route planning over connections. Our extension also serializes trip, station and route instances, with their relevant interlinking. Furthermore, our **gtfs** extension for representing delays is also supported, and is serialized using a new **Linked Connections Delay** ontology[^18] that we created.

6.3. Configuration

**podigg** accepts a wide range of parameters that can be used to configure properties of the different sub-
We first evaluate the realism of the generated datasets were executed on a 64-bit Ubuntu 14.04 machine with 128 GB of memory and a 24-core 2.40 GHz cpu. 

<table>
<thead>
<tr>
<th>trip.id</th>
<th>stop.sequence</th>
<th>delay.departure</th>
<th>delay.arrival</th>
<th>delay.departure_reason</th>
<th>delay.arrival_reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>100_4</td>
<td>0</td>
<td>0</td>
<td>1405754</td>
<td>,</td>
<td>,td:RepairWork</td>
</tr>
<tr>
<td>100_6</td>
<td>0</td>
<td>0</td>
<td>1751671</td>
<td>,</td>
<td>,td:BrokenDownTrain</td>
</tr>
<tr>
<td>100_6</td>
<td>1</td>
<td>0</td>
<td>1751671</td>
<td>,</td>
<td>,td:BrokenDownTrain</td>
</tr>
<tr>
<td>100_7</td>
<td>0</td>
<td>,2782295</td>
<td>0</td>
<td>,</td>
<td>,td:TreeAndVegetationCuttingWork,</td>
</tr>
</tbody>
</table>

Listing 1: Sample of a delays.txt file in a gtrss dataset

generators. Table 2 shows an overview of the parameters, grouped by each sub-generator. Podig and Podig-LC accept these parameters either in a JSON configuration file or via environment variables. Both Podig and Podig-LC produce deterministic output for identical sets of parameters, so that datasets can easily be reproduced given the configuration. The seed parameter can be used to introduce pseudo-randomness into the output.

7. Evaluation

In this section, we discuss our evaluation of Podig. We first evaluate the realism of the generated datasets using a constant seed by comparing its coherence to real datasets, followed by a more detailed realism evaluation of each sub-generator using distance functions. Finally, we provide an indicative efficiency and scalability evaluation of the generator and discuss practical dataset sizes. All scripts that were used for the following evaluation can be found on GitHub. Our experiments were executed on a 64-bit Ubuntu 14.04 machine with 128 GB of memory and a 24-core 2.40 GHz CPU.

7.1. Coherence

Metric In order to determine the realism of synthetic RDF datasets, the coherence metric [14] measuring the structuredness of a dataset is used. In RDF dataset generation, the goal is to reach a level of structuredness similar to real datasets. As mentioned before in Section 2 many synthetic datasets have a level of structuredness that is higher than their real-world counterparts. Therefore, our coherence evaluation should indicate that our generator is not subject to the same problem. We have implemented a command-line tool to calculate the coherence value for any given input dataset.

Results When measuring the coherence of the Belgian railway, buses and Dutch railway datasets, we discover high values for both the real-world datasets and the synthetic datasets, as can be seen in Table 3. These nearly maximal values indicate that there is a very high level of structuredness in these real-world datasets. Most instances have all the possible values, unlike most typical RDF datasets, which have values around or below 0.6 [14]. That is because of the very specialized nature of this dataset, and the fact that they originate from gtrss datasets that have the characteristics of relational databases. Only a very limited number of classes and predicates are used, where almost all instances have the same set of attributes. In fact, these very high coherence values for real-world datasets simplify the process of synthetic dataset generation, as less attention needs to be given to factors that lead to lower levels of structuredness, such as optional attributes for instances. When generating synthetic datasets using Podig with the same number of stops, routes and connections for the three gold standards, we measure very similar coherence values, with differences ranging from 0.08% to 1.64%. This confirms that Podig is able to create datasets with the same high level of structuredness to real datasets of these types, as it inherits the relational database characteristics from its gtrss-centric mimicking algorithm.

7.2. Distance to Gold Standards

While the coherence metric is useful to compare the level of structuredness between datasets, it does not give any detailed information about how real synthetic datasets are in terms of their distance to the real datasets. In this case, we are working with public transit feeds with a known structure, so we can look at the different datasets aspects in more detail. More specifically, we start from real geographical areas with their population distributions, and consider the distance functions between stops, edges, routes and trips for the synthetic and gold standard datasets. In order to check the applicability of Podig to different transport types and geographical areas, we compare with the gold standard data of the Belgian railway, the Belgian buses and the Dutch railway. The scripts that were used to derive these
<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed</td>
<td>1</td>
<td>The random seed</td>
</tr>
<tr>
<td>region_generator</td>
<td>isolated</td>
<td>Name of a region generator. (isolated, noisy or region)</td>
</tr>
<tr>
<td>lat_offset</td>
<td>0</td>
<td>Value to add with all generated latitudes</td>
</tr>
<tr>
<td>lon_offset</td>
<td>0</td>
<td>Value to add with all generated longitudes</td>
</tr>
<tr>
<td>cells_per_latlon</td>
<td>100</td>
<td>How many cells go in 1 latitude/longitude</td>
</tr>
<tr>
<td>stops</td>
<td>600</td>
<td>How many stops should be generated</td>
</tr>
<tr>
<td>min_station_size</td>
<td>0.01</td>
<td>Minimum cell population value for a stop to form</td>
</tr>
<tr>
<td>max_station_size</td>
<td>30</td>
<td>Maximum cell population value for a stop to form</td>
</tr>
<tr>
<td>start.stop_choice_power</td>
<td>4</td>
<td>Power for selecting large population cells as stops</td>
</tr>
<tr>
<td>min_interstop_distance</td>
<td>1</td>
<td>Minimum distance between stops in number of cells</td>
</tr>
<tr>
<td>factor.stops_post_edges</td>
<td>0.66</td>
<td>Factor of stops to generate after edges</td>
</tr>
<tr>
<td>edge_choice_power</td>
<td>2</td>
<td>Power for selecting longer edges to generate stops on</td>
</tr>
<tr>
<td>stop_around_edge_choice_power</td>
<td>4</td>
<td>Power for selecting large population cells around edges</td>
</tr>
<tr>
<td>stop_around_edge_radius</td>
<td>2</td>
<td>Radius in number of cells around an edge to select points</td>
</tr>
<tr>
<td>max_intracluster_distance</td>
<td>100</td>
<td>Maximum distance between stops in one cluster</td>
</tr>
<tr>
<td>max_intracluster_distance.growthfactor</td>
<td>0.1</td>
<td>Power for clustering with more distant stops</td>
</tr>
<tr>
<td>post_cluster.max_intracluster_distancefactor</td>
<td>1.5</td>
<td>Power for connecting a stop with multiple stops</td>
</tr>
<tr>
<td>loosestations.neighbourcount</td>
<td>3</td>
<td>Neighbours around a loose station that should define its area</td>
</tr>
<tr>
<td>loosestations.max_range.factor</td>
<td>0.3</td>
<td>Maximum loose station range relative to the total region size</td>
</tr>
<tr>
<td>loosestations.max.iterations</td>
<td>10</td>
<td>Maximum iteration number to try to connect one loose station</td>
</tr>
<tr>
<td>loosestations.search.radius.factor</td>
<td>0.5</td>
<td>Loose station neighbourhood size factor</td>
</tr>
<tr>
<td>routes</td>
<td>1000</td>
<td>The number of routes to generate</td>
</tr>
<tr>
<td>largest_stations_fraction</td>
<td>0.05</td>
<td>The fraction of stops to form routes between</td>
</tr>
<tr>
<td>penalize.station.size.area</td>
<td>10</td>
<td>The area in which stop sizes should be penalized</td>
</tr>
<tr>
<td>max_route_length</td>
<td>10</td>
<td>Maximum number of edges for a route in the macro-step</td>
</tr>
<tr>
<td>min_route_length</td>
<td>4</td>
<td>Minimum number of edges for a route in the macro-step</td>
</tr>
<tr>
<td>time_initial</td>
<td>0</td>
<td>The initial timestamp (ms)</td>
</tr>
<tr>
<td>time_final</td>
<td>24 * 3600000</td>
<td>The final timestamp (ms)</td>
</tr>
<tr>
<td>connections</td>
<td>30000</td>
<td>Number of connections to generate</td>
</tr>
<tr>
<td>stop.wait_min</td>
<td>60000</td>
<td>Minimum waiting time per stop</td>
</tr>
<tr>
<td>stop.wait.size.factor</td>
<td>60000</td>
<td>Waiting time to add multiplied by station size</td>
</tr>
<tr>
<td>route_choice_power</td>
<td>2</td>
<td>Power for selecting longer routes for connections</td>
</tr>
<tr>
<td>vehicle.max.speed</td>
<td>160</td>
<td>Maximum speed of a vehicle in km/h</td>
</tr>
<tr>
<td>vehicle.speedup</td>
<td>1000</td>
<td>Vehicle speedup in km/(h$^2$)</td>
</tr>
<tr>
<td>hourly.weekday_distribution</td>
<td>...</td>
<td>Hourly connection chances for weekdays</td>
</tr>
<tr>
<td>hourly.weekend_distribution</td>
<td>...</td>
<td>Hourly connection chances for weekend days</td>
</tr>
<tr>
<td>delay.chance</td>
<td>0</td>
<td>Chance for a connection delay</td>
</tr>
<tr>
<td>delay.max</td>
<td>3600000</td>
<td>Maximum delay</td>
</tr>
<tr>
<td>delay.choice_power</td>
<td>1</td>
<td>Power for selecting larger delays</td>
</tr>
<tr>
<td>delay.reasons</td>
<td>... $^2$</td>
<td>Default reasons and chances for delays</td>
</tr>
<tr>
<td>delay.reduction.duration.fraction</td>
<td>0.1</td>
<td>Maximum part of connection duration to subtract for delays</td>
</tr>
<tr>
<td>start.stop_choice_power</td>
<td>4</td>
<td>Power for selecting large starting stations</td>
</tr>
<tr>
<td>query_count</td>
<td>100</td>
<td>The number of queries to generate</td>
</tr>
<tr>
<td>time_initial</td>
<td>0</td>
<td>The initial timestamp</td>
</tr>
<tr>
<td>time_final</td>
<td>24 * 3600000</td>
<td>The final timestamp</td>
</tr>
<tr>
<td>max.time.before.departure</td>
<td>3600000</td>
<td>The maximum time until a query should be started</td>
</tr>
<tr>
<td>hourly.weekday_distribution</td>
<td>...</td>
<td>Chance for each hour to have a connection on a weekday</td>
</tr>
<tr>
<td>hourly.weekend_distribution</td>
<td>...</td>
<td>Chance for each hour to have a connection on a weekend day</td>
</tr>
</tbody>
</table>

Table 2: Configuration parameters for the different sub-generators. Time values are represented in milliseconds. $^1$ Time distributions are based on public route planning logs [7]. $^2$ Default delays are based on the Transport Disruption ontology [https://transportdisruption.github.io/].
The complexity of $\Delta$ is $O(||A|| \cdot ||B|| \cdot \kappa(f))$.

$$\kappa(f) := \arg\min_{a \in A} f(a, b)$$

Equation 5

$$\Delta(A, B, f) := \frac{\sum_{a \in A} f(a, \chi(a, B, f)) + \sum_{b \in B} f(b, \chi(b, A, f))}{||A|| + ||B||}$$

Equation 6

**Stops Distance** For measuring the distance between two sets of stops $S_1$ and $S_2$, we introduce the distance function from Equation 7. This measures the distance between every possible pair of stops using the Euclidean distance function $d$. Assuming a constant execution time for $\kappa(d)$, the computational complexity for $\Delta_s$ is $O(||S_1|| \cdot ||S_2||)$.

$$\Delta_s(S_1, S_2) := \Delta(S_1, S_2, d)$$

Equation 7

**Edges Distance** In order to measure the distance between two sets of edges $E_1$ and $E_2$, we use the distance function from Equation 8, which measures the distance between all pairs of edges using the distance function $d_e$. This distance function $d_e$, which is introduced in Equation 9, measures the Euclidean distance between the start and endpoints of each edge, and between the different edges, weighted by the length of the edges. The constant 1 in Equation 9 is to ensure that the distance between two edges that have an equal length, but exist at a different position, is not necessarily zero. The computational cost of $d_e$ can be considered as a constant, so the complexity of $\Delta_e$ becomes $O(||E_1|| \cdot ||E_2||)$.

$$\Delta_e(E_1, E_2, d_e) := \Delta(E_1, E_2, d_e)$$

Equation 8

$$d_e(e_1, e_2) := \min(d(e_1, e_2) + d(e_1, e_2),$$

$$\quad \quad d(e_1, e_2) + d(e_1, e_2)) \cdot (d(e_1, e_2) - d(e_1, e_2) + 1)$$

Equation 9

**Routes Distance** Similarly, the distance between two sets of routes $R_1$ and $R_2$ is measured in Equation 10 by applying $\Delta$ for the distance function $d_r$. Equation 11 introduces this distance function $d_r$ between two routes, which is calculated by considering the edges in each route and measuring the distance between those two sets using the distance function $\Delta_e$ from Equation 8. By considering the maximum amount of edges per route as $e_{max}$, the complexity of $d_r$ becomes $O(e_{max})$. This leads to a complexity of $O(||R_1|| \cdot ||R_2|| \cdot e_{max})$ for $\Delta_r$.

$$\Delta_r(R_1, R_2) := \Delta(R_1, R_2, d_r)$$

Equation 10

$$d_r(r_1, r_2) := \Delta_e(r_1, r_2, d_e)$$

Equation 11

**Connections Distance** Finally, we measure the distance between two sets of connections $C_1$ and $C_2$ using the function from Equation 12. The distance between two connections is measured using the function from Equation 13, which is done by considering their respective temporal distance weighed by a constant $\epsilon$ and their geospatial distance using the edge distance function $d_e$. The complexity of time calculation in $d_c$ can

22 https://github.com/PoDiGG/population-density-generator

23 When serializing time in milliseconds, we set $\epsilon$ to 60000.
be considered being constant, which makes it overall complexity $O(e_{\text{max}})$. For $\Delta_c$, this leads to a complexity of $O(||C_1|| \cdot ||C_2|| \cdot e_{\text{max}}^2)$.

$$\Delta_c(C_1, C_2) := \Delta(C_1, C_2, d_c)$$  \hspace{1cm} (12)

$$d_c(c_1, c_2) := ((c_1^{\text{departureTime}} - c_2^{\text{departureTime}}) + (c_1^{\text{arrivalTime}} - c_2^{\text{arrivalTime}})/\epsilon) + d_c(c_1, c_2)$$  \hspace{1cm} (13)

**Computability** When using the introduced functions for calculating the distance between stops, edges, routes or connections, execution times can become long for a large number of elements because of their large complexity. When applying these distance functions for realistic numbers of stops, edges, routes and connections, several optimizations should be done in order to calculate these distances in a reasonable time. A major contributor for these high complexities is $\chi$ for finding the closest element from a set of elements to a given element, as introduced in Equation 5. In practice, we only observed extreme execution times for the respective distance between routes and connections. For routes, we implemented an optimization, with the same worst-case complexity, that indexes routes based on their geospatial position, and performs radial search around each route when the closest one from a set of other routes should be found. For connections, we consider the linear time dimension when performing binary search for finding the closest connection from a set of elements.

**Metrics** In order to measure the realism of each generator phase, we introduce a realism factor $\rho$ for each phase. These values are calculated by measuring the distance from randomly generated elements to the gold standard, divided by the distance from the actually generated elements to the gold standard, as shown below for respectively stops, edges, routes and connections. We consider these randomly generated elements having the lowest possible level of realism, so we use these as a weighting factor in our realism values.

$$\rho_s(S_{\text{rand}}, S_{\text{gen}}, S_{gs}) := \Delta_s(S_{\text{rand}}, S_{gs})/\Delta_s(S_{\text{gen}}, S_{gs})$$

$$\rho_e(E_{\text{rand}}, E_{\text{gen}}, E_{gs}) := \Delta_e(E_{\text{rand}}, E_{gs})/\Delta_e(E_{\text{gen}}, E_{gs})$$

$$\rho_r(R_{\text{rand}}, R_{\text{gen}}, R_{gs}) := \Delta_r(R_{\text{rand}}, R_{gs})/\Delta_r(R_{\text{gen}}, R_{gs})$$

$$\rho_c(C_{\text{rand}}, C_{\text{gen}}, C_{gs}) := \Delta_c(C_{\text{rand}}, C_{gs})/\Delta_c(C_{\text{gen}}, C_{gs})$$

**Results** We measured these realism values with gold standards for the Belgian railway, the Belgian buses and the Dutch railway. In each case, we used an optimal set of parameters to achieve the most realistic generated output. Table 4 shows the realism values for the three cases, which are visualized in Figures 7 to 10. Each value is larger than 1, showing that the generator at least produces data that is closer to the gold standard, and is therefore more realistic. The realism for edges is in each case very large, showing that our algorithm produces edges that are very similar to actual edge placement in public transport networks according to our distance function. Next, the realism of stops is lower, but still sufficiently high to consider it as realistic. Finally, the values for routes and connections show that these sub-generators produce output that is closer to the gold standard than the random function according to our distance function. Routes achieve the best level of realism for the Belgian railway case. For this same case, the connections are however only slightly closer to the gold standard than random placement, while for the other cases the realism is more significant. All of these realism values show that podigg is able to produce realistic data for different regions and different transport types.

7.3. Performance

**Metrics** While performance is not the main focus of this work, we provide an indicative performance evaluation in this section in order to discover the bottlenecks and limitations of our current implementation that could be further investigated and resolved in future work. We measure the impact of different parameters on the execution times of the generator. The three main parameters for increasing the output dataset size are the number of stops, routes and connections. Because the number of edges is implicitly derived from the number of stops in order to reach a connected network, this can not be configured directly. In this section, we start from a set of parameters that produces realistic output data that is similar to the Belgian railway case. We let the value for each of these parameters increase to see the evolution of the execution times and memory usage.

![https://github.com/PoDiGG/podigg-evaluate/blob/master/bin/evaluate.js](https://github.com/PoDiGG/podigg-evaluate/blob/master/bin/evaluate.js)
Table 4

<table>
<thead>
<tr>
<th></th>
<th>Belgian railway</th>
<th>Belgian buses</th>
<th>Dutch railway</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stops</td>
<td>5.5490</td>
<td>297.0888</td>
<td>4.0017</td>
</tr>
<tr>
<td>Edges</td>
<td>147.4299</td>
<td>1633.4693</td>
<td>318.4131</td>
</tr>
<tr>
<td>Routes</td>
<td>2.2420</td>
<td>1.6094</td>
<td>1.3095</td>
</tr>
<tr>
<td>Connections</td>
<td>1.0451</td>
<td>1.5006</td>
<td>1.3017</td>
</tr>
</tbody>
</table>

Realism values for the three gold standards in case of the different sub-generators, respectively calculated for the stops $\rho_s$, edges $\rho_e$, routes $\rho_r$ and connections $\rho_c$.

Results

Figure 7. Stops for the Belgian railway case.

Figure 8. Edges for the Belgian railway case.

An important aspect of dataset generation is its ability to output various dataset sizes. In ronsgo, different options are available for tweaking these sizes. Increasing the time range parameter within the generator increases the number of connections while the number of stops and routes will remain the same. When enlarging the geographical area over the same period of time, the opposite is true. As a rule of thumb, based on the number of triples per connection, stops and routes, the total number of generated triples per dataset is approximately...
Figure 9. Routes for the Belgian railway case.

Figure 10. Connections per hour for the Belgian railway case.

Figure 11. Execution times when increasing the number of stops, routes or connections.

Figure 12. Memory usage when increasing the number of stops, routes or connections.

practice, we reach 235,700 triples when running with these parameters, which is slightly higher because of the other triples that are not taken into account for this simplified formula, such as the ones for trips, stations and delays.

8. Discussion

In this section, we discuss the main characteristics and the limitations of this work. Finally, we mention several poisdigit use cases.

8.1. Characteristics

Our main research question on how to generate realistic synthetic public transport networks has been answered by the introduction of the mimicking algorithm from Section 5, based on commonly used practises in transit network design. This is based on the accepted hypothesis that the population distribution of an area is correlated with its transport network design and scheduling. We measured the realism of the generated datasets using the coherence metric in Section 7.1 and more fine-grained realism metrics for different public transport aspects in Section 7.2.

poisdigit, our implementation of the algorithm, accepts a wide range of parameters to configure the mimicking algorithm. poisdigit and poisdigit-lc are able to output the mimicked data respectively as GTFS and RDF datasets, together with a visualization of the generated transit network. Our system can be used without requiring any extensive setup or advanced programming skills, as it consists of simple command lines tools that can be invoked with a number of optional parameters to configure the generator. Our system is proven to be generalizable to other transport types, as we evaluated poisdigit for the bus and train transport type, and the Belgium and Netherlands geospatial regions in Section 7.2.
8.2. Limitations and Future Work

In this section, we discuss the limitations of the current mimicking algorithm and its implementation, together with further research opportunities.

Memory Usage The sequential steps in the presented mimicking algorithm require persistence of the intermediary data that is generated in each step. Currently, podig is implemented in such a way that all data is kept in-memory for the duration of the generation, until it is serialized. When large datasets need to be generated, this requires a larger amount of memory to be allocated to the generator. Especially for large amounts of routes or connections, where 100 million connections already require almost 10GB of memory to be allocated. While performance was not the primary concern in this work, in future work, improvements could be made in the future. A first possible solution would be to use a memory-mapped database for intermediary data, so that not all data must remain in memory at all times. An alternative solution would be to modify the mimicking process to a streaming algorithm, so that only small parts of data need to be kept in memory for datasets of any size. Considering the complexity of transit networks, a pure streaming algorithm might not be feasible, because route design requires knowledge of the whole network. The generation of connections however, could be adapted so that it works as a streaming algorithm.

Realism We aimed to produce realistic transit feeds by reusing the methodologies learned in public transit planning. Our current evaluation compares generated output to real datasets, as no similar generators currently exist. When similar generation algorithms are introduced in the future, this evaluation can be extended to compare their levels of realism. Our results showed that all subgenerators, except for the trips generator, produced output with a high realism value. The trips are still closer to real data than a random generator, but this can be further improved in future work. This can be done by for instance taking into account network capacities on certain edges when instantiating routes as trips, because we currently assume infinite edge capacities, which can result in a large amount of connections over an edge at the same time, which may not be realistic for certain networks. Alternatively, we could include other factors in the generation algorithm, such as the location of certain points of interest, such as shopping areas, schools and tourist spots. In the future, a study could be done to identify and measure the impact of certain points of interest on transit networks, which could be used as additional input to the generation algorithm to further improve the level of realism. Next to this, in order to improve transfer coordination, possible transfers between trips should be taken into account when generating stop times. Limiting the network capacity will also lead to natural congestion of networks, which should also be taken into account for improving the realism. Furthermore, the total vehicle fleet size should be considered, because we currently assume an infinite number of available vehicles. It is more realistic to have a limited availability of vehicles in a network, with the last position of each vehicle being of importance when choosing the next trip for that vehicle.

Alternative Implementations An alternative way of implementing this generator would be to define declarative dependency rules for public transport networks, based on the work by Pengyue et. al. [21]. This would require a semantic extension to the engine so that is aware of the relevant ontologies and that it can serialize to one or more rdf formats. Alternatively, machine learning techniques such as deep learning could be used to automatically learn the structure and characteristics of public transport networks and create similar realistic synthetic datasets. The downside of machine learning techniques is however that it is typically more difficult to tweak parameters of automatically learned models when specific characteristics of the output need to be changed, when compared to a manually implemented algorithm. Sensitivity analysis could help to determine the impact of such parameters in order to understand the learned models better.

Streaming Extension Finally, the temporal aspect of public transport networks is useful for the domain of rdf stream processing. Instead of producing single static datasets as output, podig could be adapted to produce rdf streams of connections and delays, where information about stops and routes are part of the background knowledge. Such an extension can become part of a benchmark, such as CityBench and LSBench, for assessing the performance of rdf stream processing systems with temporal and geospatial capabilities.

8.3. podig In Use

podig and podig-lc have been developed for usage within the nöbbt platform. This platform is being developed within the nöbbt project and aims to provide an environment for benchmarking rdf systems for Big
Linked Data. The platform provides several default dataset generators, including podig, which can be used to benchmark systems.

podig, and its generated datasets are being used in the eswc Mighty Storage Challenge 2017 [16]. The first task of this challenge consists of rdf data ingestion into triple stores, and querying over this data. Because of the temporal aspect of public transport data in the form of connections, podig datasets are fragmented by connection departure time, and transformed to a data stream that can be inserted. In task 4 of this challenge, the efficiency of faceted browsing solutions is benchmarked [23]. Because of the geographical property of public transport data, podig datasets are being used for this benchmark.

Finally, podig is being used for creating virtual transport networks of variable size for the purposes of benchmarking route planning frameworks, such as Linked Connections [8].

9. Conclusions

In this paper, we introduced a mimicking algorithm for public transport data, based on steps that are used in real-world transit planning. Our method splits up this process into several sub-generators and uses population distributions of an area as input. As part of this paper, we introduce podig, a reusable framework that accepts a wide range of parameters to configure the generation algorithm.

Results show that the structuredness of generated datasets are similar to real public transport datasets. Furthermore, we introduced several functions for measuring the realism of synthetic public transport datasets compared to a gold standard on several levels, based on distance functions. The realism was confirmed for different regions and transport types. Finally, the execution times and memory usages were measured when increasing the most important parameters, which showed a linear increase for each parameter, showing that the generator is able to scale to large dataset outputs.

For these reasons, the public transport mimicking algorithm we introduced, with podig and podig-lc as implementations, provides a useful tool for benchmarking with realistic geospatial and temporal rdf datasets. Flexible configuration allows datasets of any size to be created and various characteristics to be tweaked to achieve highly specialized datasets for testing specific use cases. These datasets can for example used to benchmark the efficiency and performance of geospatial and temporal systems—such as public transport route planners—under these different circumstances, and therefore lower the barrier towards more efficient and performant systems.

Acknowledgements

We wish to thank Henning Petzka for his help with discovering issues and providing useful suggestions for the podig implementation. The described research activities were funded by the H2020 project Новьт (#688227).

References


