Playing with Cincinnati Metro (SORTA) data

CincyPy Presentation; 4.16.2012

Blaine Booher

Intro

Blaine Booher

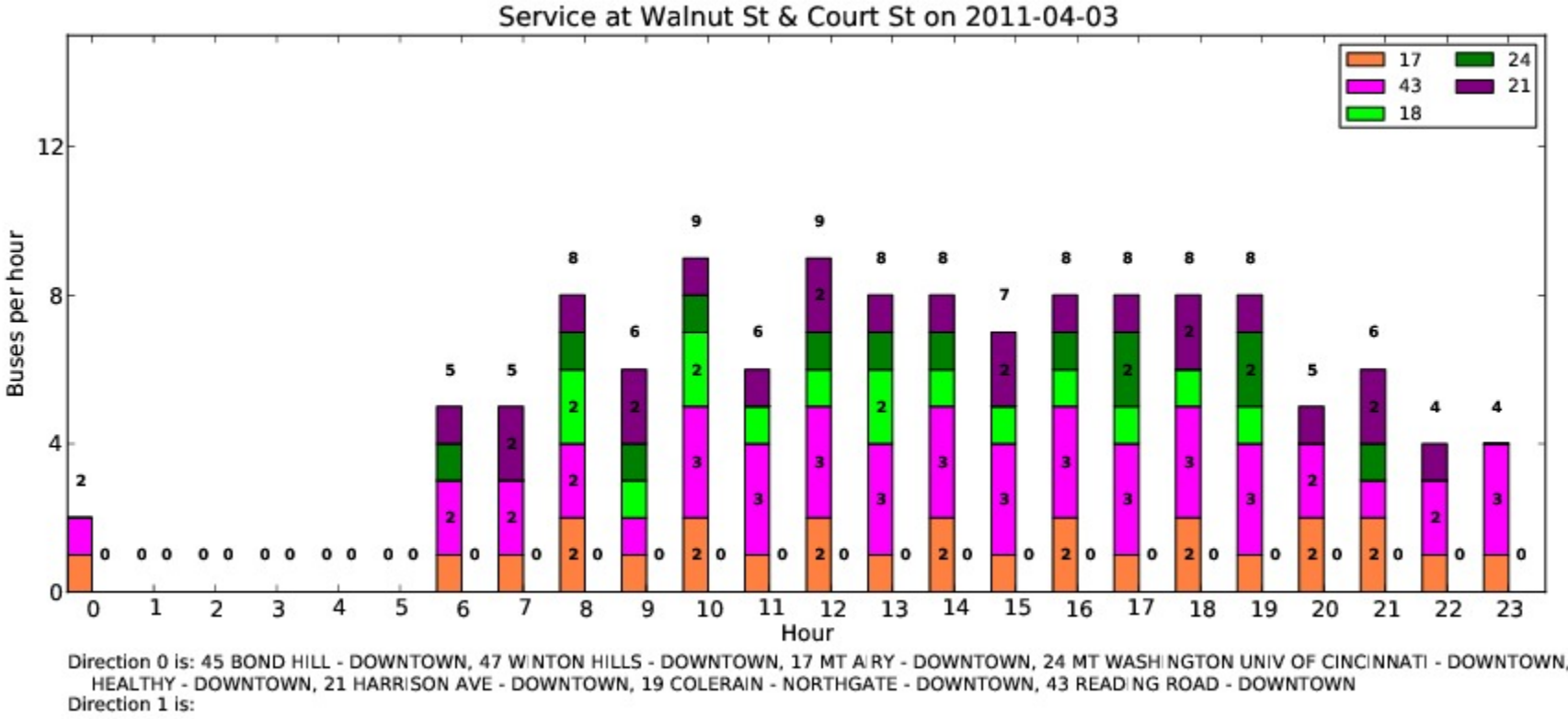
blaine@cliftonlabs.com

I am a computer engineer who loves working with python. I also love hardware & web applications. Currently looking for new clients and/or partners for projects ;)

Today we're going to talk about how to use python in the context of a practical application

I'll cover my process, questions asked, software downloaded, and we'll end up with a graph

code available here: **http://github.com/booherbg/SORTA**



Are you excited yet??

Background

In early April, SORTA released a bunch of data

- Available on their website
- Google Transit Feed Spec format (GTFS)

As a developer, this is music to my ears!

- It's new!
- It's standards compliant!
- It's relevant to my city! (Cincinnati, OH)
- It's free!
-

Available as a zip file: http://www.go-metro.com/about-metro/developer-data

What to do now?

So now we have this data... what to do?

The hard way:

Reverse engineer it. Write parsers, wrappers, boilerplate... Dive right in and try to make sense of it all without looking at it in context.

The easy way: Python! (Assuming this data does not exist in a vacuum)

Python?

OK it's true. Python doesn't inherently make it easy

But the python community, and the spirit of the python development process are where the magic happens...

What tools already exist? The question is "How far have other python guys gotten?"

I always start a new project by googling around for python tools. Odds are someone else has tried it. The Internet is Amazing.

The search

As always, this project started with me having no idea whatsoever what I'm doing.

The key: data standards

The most important thing is that we now have a bunch of data that is in a published format.

Google Transit Feed Specification (GTFS).

I contacted SORTA and the spokesperson said this alone took like two years to get operational

But there's a reason that specifications exist...

GTFS

This is a data format used by Google to manage their transit applications, like traffic, bus & train routes, and google maps integration

Here's the published spec: https://developers.google.com/transit/gtfs/reference

- Common data interfaces:
  - Stops (Physical locations associated with a route)
  - Routes (Names & unique IDs of bus routes)
  - Trips (Bi-directional configurations: Map Stops to Routes)
  - Agency (information on SORTA)
  - Shapes (physical route configuration, with GPS data)
  - Calendar (Certain trips valid on weekdays vs weekends)
- Available in plain text, CSV format
- Support for many optional parameters that Cincinnati doesn't use/need
- Tools suite: http://code.google.com/p/googletransitdatafeed/wiki/ScheduleViewer

Cincy Metro Data

The Sorta data is available as a zip file from
http://www.go-metro.com/uploads/google_transit.zip

```
-rw-r--r-- 1 180 14:30  agency.txt      # Metro Info
-rw-r--r-- 1 194 14:30  calendar.txt    # weekday/weekend info
-rw-r--r-- 1 5.6K 14:30 routes.txt      # route name info
-rw-r--r-- 1 7.4M 14:30 shapes.txt      # GPS shapes
-rw-r--r-- 1 340K 14:30 stops.txt       # all of the stops
-rw-r--r-- 1 17M 14:30  stop_times.txt # stop times (details)
-rw-r--r-- 1 244K 14:30 trips.txt       # routes :: stops map
```

What I found

First, I found a blog post of someone who blazed a trail:
http://www.kurtraschke.com/2011/05/python-gtfs-data-visualization

Second, I found a GTFS database mapping suite (amazing, really):
https://github.com/bmander/gtfs

Third, I found a visualization tool that let's you view the maps (amazingx2):
http://code.google.com/p/googletransitdatafeed/wiki/ScheduleViewer

This alone will get us pretty far... so how do we go about using these tools?

Dependencies

We should probably install the following:

- Python2.6 or Python2.7
- matplotlib & numpy
  - sudo apt-get install python-matplotlib python-numpy
- SqLite3 wrappers for python
  - should be included in python's library, depending on your distribution
  - sudo apt-get install python-sqlite

I also recommend iPython if you haven't installed it. The best part about ipython is tab completion at the command prompt, perfect for exploring unfamiliar data structures...

apt-get install ipython

easy_install ipython

Git set up

I have almost everything in my github (minus matplotlib).

git clone https://github.com/booherbg/SORTA.git

I even forked the visualization tool (tph) because it was python2.7 compatible only. I had to backport two functions to make it python2.6 compatible. Sounds complicated... it's not. See backports.py

GTFS tool:

git clone https://github.com/kurtraschke/gtfs.git

cd gtfs && python setup.py install

The Google Transit tools were very helpful (for visual aid)

easy_install transitfeed

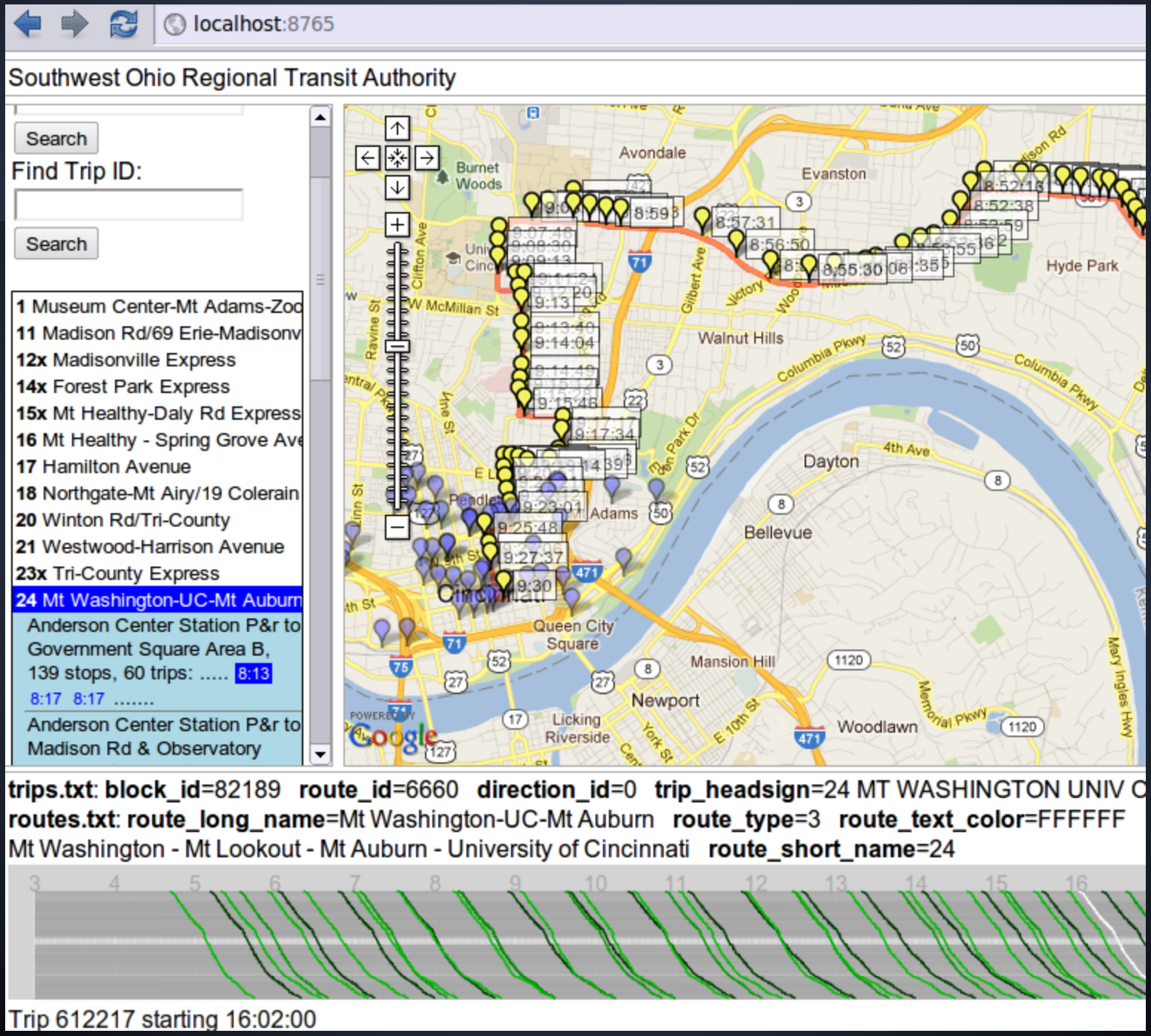
Google's TransitFeed Tools

I haven't played much with Google's tools, but this one is really neat. It allows you to play with the data before you even get your hands dirty.

It's also written in python :)

After installation, you can run the following to get a mashup of your transit data against a google map:

```
[blaine@macbook:/data/cliftonlabs/SORTA Mon Apr 16]
45$ schedule_viewer.py ./cincySorta_GTFS.zip
Loading data from feed "./cincySorta_GTFS.zip"...
(this may take a few minutes for larger cities)
To view, point your browser at http://localhost:8765/
```


localhost:8765
Southwest Ohio Regional Transit Authority
Search
Find Trip ID:
Search
1 Museum Center-Mt Adams-Zoo
11 Madison Rd/69 Erie-Madisonv
12x Madisonville Express
14x Forest Park Express
15x Mt Healthy-Daly Rd Express
16 Mt Healthy - Spring Grove Ave
17 Hamilton Avenue
18 Northgate-Mt Airy/19 Colerain
20 Winton Rd/Tri-County
21 Westwood-Harrison Avenue
23x Tri-County Express
24 Mt Washington-UC-Mt Auburn
Anderson Center Station P&r to
Government Square Area B,
139 stops, 60 trips: 8:13
8:17 8:17 .....
Anderson Center Station P&r to
Madison Rd & Observatory
Avondale
Burnet Woods
Evanston
Hyde Park
Walnut Hills
Dayton
Bellevue
Adams
Queen City Square
Mansion Hill
Newport
Licking Riverside
Woodlawn
Google
trips.txt: block_id=82189 route_id=6660 direction_id=0 trip_headsign=24 MT WASHINGTON UNIV C
routes.txt: route_long_name=Mt Washington-UC-Mt Auburn route_type=3 route_text_color=FFFFFF
Mt Washington - Mt Lookout - Mt Auburn - University of Cincinnati route_short_name=24
Trip 612217 starting 16:02:00

Break Time

Let's take a break?

GTFS tool

The GTFS tool is written by bmander and provides a clean python wrapper around an SQLAlchemy interface.

https://github.com/bmander/gtfs

It also provides you with the ability to generate a relational database one time for fast processing later.

It's fantastic, really. If you are using ipython, you can explore the data interactively while you develop. It's the best way to learn in my opinion!

Creating the database

So after poking around I found a gtfs tool. It takes the GTFS data, processes it, and throws it into an SQLite database. It took about 20 minutes to create this database, so I've provided it for you via my git repo.

To create your own database: **compile_gtfs -o sorta.db ./google_transit.zip**

>>> import gtfs

>>> sched = gtfs.load("bart.zip") # takes a few minutes to load if you haven't created the database

>>> sched.stops

[<Stop POWL>, <Stop ROCK>, <Stop SANL>, <Stop SHAY>, <Stop SSAN>, <Stop UCTY>, <Stop WCRK>, <Stop WOAK>, <Stop COLM>, <Stop MLBR>, <Stop RICH>,.<Stop NCON>, <Stop ORIN>, <Stop PHIL>, <Stop BAYF>, <Stop CAST>, <Stop CIVC>]

>>> sched.routes

[<Route AirBART>, <Route 01>, <Route 02>, <Route 05>, <Route 06>, <Route 07>, <Route 08>, <Route 11>, <Route 12>]

>>> sched.service_periods

[<ServicePeriod WKDY 1111100>, <ServicePeriod SAT 0000010>, <ServicePeriod SUN 0000001>, <ServicePeriod M-FSAT 1111110>, <ServicePeriod SUNAB 0000001>]

>>> # pretty much all relationships are mapped

>>> sched.stops[0].stop_times[0].trip.route.agency

<Agency BART>

Now we're ready to go...

We now have the GTFS data, it is compiled into a neat relational database, we have a visual tool to help find interesting data points. What next?

This brings us to tph's visualization tool (included in my github repo). Originally from:

http://www.kurtraschke.com/2011/05/python-gtfs-data-visualization

How to use it

The tph project is a simple little visualization tool written by Kurt Raschke. It interfaces with the GTFS relational database which does the heavy lifting.

It generates a bar plot that shows activity at a stop with various routes...

It's the result of one guy's exploration of the data and only a single thing of the many possible things that you can use the data for...

Once you have a config file written (defining what stops you want to analyze), it's easy:

python tph/tph.py sorta2.cfg ./sorta.db sorta2.pdf

My config file

My config file is a single stopID along with route pairings. it generates the graph we saw at the beginning of this presentation (Walnut & Court Street)

4$ cat sorta2.cfg

[config]

target_date: 2011-04-03

[route1]

target_routes: 6654, 6676, 6655,6660,6658

target_stopid: 1180880

outfile: sorta1.pdf

Scripting against the database

Before finding the visualization tool, I didn't actually know what a good stop would be. I wanted to find the highest volume stop, aka "Which stop has the most routes associated with it?".

With iPython to help me poke around, and tph's script as guidance, it turns out this is not a hard question to answer.

I found the following stops with lots of activity:

[(721, <Stop 1180880>),
(691, <Stop 2430050>),
(680, <Stop 2851845>),
(650, <Stop 1161160>),
(632, <Stop 2240010>)]

See next slide for the code.

```
from gtfs.entity import *
from gtfs import Schedule

gtfs_db = './sorta.db'
schedule = Schedule(gtfs_db, echo=False)

def find_highest_volume_stop(NUM_STOPS=5):
    nstops = list()
    for i in xrange(len(schedule.stops)):
        stop = schedule.stops[i]
        nstops.append((len(stop.stop_times), stop))
        nstops.sort()
        nstops.reverse()
        if len(nstops) > NUM_STOPS:
            nstops = nstops[0:NUM_STOPS]
        if i % 20 == 0:
            print "(%.2f%%) %d / %d (highest so far: %d; id=%d)" \
                % ((float(i) / len(schedule.stops))*100, i, len(schedule.stops), nstops[0][0], nstops[0][1].stop_id)
    return nstops
```

Pretty easy right?

Other than my laughable "sorting" algorithm, the code is tight and works great. You can even speed it up if you used pypy!

Finding info about a stop

Inside tph/ I have a small file that shows how you can find information about a stop.

python tph/stop_info.py 1180880 ./sorta.db
Stop name: Walnut St & Court St
There are 721 trips for this stop (not shown)

Routes that this stop is on:
Route 21: Westwood-Harrison Avenue (6658)
Route 17: Hamilton Avenue (6654)
Route 74x: Northgate Express (6690)
Route 43: Reading/47 Winton Hls/45 Bond Hl (6676)
Route 24: Mt Washington-UC-Mt Auburn (6660)
Route 20: Winton Rd/Tri-County (6657)
Route 71x: Kings Island Express (6689)

```
from gtfs.entity import *
from gtfs import Schedule
import sqlalchemy.orm.exc, os, sys

schedule = Schedule('./sorta.db', echo=False)
if __name__ == '__main__':
    stop_id = int(sys.argv[1])
    try:
        stop = Stop.query.filter_by(stop_id=stop_id).one()
    except sqlalchemy.orm.exc.NoResultFound: #found by using invalid stop
        print "stop id %d not found" % (stop_id)
        sys.exit(1)

    trips, routes = set(), set() # only use unique values
    for time in stop.stop_times:
        trips.add(time.trip)
        routes.add(time.trip.route)
    print "Stop name: %s" % stop.stop_name
    print "There are %d trips for this stop (not shown\n)" % len(trips)
    print "Routes that this stop is on:"
    for route in routes:
        print "Route %s: %s (%d)" \
          % (str(route.route_short_name), str(route.route_long_name), int(route.
     route_id))
```

Appendix A: Backporting to Py2.6

The original tph tool was only compatible with Python2.7... So I backported it after a painstakingly long process of trying to track down an entire suite of Python2.7 tools =\

Had to create a "Counter" and an "OrderedList" from the collections library.

Counter was easy:

```
from collections import defaultdict

# Counter
def factory():
    return defaultdict(int)
Counter = factory
```

OrderedDict

OrderedDict, on the other hand, was a piece of work. I found it on a stackoverflow thread. See backports.py on the implementation.

It's basically just a dictionary that remembers the order that things are added. Essentially a queue of hashable sets?

Questions?

Now's the time to go off on your own. Good luck!

I hope this post has inspired you. Feel free to ask me questions at any time.

blaine@cliftonlabs.com