OpenStreetMap Data Case Study

Map Area

Charlotte, NC, United States

- https://www.openstreetmap.org/relation/177415
- http://metro.teczno.com/#charlotte

This map is of my hometown, so I'm more interested to see what database querying reveals, and I'd like an opportunity to contribute to its improvement on OpenStreetMap.org.

Problems Encountered in the Map

After initially downloading a small sample size of the Charlotte area and running it against a provisional data.py file, I noticed five main problems with the data, which I will discuss in the following order:

- Overabbreviated street names *("S Tryon St Ste 105")*
- Inconsistent postal codes *("NC28226", "282260783", "28226")*
- "Incorrect" postal codes (Charlotte area zip codes all begin with "282" however a large portion of all documented zip codes were outside this region.)
- Second level `"k"` tags with the value `"type"` (which overwrites the element's previously processed `node["type"]field`).
- Street names in second level `"k"` tags pulled from Tiger GPS data and divided into segments, in the following format:

```
<tag k="tiger:name_base" v="Stonewall"/>
<tag k="tiger:name_direction_prefix" v="W"/>
<tag k="tiger:name_type" v="St"/>
```

Overabbreviated Street Names

Once the data was imported to SQL, some basic querying revealed street name abbreviations and postal code inconsistencies. To deal with correcting street names, I opted not use regular expressions, and instead iterated over each word in an address, correcting them to their respective mappings in audit.py using the following function:

```
def update(name, mapping):
    words = name.split()
    for w in range(len(words)):
        if words[w] in mapping:
            if words[w•1].lower() not in ['suite', 'ste.', 'ste']:
                # For example, don't update 'Suite E' to 'Suite East'
                words[w] = mapping[words[w]] name = " ".join(words)
    return name
```

This updated all substrings in problematic address strings, such that:
"S Tryon St Ste 105"
becomes:
"South Tryon Street Suite 105"

Postal Codes

Postal code strings posed a different sort of problem, forcing a decision to strip all leading and trailing characters before and after the main 5digit zip code. This effectively dropped all leading state characters (as in "NC28226") and 4digit zip code extensions following a hyphen ("28226-0783"). This 5digit restriction allows for more consistent queries.

Regardless, after standardizing inconsistent postal codes, some altogether "incorrect" (or perhaps misplaced?) postal codes surfaced when grouped together with this aggregator:

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
      UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key='postcode'
GROUP BY tags.value
ORDER BY count DESC;
```

Here are the top ten results, beginning with the highest count:

```
value|count
28205|900
28208|388
28206|268
28202|204
28204|196
28216|174
28211|148
28203|120
28209|104
28207|86
```

These results were taken before accounting for Tiger GPS zip codes residing in second level “k” tags. Considering the relatively few documents that included postal codes, of those, it appears that out of the top ten, seven aren’t even in Charlotte, as marked by a “#”. That struck me as surprisingly high to be a blatant error, and found that the number one postal code and all others starting with“297”lie in Rock Hill, SC. So, I performed another aggregation to verify a certain suspicion...

Sort cities by count, descending

```
sqlite> SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key LIKE '%city'
GROUP BY tags.value
ORDER BY count DESC;
```

And, the results, edited for readability:

```
Rock Hill   111
Pineville   27
Charlotte   26
York        24
Matthews    10
Concord     4
3000        3
10          2
Lake Wylie  2
1           1
3           1
43          1
61          1
Belmont, N  1
Fort Mill,  1
```

These results confirmed my suspicion that this metro extract would perhaps be more aptly named “Metrolina” or the “Charlotte Metropolitan Area” for its inclusion of surrounding cities in the sprawl. More importantly, three documents need to have their trailing state abbreviations stripped. So, these postal codes aren’t “incorrect,” but simply unexpected. However, one final case proved otherwise.

A single zip code stood out as clearly erroneous. Somehow, a “48009” got into the dataset. Let’s display part of its document for closer inspection (for our purposes, only the “address” and “pos” fields are relevant):

```
sqlite> SELECT *
FROM nodes
WHERE id IN (SELECT DISTINCT(id) FROM nodes_tags WHERE key='postcode'
AND value='48009')
```

`1234706337|35.2134608|-80.8270161|movercash|433196|1|7784874|2011-04-06T13:16:06Z`

`sqlite> SELECT * FROM nodes_tags WHERE id=1234706337 and type='addr';`

```
1234706337|housenumber|280|addr
1234706337|postcode|48009|addr
1234706337|street|North Old Woodward Avenue|addr
```

It turns out, *"280 North Old Woodward Avenue, 48009"* is in Birmingham, Michigan. All data in this document, including those not shown here, are internally consistent and verifiable, except for the latitude and longitude. These coordinates are indeed in Charlotte, NC. I'm not sure about the source of the error, but we can guess it was most likely sitting in front of a computer before this data entered the map. The document can be removed from the database easily enough.

Data Overview and Additional Ideas

This section contains basic statistics about the dataset, the MongoDB queries used to gather them, and some additional ideas about the data in context.

File sizes

```
charlotte.osm ......... 294 MB
charlotte.db .......... 129 MB
nodes.csv ............. 144 MB
nodes_tags.csv ........ 0.64 MB
ways.csv .............. 4.7 MB
ways_tags.csv ......... 20 MB
ways_nodes.cv ......... 35 MB
```

Number of nodes

```
sqlite> SELECT COUNT(*) FROM nodes;
```

1471350

Number of ways

```
sqlite> SELECT COUNT(*) FROM ways;
```

84502

Number of unique users

```
sqlite> SELECT COUNT(DISTINCT(e.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;
```

337

Top 10 contributing users

```
sqlite> SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
GROUP BY e.user
ORDER BY num DESC
LIMIT 10;
```

```
jumbanho    823324
woodpeck_f  481549
TIGERcnl    44981
bot-mode    32033
rickmastfa  18875
Lightning   16924
grossing    15424
gopanthers  14988
KristenK    11023
Lambertus   8066
```

Number of users appearing only once (having 1 post)

```
sqlite> SELECT COUNT(*)
FROM
    (SELECT e.user, COUNT(*) as num
     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
     GROUP BY e.user
     HAVING num=1)  u;
```

56

Additional Ideas

Contributor statistics and gamification suggestion

The contributions of users seems incredibly skewed, possibly due to automated versus manual map editing (the word “bot” appears in some usernames). Here are some user percentage statistics:

- Top user contribution percentage (“jumbanho”) 52.92%
- Combined top 2 users' contribution (“jumbanho” and “woodpeck_fixbot”) 83.87%
- Combined Top 10 users contribution
  94.3%
- Combined number of users making up only 1% of posts 287 (about 85% of all users)

Thinking about these user percentages, I’m reminded of “gamification” as a motivating force for contribution. In the context of the OpenStreetMap, if user data were more prominently displayed, perhaps others would take an initiative in submitting more edits to the map. And, if everyone sees that only a handful of power users are creating more than 90% a of given map, that might spur the creation of more efficient bots, especially if certain gamification elements were present, such as rewards, badges, or a leaderboard.

Additional Data Exploration

Top 10 appearing amenities

```
sqlite> SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
```

```
place_of_worship  580
school            402
restaurant        80
grave_yard        75
parking           63
fast_food         51
fire_station      48
fuel              31
bench             30
library           28
```

Biggest religion (no surprise here)

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE
value='place_of_worship') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 1;
```

```
christian   571
```

Most popular cuisines

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant')
i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC;
```

```
american    9
pizza       5
steak_hous  4
chinese     3
japanese    3
mexican     3
thai        3
italian     2
sandwich    2
barbecue    1
```

Conclusion

After this review of the data it's obvious that the Charlotte area is incomplete, though I believe it has been well cleaned for the purposes of this exercise. It interests me to notice a fair amount of GPS data makes it into OpenStreetMap.org on account of users' efforts, whether by scripting a map editing bot or otherwise. With a rough GPS data processor in place and working together with a more robust data processor similar to data.pyI think it would be possible to input a great amount of cleaned data to OpenStreetMap.org.