Journal of Transport Geography 62 (2017) 92–97

Contents lists available at ScienceDirect

Journal of Transport Geography

journal homepage: www.elsevier.com/locate/jtrangeo

Constructing a routable retrospective transit timetable from a real-time vehicle location feed and GTFS

Nate Wessel*, Jeff Allen, Steven Farber

University of Toronto, Sidney Smith Hall, 100 St. George Street, Room 5047, Toronto M5S 3G3, Ontario, Canada

A B S T R A C T

We describe a method for retroactively improving the accuracy of a General Transit Feed Specification (GTFS) package by using a real-time vehicle location data set provided by the transit agency. Once modified, the GTFS package contains the observed rather than the scheduled transit operations and can be used in research assessing network performance, reliability and accessibility. We offer a case study using data from the Toronto Transit Commission and find that substantial aggregate accessibility differences exist between scheduled and observed services. This 'error' in the scheduled GTFS data may have implications for many types of measurements commonly derived from GTFS data.

1. Introduction

Over the last ten years, the General Transit Feed Specification (GTFS) has emerged as an industry standard for publishing data about transit operations. Data in this format has issued from more than a thousand transit agencies around the world and that data has been incorporated into just as many user-facing routing applications. GTFS data defines transit schedule information in a format that is essentially a routable spatiotemporal network graph with stops as nodes, scheduled travel between stops as edges, and estimated travel times as the cost. This not only allows people to find their way from A to B, but due to the open nature of the standard, has allowed researchers to ask interesting questions and have them answered with a degree of accuracy and scope that would have been impossible before GTFS. Such questions, still very much under active research, include measures of disparities in service provision (Farber et al., 2016; Fransen et al., 2015), temporal variability (Farber et al., 2014), the role of relative travel times and costs in mode choice (Owen and Levinson, 2015; Salonen and Toivonen, 2013), the degree of accessibility offered by competing transit development plans (Farber and Grandez, 2017), and many others. Yet, such research using GTFS is subject to a serious criticism: it is based entirely on schedules, which are expectations about services, rather than observations of them. It is common knowledge that transit does not run precisely as scheduled, and that it often differs substantially from the schedule. Occasionally there are major unscheduled disruptions due to severe congestion, vehicle breakdowns, or signal malfunctions. These disruptions are a fact of life for most transit users and well acknowledged by transit agencies themselves.

One way that agencies have acknowledged service delays and disruptions is to issue live updates to their transit schedules. While for some smaller agencies, these take the form of hastily posted signs and twitter notifications, for many larger operations, the effort to update their customers has become perpetual. We refer here to the "real-time" data sources that contain constantly updated arrival predictions as well as live vehicle location reports based on Geographic Positioning Systems (GPS). These algorithmically produced updates have proven quite useful to many transit users (Brakewood et al., 2015; Tang and Thakuriah, 2012; Watkins et al., 2011), and the automatic vehicle location (AVL) systems that enable them have proven useful to transit researchers who have primarily used them to assess reliability at the level of stops or lines (e.g. El-Geneidy et al., 2011; Tribone et al., 2016) or to propose ways for transit operators to improve reliability or prediction accuracy. Yet transportation researchers have not yet to our knowledge made use of such data to answer research questions that involve routing across the transit network, a type of query enabled by GTFS data.

In this paper, we propose a novel way of making this "real-time" data available to answer precisely the same sorts of research questions that people have already been asking of and answering with GTFS. We do this by using it to update an existing, schedule-based GTFS package with observed trips and arrival times, yielding a GTFS package based on observation rather than schedule. Of course, such data cannot be used directly for routing actual passengers since the events it describes will already have transpired. Yet, most research questions currently answered by GTFS data might be better directed not at a schedule but at a measure of average performance which could be derived from past

* Corresponding author.
E-mail address: nate.wessel@mail.utoronto.ca (N. Wessel).

http://dx.doi.org/10.1016/j.jtrangeo.2017.04.012
Received 18 January 2017; Received in revised form 26 April 2017; Accepted 26 April 2017
Available online 09 June 2017

events, or at the events of some particular day or time in the past.

In the remainder of this paper, we will describe our method for creating this retrospective GTFS package from GTFS and real-time data. We will then undertake a brief case study of the Toronto Transit Commission to illustrate the potential utility of this approach. Because real-time data is currently not well standardized, we will have to describe our algorithm to some extent as it was designed around a particular standard - the data available from the NextBus company's API (NextBus, 2016) which is used in Toronto and many other cities. The code we developed is available online,[1] and we encourage others to use and contribute to the project. Work to extend the code to accommodate other real-time data standards is underway, and we will try to describe the project here with a degree of abstraction sufficient to allow the reader to see how the technique can be applied to any real-time data standard.

2. Required data

Our method relies upon three data sets:

1. A current GTFS package, which will be used primarily to provide the locations of stops and stations.
2. A real-time feed of vehicle locations which are preferably associated with some route information.
3. A routable road and/or rail network data set including all ways used by transit vehicles.

The method we have developed is based on the information provided by a basic GTFS package, and the real-time information provided by the API delivered by the NextBus company, which currently serves data for about 50 agencies around the world (NextBus, 2016). Other real-time data formats exist, and real-time data has not yet standardized to the same degree as schedule data; some agencies provide different information in their feeds. We believe that our algorithm should be generalizable to most real-time data feeds, so long as they provide accurate vehicle locations and update them at a sufficient frequency.

3. Algorithm

3.1. Outline

The basic ontological units of a GTFS package are routes, trips, blocks, and stops. Routes are sets of typical service patterns grouped under a single name, usually a number. Trips are particular occasions when a vehicle goes from one end of a route to the other, serving an ordered set of stops. Blocks are sets of trips in the order of their performance, operated continuously by the same vehicle.[2] Stops are the locations where passengers can access a trip.

Real-time data often does not describe the transit system in the same terms as does GTFS; it is not designed to provide a routable network, but to give updates on particular vehicles and lines. The task of our algorithm then must be to translate the description provided by the real-time data into the terms/units used by GTFS. In the real-time data provided by NextBus for example, there is no explicit concept of trips or blocks and these must be inferred. The focus of the NextBus data is on vehicles and stops.

Our basic approach is to monitor vehicles in real-time as their locations are updated, keeping track of where they are and when. When we observe that a vehicle seems to have completed a trip, we process the data associated with that trip and begin building up a new trip for the vehicle if it is still in service. Finished trips are associated with stops from the schedule data and stop times are derived from the timestamps of vehicles passing those stops. Next, trips are assigned to blocks and routes and the data is stored in a database. At this point, it is straightforward to extract the data in the text-based GTFS format. Much of the hard work of the algorithm is in error handling and data cleaning as reported vehicle locations are always a bit fuzzy and occasionally spurious.

3.2. Collecting and storing the data

We developed a program in Python to collect real-time data from the NextBus API, which is used by the Toronto Transit Commission. The NextBus API is a publicly available web service designed to serve real-time transit data primarily to mobile phone or web applications. One of its functions is to report the latest locations for all operating vehicles in the fleet, the idea being that these will be displayed to users on a mobile web map. For each vehicle, the API returns information on the vehicle's ID, route, heading, last known location, and the time it reported that location to the server. Vehicles update their location every 20 s on average.

Our program requests all updated vehicle locations every 10 s to ensure that all new data is collected and then stores the locations in a PostGIS database along with a timestamp and the other associated information.

3.3. Delimiting trips and blocks

We assume that each vehicle reported to be in service is operating a trip and that that trip belongs to a schedule block. Each vehicle under observation must be assigned a unique trip id and a unique block id. A block ends only when the vehicle goes out of service, defined as failing to report a location for three or more minutes (or some other threshold, as appropriate). The NextBus API did not explicitly report when a vehicle was going out of service. The end of a schedule block implies the end of any ongoing trip, but a trip may also end in a number of other conditions. For the data available to us from NextBus, we defined these as

1. The vehicle reporting a different route.
2. The vehicle reporting a different headsign.

Headsigns typically indicate the direction of travel, i.e. “501 E” and a change to “501 W” implies that the terminal station has been reached and the return trip begun in the opposite direction. With the NextBus data, we were fortunate that the headsign and route changes were a good indicator of trip endings. Other real-time or historical data sets could necessitate other techniques, perhaps simpler, perhaps not, for determining when trips are ending. For pure GPS data, it may be necessary to determine when a vehicle turns back on its route to go the other way.

Once a trip has ended, all information associated with that trip is pulled from the database and processed, as the rest of this section will describe. If the vehicle is still in service, a new trip is started for that vehicle, and new location reports begin to accumulate in the database associated with the new trip ID.

3.4. Spatial matching and positional error handling

Visual inspection of the data provided by the NextBus API showed normal, expected levels of GPS position error for most points, but also some extreme errors associated with trips either starting or ending in a bus garage, where vehicles reported positions many kilometres from the last reported location before going out of service. The extreme errors were easily dealt with by measuring the speed (*distance/time*) between location reports. Segments over 120 km/h indicated obvious errors and the offending points were removed accordingly. As these were almost always associated with vehicles entering or leaving service, this was not

[1] https://github.com/SAUSy-Lab.

[2] Passengers can often stay on the vehicle as it transitions between trips.