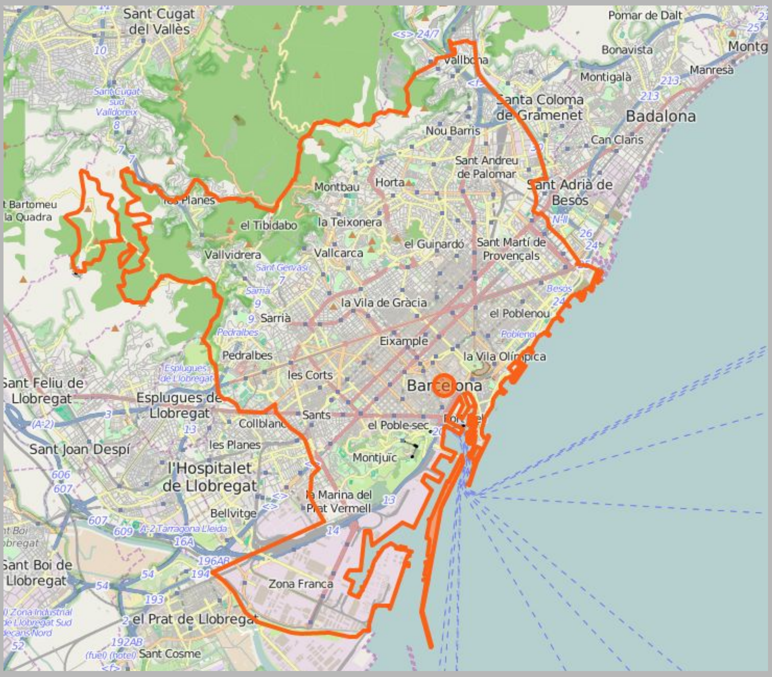
OpenStreetMap Project

Data Wrangling with MongoDB

Map Area	Barcelona, Spain
OSM link	https://www.openstreetmap.org/relation/347950
Metro link	http://metro.teczno.com/#barcelona
OSM file size	186.2Mb
	

Alex Trejo
July 2015

Auditing the data

First thing I did, was to inspect the OSM file to know the number and type of tags, and audit the tag “k” values for each "<tag>" and see if they can be valid keys in MongoDB, as well as see if there are any other potential problems. I included two more regex patterns to detect uppercase and Mixed (lower & uppercase), and another one to detect how many levels (words separated by colon) have the key.

```
{
'bounds': 1,
'member': 34272,
'nd': 1073842,
'node': 842302,
'osm': 1,
'relation': 2519,
'tag': 386630,
'way': 107462
}
```

Source: 0_tag_parsing.py

```
{
'Four Levels Colon': 148,
'Lowercase': 316614,
'Mixed': 8,
'Other': 16,
'Problemchars': 3,
'Three Levels Colon': 1005,
'Two levels Colon': 68800,
'Uppercase': 36
}
```

Source: 1_tag_audit.py

The results revealed that there are 3 keys with problematic chars and 16 other keys that can’t be classified.

- Problems:
  - `['Passeig Dos de Maig', 'Passeig Dos de Maig', 'Motor home']`
  - The three problems are the spaces. The first two are not exaclty keys, but can be values for the key "name”. The last one can be solved using an underscore.
- Other:
  - `['route_ref_1', u'L\xedneas', u'L\xedneas', 'shop_1', 'operator_1', 'name_1', 'name_1', 'craft_1', 'amenity_1', 'amenity_2', 'surface_1', 'name_1', 'name_1', 'alt_name2', 'ISO3166-2', 'ISO3166-1']`
  - These keys are in *others* because of the existence of numbers in the name. That’s not a problem.

After this initial quick audit, I decided to apply a second audit level, exploring deeper the key levels (words separated by colon). Programmatically, I passed all the tag keys, by generating dynamically a dictionary and counting each key alone or combined with one or more levels regarding the number of colons.

A problem was found here when a key appears first alone and in next steps appears combined with an other level. After the next table, I explain how I solved this problem when constructing the JSON file.

A reduced sample output for node tags is shown in the next table. The complete one can be found in the source file.

```
{'CIF': {'Total': 2},
 'FIXME': {'Total': 5},
 'Generalitat': {'Total': 1},
 u'L\xedneas': {'Total': 2},
 'Restes': {'Total': 6},
 '_description_': {'Total': 338},
 '_ref': {'Total': 1},
 'abandoned': {'Total': 4},
 'access': {'Total': 244},
 'addr': {'city': {'Total': 3276},
          'country': {'Total': 590},
          'district': {'Total': 1},
          'door': {'Total': 1},
          'floor': {'Total': 8},
          'full': {'Total': 3},
          'housename': {'Total': 115},
          'housenumber': {'Total': 4418},
          'interpolation': {'Total': 1},
          'place': {'Total': 4},
          'postcode': {'Total': 3210},
          'province': {'Total': 2},
          'state': {'Total': 61},
          'street': {'Total': 4648},
          'unit': {'Total': 14}},
 'address': {'Total': 1},
 'admin_level': {'Total': 51},
 'aerialway': {'Total': 6},
 'aerodrome': {'Total': 1},
 'aeroway': {'Total': 49},
 'alcohol': {'Total': 1},
 'alt_name': {'Total': 74},
 'altitude': {'Total': 1},
 'amenity': {'Total': 8181,
             'ca': {'Total': 1},
             'es': {'Total': 1}},
...

'recycling': {'batteries': {'Total': 43},
               'books': {'Total': 34},
               'cans': {'Total': 284},
               'cardboard': {'Total': 59},
               'cartons': {'Total': 36},
               'clothes': {'Total': 162},
               'electrical_appliances': {'Total': 34},
               'glass': {'Total': 306},
               'glass_bottles': {'Total': 35},
               'green_waste': {'Total': 117},
               'magazines': {'Total': 32},
               'newspaper': {'Total': 34},
               'organic': {'Total': 7},
               'packaging': {'Total': 2},
               'paper': {'Total': 305},
               'paper_packaging': {'Total': 34},
               'plastic': {'Total': 155},
               'plastic_bottles': {'Total': 35},
               'plastic_packaging': {'Total': 33},
               'scrap_metal': {'Total': 43},
               'small_appliances': {'Total': 33},
               'waste': {'Total': 46},
               'wood': {'Total': 34}},
 'recycling_type': {'Total': 256},
...
```

```
...
'seamark': {'beacon_cardinal': {'category': {'Total': 5},
                                 'colour': {'Total': 5},
                                 'colour_pattern': {'Total': 5},
                                 'height': {'Total': 3},
                                 'shape': {'Total': 1}},
             'buoy_safe_water': {'colour': {'Total': 1},
                                 'colour_pattern': {'Total': 1},
                                 'shape': {'Total': 1}},
             'harbour': {'category': {'Total': 1}},
             'information': {'Total': 4},
             'light': {'1': {'character': {'Total': 2},
                             'colour': {'Total': 2},
                             'group': {'Total': 1},
                             'height': {'Total': 2},
                             'period': {'Total': 1},
                             'range': {'Total': 2},
                             'sector_end': {'Total': 2},
                             'sector_start': {'Total': 2},
                             'sequence': {'Total': 2}},
                       'category': {'Total': 1},
                       'character': {'Total': 25},
                       'colour': {'Total': 25},
                       'group': {'Total': 22},
                       'height': {'Total': 24},
                       'period': {'Total': 25},
                       'range': {'Total': 24},
                       'reference': {'Total': 25},
                       'sequence': {'Total': 15}},
             'light_major': {'height': {'Total': 2}},
             'light_minor': {'height': {'Total': 13}},
             'name': {'Total': 8},
             'radar_reflector': {'Total': 2},
             'radar_transponder': {'category': {'Total': 1},
                                   'group': {'Total': 1}},
             'type': {'Total': 28}},
 'seats': {'Total': 317},
 'second_hand': {'Total': 3},
 'segregated': {'Total': 4},
...

'website': {'Total': 815, 'reviews': {'Total': 2}},
 'wetap': {'credit': {'Total': 1}, 'status': {'Total': 5}},
 'wheelchair': {'Total': 695, 'description': {'Total': 16}},
 'width': {'Total': 2},
 'wifi': {'Total': 17},
 'wikipedia': {'Total': 568, 'en': {'Total': 1}},
 'wood': {'Total': 7},
 'wpt_symbol': {'Total': 1},
 'xmas': {'day_date': {'Total': 1},
          'feature': {'Total': 1},
          'name': {'Total': 1},
          'openning_hours': {'Total': 1},
          'url': {'Total': 1}},
 'year': {'Total': 2},
 'yelp': {'Total': 1},
 'zone': {'maxspeed': {'Total': 1}}}
```

Source 2_key_parsing.py

As commented before the last table, I decided to contruct the JSON file using the same methodology, but I found a problem when a key appears first alone and in next steps appears combined with an other level. For example:

- The key *crossing* appears first alone:
  - `<tag k="crossing" v="traffic_signals"/>`
- And after appears combined:
  - `<tag k="crossing:light" v="yes"/>`

Since I first assign the value v to k, when I tried to construct the next level to v and I found an error because v is not a dictionary. So, I decided to create new single keys to this problematic keys, by concatenating the different levels using underscores. For example, regarding the same example used before with the *crossing* key, the resulting JSON will contain "crossing" and "crossing_light". This solution was applied with several keys:

```
['name', 'old_name', 'source', 'population', 'atm', 'internet_access', 'ref', 'is_in',
'capacity', 'wheelchair', 'wikipedia', 'fuel', 'phone', 'website', 'historic',
'social_facility', 'shop', 'amenity', 'railway', 'description', 'subject', 'fee', 'building',
'opening_hours', 'species', 'crossing', 'heritage', 'ele', 'transformer', 'drinking_water',
'inscription', 'surveillance', 'oneway', 'area', 'lanes', 'bridge', 'left', 'both', 'right',
'cycleway', 'scale', 'construction', 'animal_shelter', 'access', 'height']
```

JSON Structure

The final JSON file was constructed following the following structure:

```
{
  "name": "str",
  "type": "node/way",
  "id": "str",
  "visible": "true/false",
  "created": {
      "uid": "str",
      "changeset": "str",
      "version": "str",
      "user": "str",
      "timestamp": "str"
  },
  "address": {
      "city": "str",
      ...
  },
  "pos": [
      float,
      float
  ],
  "singleLevelKey": "value",
  "singleLevelKey_problem_with_levels": "value",
  "multiLevelKey": {
      "secondLevel": {
          "thirdLevel": {
              "fourthLevel": "value"
          },
      },
  },
}
```

Once applied all the considerations commented before, the data was stored in a collection named barcelona inside a Mongo DB named map using the program file *5_Preparing_and_inserting_Database.py*

Problems encountered in the map

Lots of keys with low frequency

Most of the used keys have low frequency (total count lower than 10-5), this could be caused basically for two reasons. Or the users entered customized / similar keys or there are too few entries in the map. In general I found the map highly incomplete.

Similar keys

I found several keys that can be grouped together. For example:

```
['plastic': {'Total': 155}, 'plastic_bottles': {'Total': 35}, 'plastic_packaging': {'Total':
33}]
```

That's a problem, because you have to know all the keys levels. Creaning that, is a hard manual work, because I need to review all the keys one by one…

Mix of Languages (Spanish & Catalan)

Before I started to audit the data, I suposed that this could be a problem because Barcelona is part of the region of Catalonia, where coexits two official languages, and effectively that was true. For example, I found streets with Street names in spanish. Next table shows an example regarding *Calle* (spanish) vs. *Carrer* in Catalan:

```
db.barcelona.find({"name": /\bCarrer\b/}).count()
```

```
14323
```

```
db.barcelona.find({"name": /\bCalle\b/}).count()
```

```
14
```

Abbreviations in names

Some names have abbreviations like "C/ or C." for street, "Av." for avenue, "Pl." for square, "Pg." for passage. Next table shows an example for "Av. or Av":

```
db.barcelona.aggregate( {$match: {"name": {$exists: true, $regex: /\bAv\b/ }}}, {
$group:{_id:"$name","count":{$sum:1}} }, { $sort:{count:-1}}, {$limit: 5} )
```

```
{ "_id" : "Av. 3 nº 24", "count" : 4 }
{ "_id" : "Av. Piera - Montseny", "count" : 4 }
{ "_id" : "Av. Piera - Masia Carrancà", "count" : 4 }
{ "_id" : "Av. Maria Torras", "count" : 4 }
{ "_id" : "Av. Mossen Cinto Verdaguer - Rambla", "count" : 4 }
```

```
db.barcelona.find({"name": /\bAv\b/}).count()
```

```
277
```

Missing Street type or errors

I found multiple street names poorly written and with some typographical errors that were revised using the program. The list of expexted names were updated, as well as the mapping dict. In addition, I created a list of streets where the street type was missing, so I can update it programatically. A reduced sample output is shown below.

```
C/ Burgos => Carrer Burgos
Avda. Badalona => Avinguda Badalona
Pg Gràcia => Passeig Gràcia
Av. Diagonal => Avinguda Diagonal
...
```

Source 3_improving_street_names.py

Some adresses are not in Barcelona

While checking the data, I discovered some cities included in the Barcelona Metropolitan Area but not in the city. By quering the db, trying to find the top 10 cities:

```
db.barcelona.aggregate([{"$match":{"address.city":{"$exists":1}}},{"$group":{"_id":"$address.cit
y", "count":{"$sum":1}}}, {"$sort":{"count": -1}}, {"$limit":10}])
```

```
{ "_id" : "Barcelona", "count" : 7059 }
{ "_id" : "Santa Coloma de Cervelló", "count" : 2976 }
{ "_id" : "Badalona", "count" : 450 }
{ "_id" : "Cornellà de Llobregat", "count" : 303 }
{ "_id" : "El Prat de Llobregat", "count" : 236 }
{ "_id" : "L'Hospitalet de Llobregat", "count" : 109 }
{ "_id" : "Sant Cugat del Vallès", "count" : 79 }
{ "_id" : "Sant Boi de Llobregat", "count" : 79 }
{ "_id" : "Santa Coloma de Gramenet", "count" : 21 }
{ "_id" : "Cerdanyola del Vallès", "count" : 20 }
```

Inconsistent Post Codes

The standard post code in Barcelona is a 5 digit number starting by 08, and I found some of them not matching this format:

```
db.barcelona.aggregate( {$match: {"address.postcode": {$exists: true, $not: /(^\d{5}$)/ }}}, {
$group:{_id:"$address.postcode","count":{$sum:1}} }, { $sort:{count:-1} } )
```

```
{ "_id" : "08", "count" : 3 }
{ "_id" : "Barcelona", "count" : 1 }
{ "_id" : "8007", "count" : 1 }
{ "_id" : "089809", "count" : 1 }
{ "_id" : "08193 ", "count" : 1 }
{ "_id" : "08037 BCN", "count" : 1 }
{ "_id" : "72", "count" : 1 }
```

Programatically I fixed all the incorrect post codes obtaining:

```
72 => 08072
08037 BCN => 08037
089809 => 08980
08193  => 08193
08 => 08000
8007 => 08007
Barcelona => 08000
```

Source 4_Improving_post_codes.py

Data Overview

Some Satistics about the data:

```
db.barcelona.stats()
```

```
{
        "ns" : "map.barcelona",
        "count" : 949764,
        "size" : 289624512,
        "avgObjSize" : 304,
        "numExtents" : 14,
        "storageSize" : 335900672,
        "lastExtentSize" : 92585984,
        "paddingFactor" : 1,
        "paddingFactorNote" : "paddingFactor is unused and unmaintained in 3.0. It remains hard
coded to 1.0 for compatibility only.",
        "userFlags" : 1,
        "capped" : false,
        "nindexes" : 1,
        "totalIndexSize" : 30823520,
        "indexSizes" : {
                "_id_" : 30823520
        },
        "ok" : 1
}
```

Number of unique users:

```
db.barcelona.distinct('created.user').length
```

```
1639
```

Number of nodes:

```
db.barcelona.find({"type":"node"}).count()
```

```
842095
```

Number of ways:

```
db.barcelona.find({"type":"way"}).count()
```

```
107372
```

Top 10 contributing Users:

```
db.barcelona.aggregate([{"$match":{"created.user":{"$exists":1}}},{"$group":{"_id":"$created.use
r", "count":{"$sum":1}}}, {"$sort":{"count": -1}}, {"$limit":10}])
```

```
{ "_id" : "josepmunoz", "count" : 96083 }
{ "_id" : "DanielBautista", "count" : 80595 }
{ "_id" : "EliziR", "count" : 60390 }
{ "_id" : "woodpeck_repair", "count" : 33664 }
{ "_id" : "moogido", "count" : 31792 }
{ "_id" : "Carlos_Sánchez", "count" : 26465 }
{ "_id" : "Toni Guasch", "count" : 24677 }
{ "_id" : "Luis Peña", "count" : 22915 }
{ "_id" : "jolivares", "count" : 20833 }
{ "_id" : "sim6", "count" : 20180 }
```

Top 10 amenities:

```
db.barcelona.aggregate([{"$match":{"amenity":{"$exists":1}}},{"$group":{"_id":"$amenity",
"count":{"$sum":1}}}, {"$sort":{"count": -1}}, {"$limit":10}])
```

```
{ "_id" : "parking", "count" : 1346 }
{ "_id" : "restaurant", "count" : 981 }
{ "_id" : "drinking_water", "count" : 722 }
{ "_id" : "bench", "count" : 676 }
{ "_id" : "school", "count" : 587 }
{ "_id" : "parking_entrance", "count" : 521 }
{ "_id" : "recycling", "count" : 496 }
{ "_id" : "bank", "count" : 470 }
{ "_id" : "pharmacy", "count" : 417 }
{ "_id" : "bar", "count" : 357 }
```

Entries per Year

```
db.barcelona.aggregate({$project:{year: { $substr: [ "$created.timestamp", 0, 4 ]
}}},{$group:{_id: {year:"$year"}, entries:{$sum:1}}},{$sort:{entries:-1}})
```

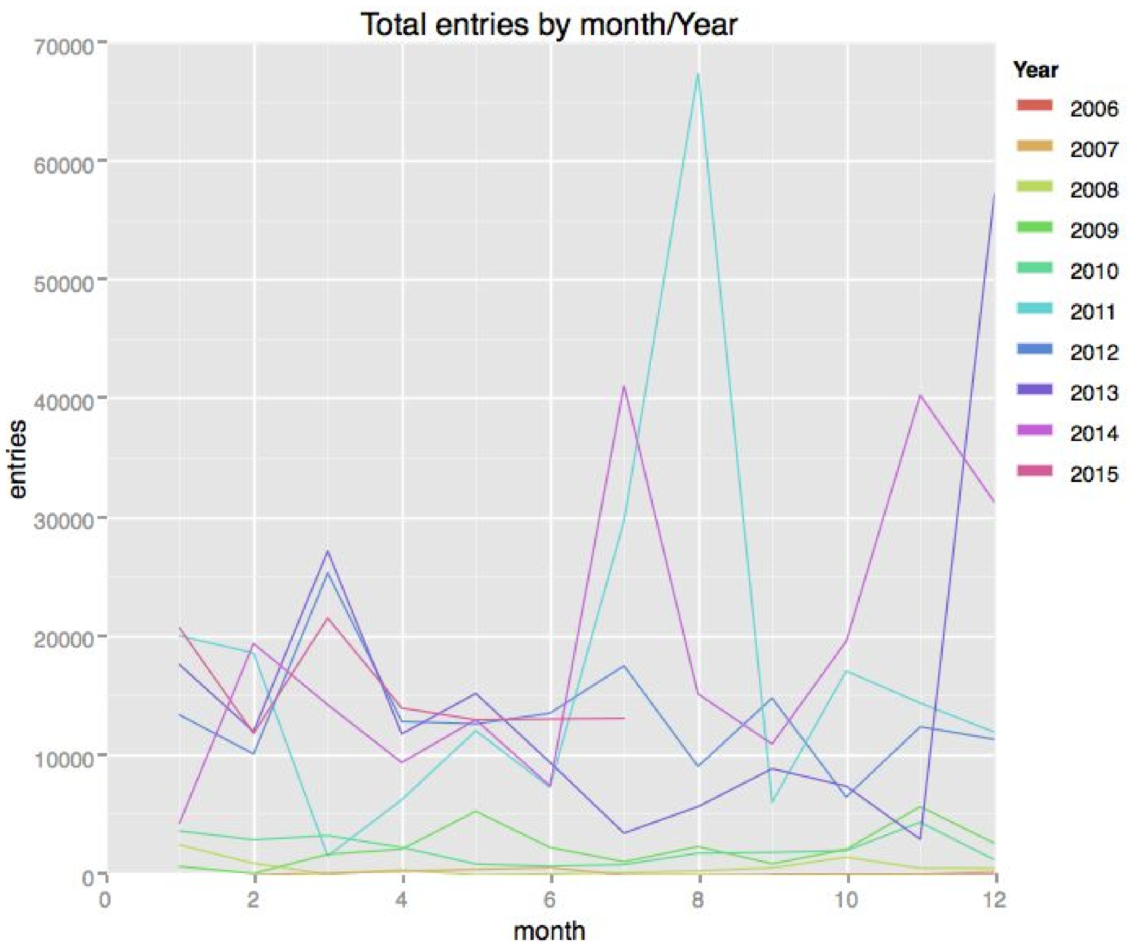
```
{ "_id" : { "year" : "2014" }, "entries" : 226943 }
{ "_id" : { "year" : "2011" }, "entries" : 213144 }
{ "_id" : { "year" : "2013" }, "entries" : 179486 }
{ "_id" : { "year" : "2012" }, "entries" : 160306 }
{ "_id" : { "year" : "2015" }, "entries" : 107668 }
{ "_id" : { "year" : "2009" }, "entries" : 27290 }
{ "_id" : { "year" : "2010" }, "entries" : 26174 }
{ "_id" : { "year" : "2008" }, "entries" : 7706 }
{ "_id" : { "year" : "2007" }, "entries" : 1032 }
{ "_id" : { "year" : "2006" }, "entries" : 15 }
```

Additional Ideas

Improving analysis

Using the pipeline operator "$project" and string operator “$substr”, lets me extend the analysis beyond, because I can study detailed data over time. In addition, If I apply plotting concepts learnt in Intro to Data Science Course, I obtain a richer information:

```
pipeline = [{ "$project": {"year": { "$substr": [ "$created.timestamp", 0, 4 ] },
                           "month": { "$substr": [ "$created.timestamp", 5, 2 ] }}
            },
            { "$group":   {"_id": { "year" : "$year" , "month" : "$month" },
                           "entries": { "$sum" : 1 }}
            },
            { "$sort":    { "_id" : 1 }}]
```



Source 6_improving_analysis.py

Quickly, I can observe that there are months that seems to have more entries than other like March, July, August, November and December (maybe more entries in summer and winter…), or that entries are higher year over year, etc...

Improving data quality

In general the Barcelona data obtained from OpenStreetMaps is rich regarding the number of nodes/ways, but poor in extra information like amenities. For example, there are only aprox. 1000 restaurants labelled, while in 2007 Barcelona had near 6000 (source).

The quality of the information stored, is also poor, with a lot of typographical errors that must be audited and fixed prior to analyse it, and a lot of custom-keys with low frequency.

The level of contribution is quite high, and is increasing year by year. While writing this text, the number of entries in 2015 is close to 50% of the maximum obtained in 2014, so I suppose that the level will be the equal or better than that.

This basic analysis make me think in some solutions/ideas:

1. To limit the tags to a closed list: this will solve the problem of low frequency keys or custom-made ones, and forces the editor to choose and not create. I'm thinking not in to lock completely the freedom of the user to introduce new data, but to control the data introduction by using validation methods by using the same users to validate the information submitted by other users (i.e.: A user that creates a node will be the node owner and could review the updates/edits done by other users).
2. Sharing data between administrations and statistical Institutes: most of the data stored can be obtained from other sources and avoid the user manual introduction. This idea is easy to say and a bit difficult to putting into practice for several reasons… First, most of established sources are apprehensive to share their data in an open way. I don't know if this is a common practice in the rest of Europe/World, but in Spain is really difficult to deal with these organizations. Second, these sources have their data outdated (i.e.: I was unable to find the actual total number of restaurants in Barcelona City. The nearest data are from 2007 so they need an update too...)
3. Realtime updates & automated introduction using mobile devices: nowadays the use of applications that record our position is increasing. I'm thinking in some kind of application, where users checks a location and introduces/updates the OSM data in realtime using the existing OSM API. An additional control and security firewalls will be needed as well as a robust app development, but for sure, the user participation will increase. Using this idea, could introduce a gamification concept, that users will appreciate.

The ideas provided use the collective intelligence to maintain updated the OSM data, because the small contribution of one user supported by all the OSM users community, has an enormous potential. On the other hand increases the possibility of having/introducing the spam into the community, so an extra level of control must be implemented, but again, the collective intelligence comes to play, because if every user/contributor, controls an small piece of the map, all the map could be covered.

Another potential problem (trade-offs), is the increase of the computational power and time bound to putting into practice some of these solutions.