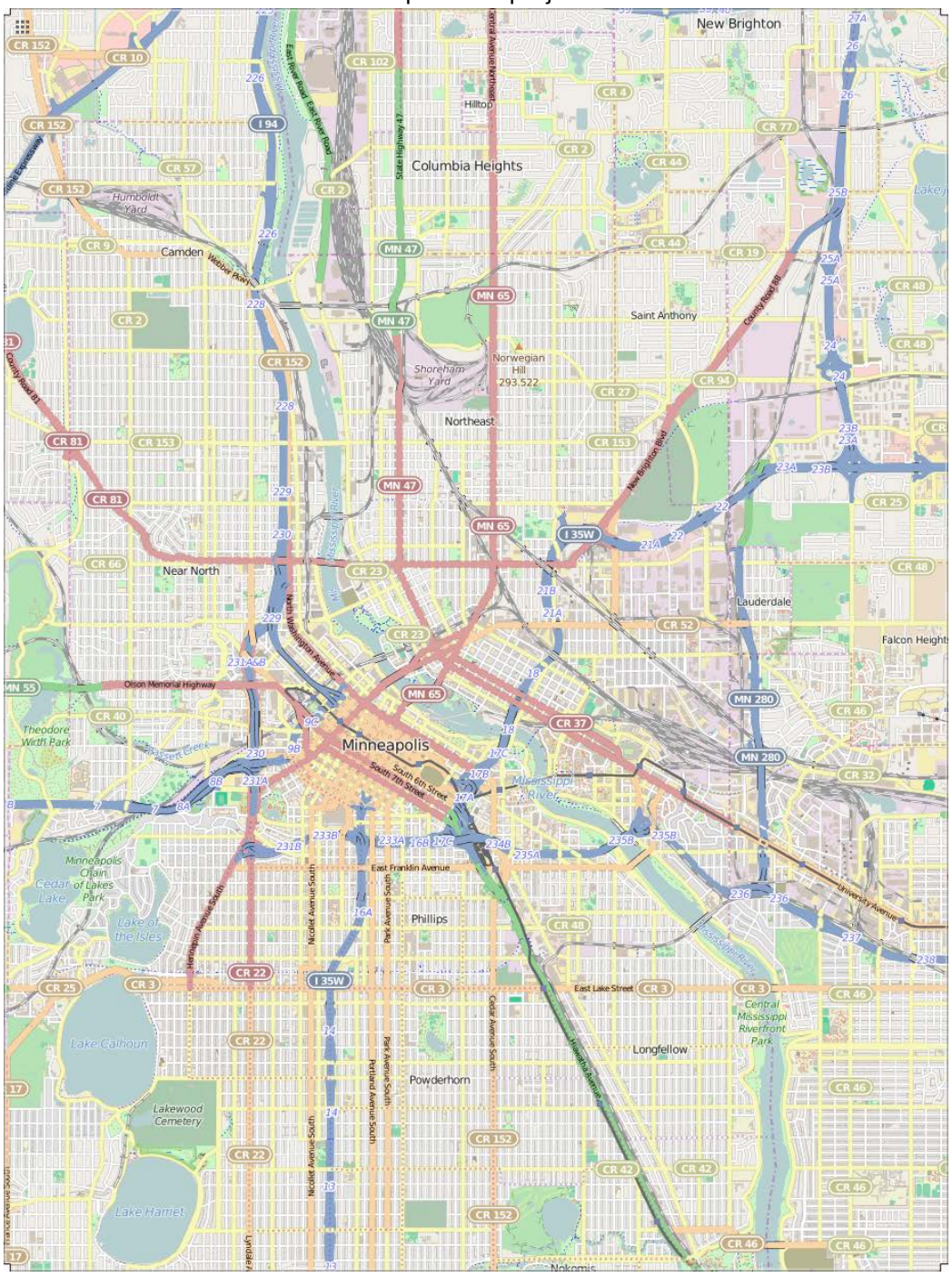
For this project I have chosen my home area Minneapolis, MN. The reason is to be better able to validate the data by cross-referencing the data with personal knowledge of the area. The area is mainly Minneapolis, MN with some of the suburbs including New Brighton, Hopkins, and Minneapolis' Twin City, St. Paul. Here is a link to the Overpass API for downloading the OSM for the region. http://overpass-api.de/api/map?bbox=-93.3292,44.9150,-93.1713,45.0644.

Below is the Map from OSM.

All the relevant code will be located in the end of this document with all the links that helped me complete the project.



1. Problems encountered in your Map
    - File-size > 87MB
        i. Even with iterative parsing, auditing and identifying meaningful patterns took longer than expected
        ii. Took a smaller sample (<12MB) for creating functions for auditing, parsing, cleaning, etc.
        iii. Once the functions are complete, they will be applied to the 87MB OSM file

- Values for cities were redundant
  i. For example—there were 4 different representations of Saint Paul. Here are the mappings

```
city_mapping = {"Minneapolis, MN" : "Minneapolis",
                "St. Anthony" : "Saint Anthony",
                "St. Paul" : "Saint Paul",
                "St Anthony" : "Saint Anthony",
                "St Paul" : "Saint Paul"}
```

- Values for postal codes ranged from being 3-digits long to 9-digits, alphabetical, or simply repeated values from different fields (*postcode = street*). One thing to note is out of the 108 unique *postcode* values, once extracting only the 5-digit prefixes, we were left with only 33 unique results which is consistent with the area I've chosen.
  i. First I chose to limit the postal codes to 5-digit lengths. This is to standardize the data by the 'least common denominator' so the data was more intuitively queryable.
  ii. The problem postal codes had letters, were too short, or misplaced from a different field. Some The only way to find the correct postal codes was to search the internet using the other fields for the search criteria and then extrapolate the postal codes. It could be done case-by-case manually but it would be faster if we could do it programmatically.
  iii. Luckily I found pygeocoder
    - Pygeocoder is a Python Library that utilizes Google's Geocoding Funcitonality to extrapolate geographic information such as coordinates, postal codes, etc.
    - I decided to use the *reverse_geocode()* function which will take the latitudinal and longitudinal coordinates and return an object from which the postal codes could be called from.

```
> from pygeocoder import Geocoder
> results = Geocoder.reverse_geocode(45.0079126, -93.2473816)
> int(results.postal_code)
> 55418
```

    - There were some *postalcode* fields with did not have coordinate systems to use. Pygeocoder also has the *geocode()* function which takes in a string argument, preferably a street address and state, and returns an object which also contains the postal code information.

```
> results = Geocoder.geocode('The Carlyle MN')
> int(results.postal_code)
> 55401
```

- Values for street suffixes were redundant
  i. The original mapping from Lesson 6 only covered a few keys to be normalized. After sifting through the data, I have discovered many more keys. Here is my mapping for street abbreviations.

```
street_mapping = { "St": "Street",
                   "St.": "Street",
                   "Ave" : "Avenue",
                   "Rd." : "Road",
                   "Rd" : "Road",
                   "SE" : "Southeast",
                   "S.E." : "Southeast",
                   "NE" : "Northeast",
                   "S" : "South",
                   "Dr" : "Drive",
                   "Rd/Pkwy" : "Road/Parkway",
                   "Ln" : "Lane",
                   "Dr." : "Drive",
                   "E" : "East",
                   "Pl" : "Plain",
                   "ne" : "Northeast",
                   "NW" : "Northwest",
                   "Ave." : "Avenue",
                   "N." : "North",
                   "W" : "West",
                   "Pkwy" : "Parkway",
                   "Ter" : "Terrace",
                   "Pky" : "Parkway",
                   "SW" : "Southwest",
                   "N" : "North",
                   "Blvd" : "Boulevard"}
```

- The original *shape_element* function created in Lesson 6, when applied to this particular OSM file resulted in 447 unique top-level descriptors. A large number of them had similar prefixes meaning they could be grouped under a single dictionary to simplify querying.

  My goal is to aggregate related descriptors into a single top-level key which can be accessed by calling that key. This will result in a more organized representation of the data. The following describes the grouped descriptors and their sources.

  i. 'address:' fields
     - These were fields dealt with in Lesson 6

  ii. 'metcouncil:' fields
     - These fields with information from the Metropolitan Council of Minnesota

  iii. 'tiger:' fields
     - **T**opologically **I**ntegrated **G**eographic **E**ncoding and **R**eferencing system (TIGER) data was produced by the US Census Bureau and later merged onto OSM database

  iv. 'metrogis:' fields
     - Regional geographic information system of the seven-county metropolitan area of Minneapolis-St. Paul, Minnesota

  v. 'umn:' fields
     - Fields related to the University of Minnesota, example.
     - This field presented a unique programmatic problem—If it existed, the positional data (Latitude,Longitude) was located in its parent element (in this case *way*), it would be located in a *umn:* k sub-element.

  vi. 'gnis:' fields
     - USGS **G**eographic **N**ames **I**nformation **S**ystem (GNIS) contains millions of names for geographic features in the US.

  After grouping I reduced the number of descriptors from 447 to 364. The fields mentioned above are important because they determine the sources of information

merged onto the OSM user-based database. Cross-referencing these fields could determine accuracy, constituency, and completeness of the data.

2. Overview of the Data
   - Statistics
     i. File Sizes
        - map.osm – 87MB
        - map.json – 95MB
     ii. Documents

```
> db.minneapolis.find().count()
> 436802
```

     iii. Nodes

```
> db.minneapolis.find({'type' : 'node'}).count()
> 373017
```

     iv. Way

```
> db.minneapolis.find({'type' : 'way'}).count()
> 59143
```

     v. Unique users

```
> len(db.minneapolis.distinct('created.user'))
> 401
```

     vi. Top 5 contributing users

```
> pipeline = [{'$group' : {'_id' : '$created.user', 'count' : {'$sum' : 1}}},
                 {'$sort' : {'count' : -1}},
                 {'$limit' : 5}]
> top5 = db.minneapolis.aggregate(pipeline)
> for i in top5['result']:
         print i['_id'], i['count']
> Mulad 146082
  iandees 81267
  stucki1 63803
  DavidF 27032
  sota767 25901
```

     vii. Top 5 contributing areas by city

     viii. Asd

3. Other ideas about the Datasets

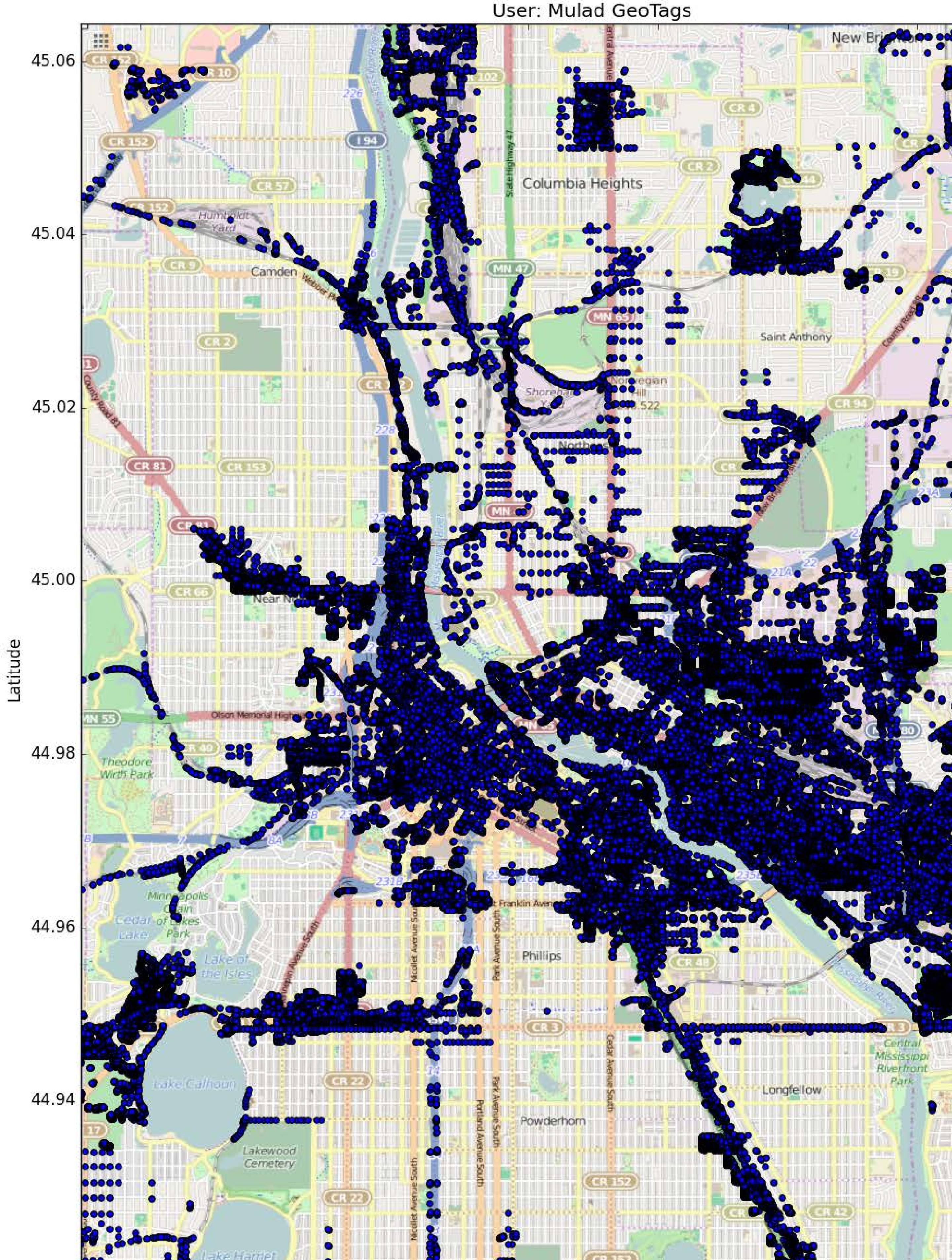
   I've decided to take the positional data (latitude, longitude) of the top 5 contributing users and create a graphical display of the places they contributed overlaid on the map area the dataset came from. This should create a sort of 'heat-map' of each users' territory, so to speak. We may be able

to see home neighborhoods or traveling patterns. The image of the top 5 contributors on to one map became too confusing to interpret so I split it into 5 individual maps/graphs. The codes for making the maps will be available in the Codes section.

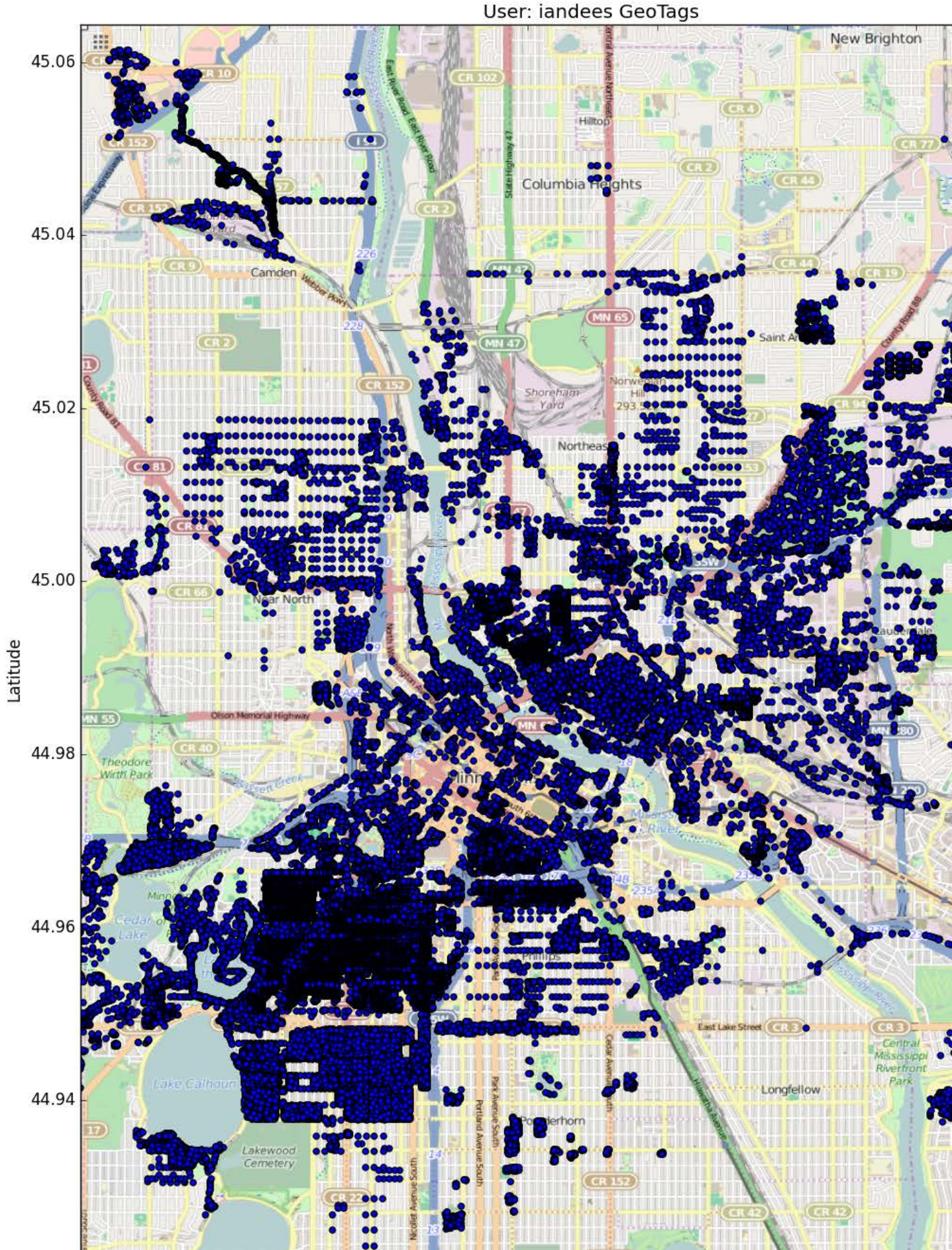
- From the previous section we have discovered the top 5 contributors to be…
  i. Mulad – 146082

```
## Query to find percentage of total documents user contributed
db.minneapolis.find({'created.user': 'Mulad'}).count() /
float(db.minneapolis.find().count())
```

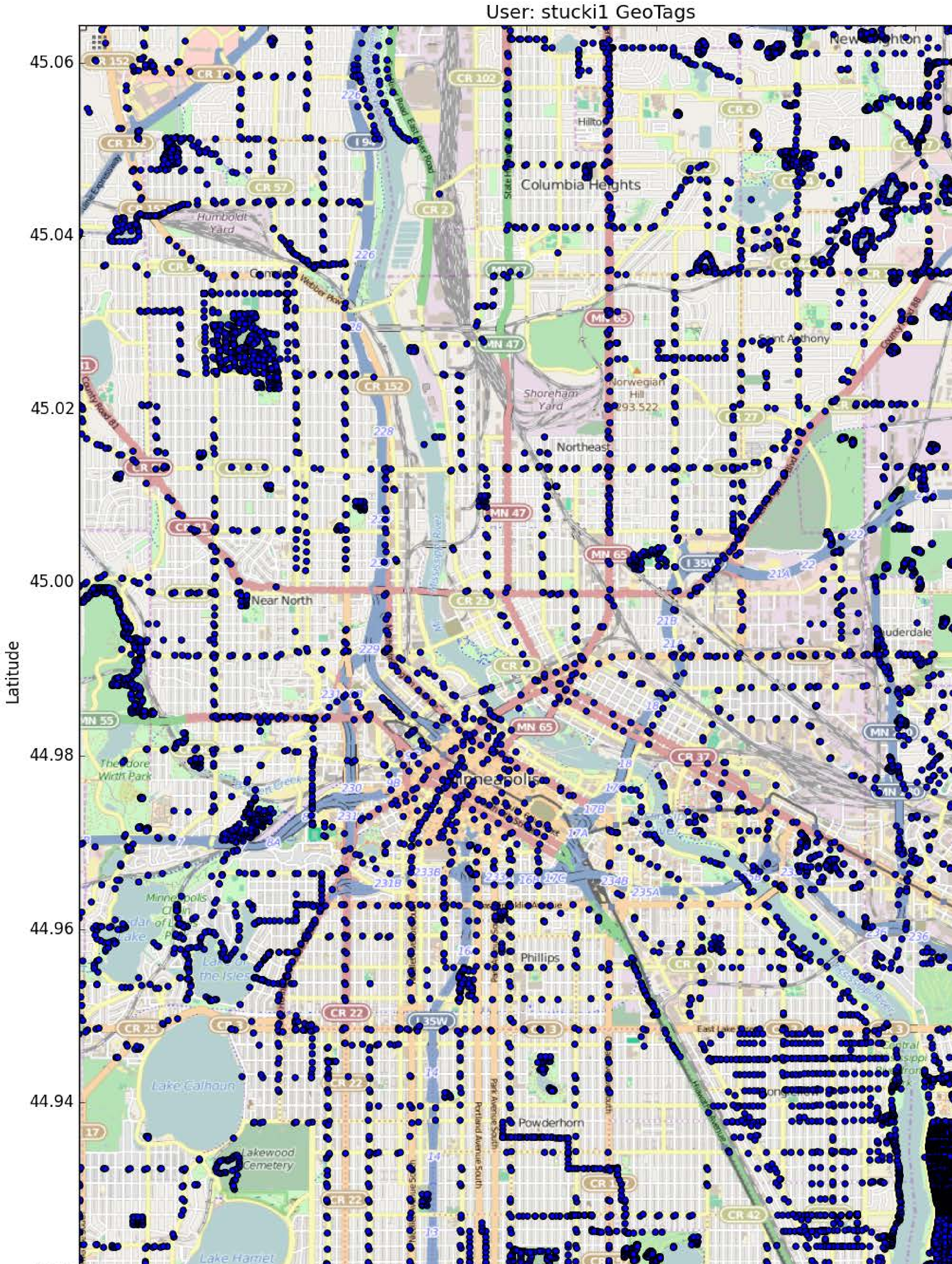
  - ~33.4 % of the total documents


User: Mulad GeoTags
Latitude
45.06
45.04
45.02
45.00
44.98
44.96
44.94
Columbia Heights
Camden
Saint Anthony
Phillips
Powderhorn
Longfellow
Theodore Wirth Park
Lake Calhoun
Lakewood Cemetery
Central Mississippi Riverfront Park

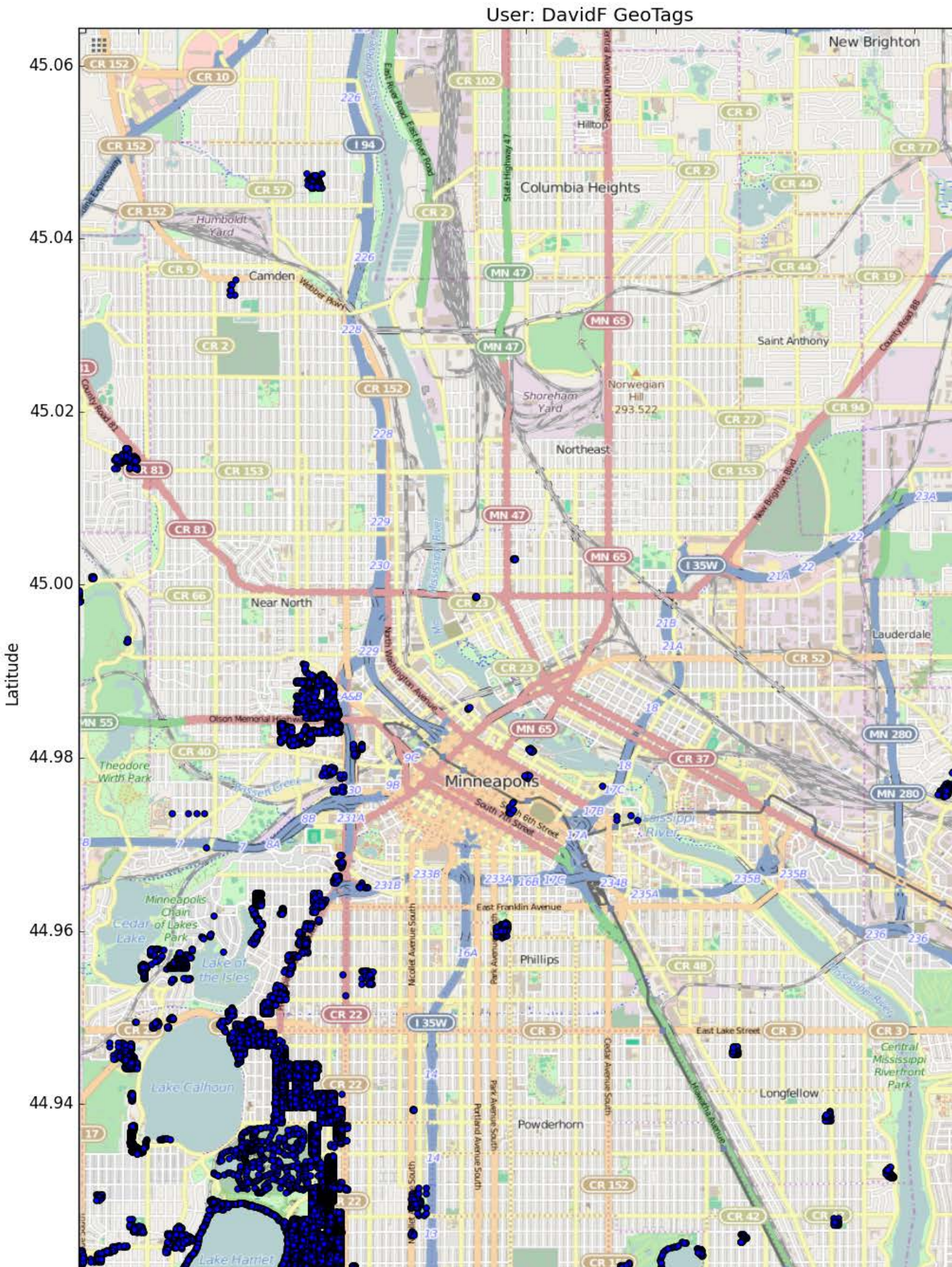
ii. iandees – 81267
    - ~18.6 % of the total documents


User: iandees GeoTags
Latitude
45.06
45.04
45.02
45.00
44.98
44.96
44.94
New Brighton
Columbia Heights
Hilltop
Camden
Shoreham Yard
Northeast
Near North
Theodore Wirth Park
Olson Memorial Highway
Lake Calhoun
Lakewood Cemetery
Phillips
Longfellow
East Lake Street
Central Mississippi Riverfront Park
Portland Avenue South
Park Avenue South
Nicollet Avenue South
Cedar Avenue South
Hiawatha Avenue
CR 10
CR 102
CR 4
CR 77
CR 152
CR 2
CR 44
CR 157
CR 9
CR 19
CR 81
CR 66
CR 40
CR 3
CR 42
MN 65
MN 47
MN 55
MN 280
I 94

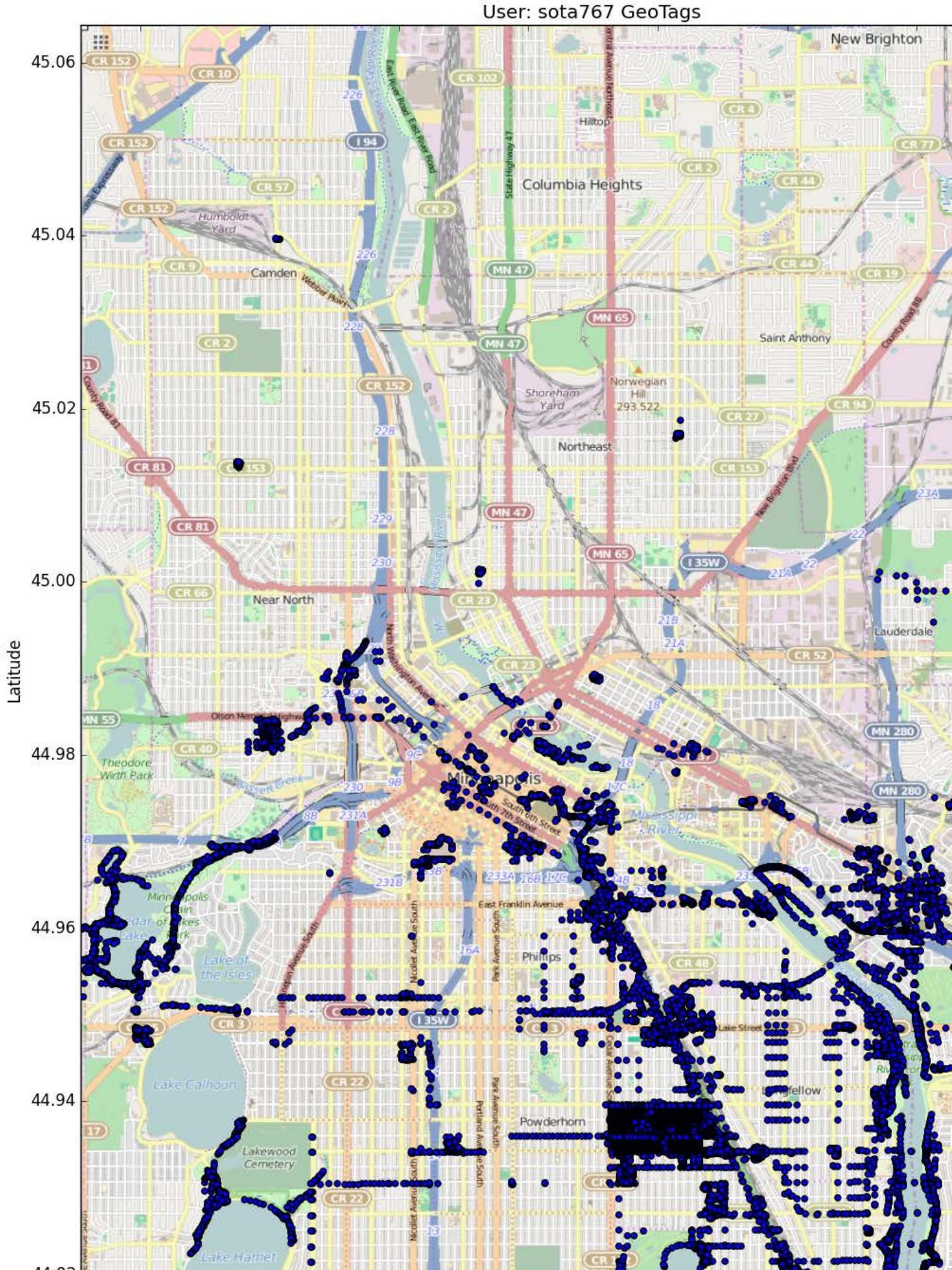
iii. stucki1 – 63803
    - ~14.6 %


User: stucki1 GeoTags
Latitude
45.06
45.04
45.02
45.00
44.98
44.96
44.94
Columbia Heights
Northeast
Near North
Phillips
Powderhorn
Lake Calhoun
Lakewood Cemetery
Shoreham Yard
Humboldt Yard

iv. DavidF – 27032
    - ~6.2 %


User: DavidF GeoTags
Latitude
45.06
45.04
45.02
45.00
44.98
44.96
44.94
New Brighton
Columbia Heights
Hilltop
Saint Anthony
Norwegian Hill
293.522
Shoreham Yard
Humboldt Yard
Camden
Northeast
Near North
Minneapolis
Lauderdale
Phillips
Powderhorn
Longfellow
Theodore Wirth Park
Minneapolis Chain of Lakes Park
Lake Calhoun
Lake of the Isles
Lake Harriet
Central Mississippi Riverfront Park
East Franklin Avenue
East Lake Street
Olson Memorial Highway
Nicollet Avenue South
Park Avenue South
Portland Avenue South
Cedar Avenue South
Hiawatha Avenue
East River Road
Webber Parkway
North Washington Avenue
New Brighton Blvd
I 94
I 35W
MN 47
MN 65
MN 280
MN 55
CR 152
CR 10
CR 102
CR 4
CR 77
CR 57
CR 2
CR 44
CR 9
CR 19
CR 81
CR 153
CR 94
CR 27
CR 66
CR 23
CR 52
CR 40
CR 37
CR 22
CR 3
CR 48
CR 42

v. sota767 – 25901
    - ~5.9


User: sota767 GeoTags
Latitude
45.06
45.04
45.02
45.00
44.98
44.96
44.94
New Brighton
Columbia Heights
Hilltop
Camden
Humboldt Yard
Shoreham Yard
Norwegian Hill
293.522
Saint Anthony
Northeast
Near North
Lauderdale
Minneapolis
Phillips
Powderhorn
Theodore Wirth Park
Lake of the Isles
Lake Calhoun
Lakewood Cemetery
Lake Harriet
Mississippi River
East Franklin Avenue
Lake Street
I 94
I 35W
MN 47
MN 65
MN 280
MN 55
CR 81
CR 152
CR 10
CR 102
CR 57
CR 2
CR 9
CR 153
CR 66
CR 23
CR 52
CR 40
CR 3
CR 22
CR 4
CR 77
CR 44
CR 19
CR 27
CR 94
CR 4B