Map Area

I decided to look into Beijing, with a focus on the types and distribution of buildings. I may end up going there to teach English at some point in the next few years, and I wanted to get a better understanding of the city's layout.

Issues cleaning data

Cultural issues

In China, hyphenated words are common. For this reason, regex queries meant to test the keys on tags were modified to include dashes as equivalent to lowercase.

```
lower = re.compile(r'^([a-z]|_)*$')
lower_with_dash = re.compile(r'^([a-z]|_|-)*$')
```

The dashes were considered potentially problematic, as they could have been mistaken for minus signs. Therefore, they were replaced with underscores when piped into csv.

Additionally, Chinese characters in unicode would often show up, for obvious reasons.

Problematic characters

In beijing_china.osm, the nature of the keys in tags was examined to determine whether it would cause any problems later.

```
{'FIXME': 556,
 'lower': 300597,
 'lower_colon': 17647,
 'lower_colon_with_dash': 46,
 'lower_double_colon': 56,
 'lower_with_dash': 22,
 'other': 169,
 'problemchars': 5,
 'unicode': 73}
```

FIXME does not correspond to a physical feature. It is the key for a type of tag which indicates there may be something wrong with the thing it is tagging.

The contents of Other are primarily tags with uppercase characters and/or numbers in them.

The five entries with problematic keys were found and examined individually. They are listed below.

```
{'k': 'No.', 'v': '985,852,974'}
{'k': 'emergency shelter', 'v': 'park'}
{'k': 'emergency shelter', 'v': 'yes'}
{'k': 'emergency shelter', 'v': 'yes'}
{'k': 'emergency shelter', 'v': 'yes'}
```

In every case, the meaning was obvious, and the key could be converted (“No.”->”Number”, “emergency shelter” → “emergency_shelter”) without any problems.

Categorising building types

Values found associated with the “Building” key (found using collect_building_tags.py) were as follows.

{'3': 1,
'33': 1,
'Dormitory': 1,
'apartments': 470,
'bridge': 7,
'cathedral': 1,
'church': 1,
'collapsed': 1,
'college': 20,
'commercial': 387,
'construction': 2,
'dam': 1,
'dormitory': 47,
'entrance': 3,
'garage': 4,
'greenhouse': 5,
'hospital': 24,
'hotel': 13,
'house': 1395,
'industrial': 72,
'kindergarten': 3,
'office': 63,
'pavilion': 1,
'public': 4,
'railway station': 1,
'residential': 1886,
'retail': 16,
'roof': 36,
'school': 114,
'shed': 1,
'stable': 1,
'stadium': 2,
'temple': 2,
'toll_booth': 6,
'tower': 8,
'train_station': 150,
'university': 46,
'warehouse': 5,
'yes': 27596,
'yesm': 1,
u'\u516c\u4ea4\u9996\u672b\u7ad9': 2,
u'\u6559\u5ba4': 1,
u'\u96d5\u5851': 1}

The most common 'type' of building turned out to simply be 'yes': most buildings in the map were not specified as anything other than existent. To avoid confusion, all “yes”es (and the one “yesm”) were replaced with “unspecified” when the data was being piped in.

The multiple entries for Dormitory and dormitory made it clear why everything needed to be lowercase. All keys with uppercase letters had them programmatically replaced with their lowercase equivalents when the converted to csv format.

Numbers and unicode tags were rare enough that they could just be excluded from the data set without a major impact on data quality.

Categorising amenities

A similar test was done on the types of amenities.

{'Beer Garden': 1,
'Massage': 1,
'arts_centre': 7,
'atm': 96,
'bank': 400,
'bar': 108,
'bbq': 1,
'bench': 45,
'bicycle_parking': 55,
'bicycle_rental': 39,
'bicycle_repair_station': 7,
'biergarten': 1,
'billiards': 1,
'bureau_de_change': 3,
'bus_station': 73,
'cafe': 194,
'car_wash': 12,
'charging_station': 10,
'childcare': 3,
'cinema': 28,
'clinic': 7,
'clock': 1,
'college': 37,
'commercial building': 1,
'community_centre': 5,
'conference_centre': 1,
'convenience shop': 1,
'courthouse': 3,
'dentist': 8,
'doctors': 3,
'dorm': 1,
'drinking_water': 3,
'driving_school': 2,
'embassy': 101,
'exhibition_centre': 1,
'fast_food': 275,
'fire_station': 6,
'fountain': 24,
'fuel': 280,
'grave_yard': 6,

'hanging rings': 1,
'hospital': 165,
'ice_cream': 3,
'investment_bank': 1,
'kindergarten': 37,
'library': 26,
'marketplace': 43,
'massage': 1,
'nightclub': 30,
'parking': 637,
'parking_entrance': 9,
'parking_space': 2,
'pharmacy': 64,
'pingpong': 1,
'place_of_worship': 81,
'police': 53,
'post_box': 57,
'post_office': 43,
'prison': 3,
'pub': 21,
'public_bookcase': 4,
'public_building': 11,
'punching_bag': 1,
'recycling': 4,
'research_institute': 1,
'restaurant': 1102,
'school': 364,
'shelter': 26,
'spa': 2,
'studio': 2,
'swimming_pool': 20,
'taxi': 22,
'telephone': 152,
'theatre': 20,
'toilets': 348,
'townhall': 17,
'training': 1,
'university': 101,
'vending_machine': 8,
'veterinary': 4,
'waste_basket': 50,
'waste_disposal': 1}

This data was fairly clean in and of itself. The capital letters would be taken care of by the lowercasing inspired by the building tag values. The lack of a space in Beer Garden made it clear that values as well as keys should have spaces auto-replaced with underscores.

The amenities data had more issues in light of the buildings data: “hospital”, “university” and “college” showed up as both amenities and buildings, with different counts. To account for this, all hospitals, universities and colleges with only a building tag were given a corresponding amenity tag when being piped in; likewise, hospitals and colleges with only an amenity tag were given a building tag.

Possibility of duplication

Visually skimming the .osm data revealed that nodes, ways and relations all had tags with "building" as the key. This raised the possibility of double-counting: if one person tagged a node as a building, and another person more thoroughly investigated the same building and made a relation representing it, it's not obvious that this duplication would be found or corrected. Confirming that no such duplication took place became a major part of the analysis.

Data overview

Original .osm dataset

The .osm file used is 161.1MB uncompressed.

Running count_tags.py on beijing_china.osm finds the number of XML items with each kind of tag. These are as follows:

{'bounds': 1,
'member': 60383,
'nd': 864391,
'node': 724008,
'osm': 1,
'relation': 5560,
'tag': 319171,
'way': 106756}

Note that nd and node aren't redundant: nd is an object which relates a node to a way that depends on it.

SQL dataset

All nodes, ways, relations and associated tags were piped into a SQL database, along with all objects which identified which nodes belonged to each way and which ways and nodes belonged to each relation.

The total numbers of nodes, ways and relations were checked, to ensure all data was successfully piped.

sqlite> SELECT count(*) FROM Node;
724008
sqlite> SELECT count(*) FROM Way;
106756
sqlite> SELECT count(*) FROM Relation;
5560

The five users who made most nodes, most ways and most relations were found.

sqlite> SELECT user, count(uid) FROM Node GROUP BY uid ORDER BY count(uid) DESC LIMIT 5;
"Chen Jia",168190
R438,128122

ij_,49986
hanchao,40635
katpatuka,22865

sqlite> SELECT user, count(uid) FROM Way GROUP BY uid ORDER BY count(uid) DESC LIMIT 5;
"Chen Jia",25743
R438,22907
hanchao,7631
nuklearerWintersturm,2773
RationalTangle,2124

sqlite> SELECT user, count(uid) FROM Relation GROUP BY uid ORDER BY count(uid) DESC LIMIT 5;
R438,4488
gerg1,206
"Chen Jia",176
4rch,76
hanchao,61

This shows that the userbase which made nodes and the userbase which mostly makes relations are substantially different, confirming that the possibility of duplicated buildings is legitimate.

The total number of contributors was found by looking at the number of unique user ids which were involved in adding nodes.

*sqlite> SELECT count(*) FROM (SELECT * FROM Node GROUP BY uid);*

1361

By comparison, far fewer were involved in adding relations.

*sqlite> SELECT count(*) FROM (SELECT * FROM Relation GROUP BY uid);*
182

Data analysis

Check for double-counted buildings

(Note that this subsection is focused on ensuring data quality by working out which parts of the imported data to ignore, and may qualify as an unusually blunt form of cleaning in addition to analysis.)

The first question considered was how many buildings were counted as nodes, ways and relations.

sqlite> SELECT count(*) FROM (Node INNER JOIN NodeTag ON Node.id=NodeTag.id) WHERE NodeTag.key="building";
349
sqlite> SELECT count(*) FROM (Way INNER JOIN WayTag ON Way.id=WayTag.id) WHERE WayTag.key="building";
32231
sqlite> SELECT count(*) FROM (Relation INNER JOIN RelationTag ON

Relation.id=RelationTag.id) WHERE RelationTag.key="building";
90

The overwhelming majority of buildings are listed as ways.

The possibility of overlap was tested for by finding the maximum and minimum longitude, and the maximum and minimum latitude, of every way marked as a building; and then, testing every node marked as a building to see whether it is within those bounds. (Note that, as buildings are not always perfect north-facing rectangles, this test may give false positives for overlap, but no false negatives.)

*sqlite> SELECT count(*) from (SELECT * FROM (Node INNER JOIN NodeTag ON Node.id=NodeTag.id) WHERE NodeTag.key="building") as BNode, (SELECT BWay.id as bwid, max(lat) as maxlat, min(lat) as minlat, max(lon) as maxlon, min(lon) as minlon FROM (((SELECT * FROM (Way INNER JOIN WayTag ON Way.id=WayTag.id) WHERE WayTag.key="building") as BWay JOIN WayNode ON BWay.id=WayNode.id) JOIN Node ON Node.id=WayNode.node_id) GROUP BY bwid) as limits WHERE BNode.lat>=minlat AND BNode.lat<=maxlat AND BNode.lon>=minlon AND BNode.lon<=maxlon;*

154

The values in the tags for the 76 pairs of nodes and ways were found using the command below.

*sqlite> SELECT BNode.value, limits.bwvalue from (SELECT * FROM (Node INNER JOIN NodeTag ON Node.id=NodeTag.id) WHERE NodeTag.key="building") as BNode, (SELECT BWay.id as bwid, BWay.value as bwvalue, max(lat) as maxlat, min(lat) as minlat, max(lon) as maxlon, min(lon) as minlon FROM (((SELECT * FROM (Way INNER JOIN WayTag ON Way.id=WayTag.id) WHERE WayTag.key="building") as BWay JOIN WayNode ON BWay.id=WayNode.id) JOIN Node ON Node.id=WayNode.node_id) GROUP BY bwid) as limits WHERE BNode.lat>=minlat AND BNode.lat<=maxlat AND BNode.lon>=minlon AND BNode.lon<=maxlon;*

It was determined that, in the cases where a type other than “yes” was specified for both the way-building and the node-building, they were the same type a majority of the time. This is clear evidence of double-counting. Furthermore, a disproportionate number of double-counted buildings were apartments or hospitals, which could have biased the sample.

Characterise buildings

To guarantee an absence of double-counted buildings, only the data from ways was used.

sqlite> SELECT Value, count(*) FROM WayTag WHERE Key="building" GROUP BY Value;

apartments,440
bridge,7
church,1
college,38
commercial,373
construction,2
dam,1
dormitory,47
garage,4

greenhouse,4
hospital,70
hotel,9
house,1376
industrial,72
kindergarten,3
office,45
public,3
railway_station,1
residential,1852
retail,16
roof,33
school,99
shed,1
stable,1
stadium,1
temple,2
toll_booth,6
tower,8
train_station,149
university,127
unspecified,27435
warehouse,5

This can be summarised as 3724 homes (residential, dormitory, apartments, hotel, house), and 27435 unspecified buildings (unspecified), and 1072 confirmed nonhomes (everything else).

Characterising amenities

The first question considered was how many buildings were counted as nodes, ways and relations.

sqlite> SELECT count(*) FROM (Node INNER JOIN NodeTag ON Node.id=NodeTag.id) WHERE NodeTag.key="amenity";
3828
sqlite> SELECT count(*) FROM (Way INNER JOIN WayTag ON Way.id=WayTag.id) WHERE WayTag.key="amenity";
1645
sqlite> SELECT count(*) FROM (Relation INNER JOIN RelationTag ON Relation.id=RelationTag.id) WHERE RelationTag.key="amenity";
8

Following the example of the building queries, only the nodes were considered, to avoid double-counting.

sqlite> SELECT Value, count(*) FROM NodeTag WHERE Key="amenity" GROUP BY Value;
arts_centre,2
atm,96
bank,393
bar,107
bbq,1
beer_garden,1
bench,42

bicycle_parking,19
bicycle_rental,39
bicycle_repair_station,7
biergarten,1
billiards,1
bureau_de_change,3
bus_station,37
cafe,190
car_wash,7
charging_station,9
childcare,1
cinema,27
clinic,4
clock,1
college,15
community_centre,3
conference_centre,1
convenience_shop,1
courthouse,3
dentist,8
doctors,3
dorm,1
drinking_water,3
driving_school,1
embassy,19
fast_food,272
fire_station,5
fountain,15
fuel,102
grave_yard,3
hanging_rings,1
hospital,99
ice_cream,3
investment_bank,1
kindergarten,23
library,9
marketplace,22
massage,2
nightclub,27
parking,114
parking_entrance,9
pharmacy,63
pingpong,1
place_of_worship,38
police,37
post_box,57
post_office,37
pub,21
public_bookcase,3
public_building,1
punching_bag,1
recycling,4

restaurant,1049
school,145
shelter,25
spa,2
studio,1
swimming_pool,9
taxi,23
telephone,152
theatre,10
toilets,306
townhall,13
training,1
university,13
vending_machine,8
veterinary,4
waste_basket,50
waste_disposal,1

The most common amenity represented as a node was “restaurant”, by a fairly wide margin.

East/West distinction

It is possible to find the approximate average location of a building by averaging its maximum and minimum latitude, and doing the same with latitude. By finding the average of the average longitudes, we can find the longitude of a line which rougly divides the east and west halves of the city.

*sqlite> SELECT avg(approxlon) FROM (SELECT BWay.id, (max(lat)+min(lat))/2 as approxlat, (max(lon)+min(lon))/2 as approxlon FROM ((SELECT * FROM (Way INNER JOIN WayTag ON Way.id=WayTag.id) WHERE WayTag.key="building") as BWay JOIN WayNode ON BWay.id=WayNode.id) JOIN Node ON Node.id=WayNode.node_id);*

116.4044316

The character of the buildings on the east half of the city can be found using the command below.

*SELECT Value, count(*) FROM ((SELECT *, (max(lat)+min(lat))/2 as approxlat, (max(lon) +min(lon))/2 as approxlon FROM ((Way JOIN WayNode ON Way.id=WayNode.id) JOIN Node ON Node.id=WayNode.node_id) GROUP BY Way.id) as LWay INNER JOIN WayTag ON LWay.id=WayTag.id) WHERE WayTag.key="building" AND LWay.approxlon>116.4044316 GROUP BY Value;*

apartments,121
bridge,7
church,1
college,7
commercial,232
construction,1
dam,1
dormitory,2
garage,1
greenhouse,3

hospital,36
hotel,5
house,103
industrial,55
kindergarten,2
office,18
public,1
residential,1565
retail,13
roof,19
school,29
shed,1
stable,1
train_station,68
university,20
unspecified,13249
warehouse,4

There is not much difference from the overall, though interestingly fewer than a tenth of the buildings marked as houses are in the east side of the city. It is hard to say whether this is a fact about the city or about the map, due to the large number of unspecified buildings, and the larger number of buildings which have not yet been added to the map.

Possibilities for Improvement

Buildings

I believe the data would be more easily navigated if buildings were only ever represented as ways, and never nodes or relations.

Some buildings (those with internal courtyards etc) would still require relations to define their internal geometry, but consistently classing the building as a way and the building detail as a relation associated with that way would give data analysts an easier job.

This would increase the complexity of the (already fairly complex) representation of buildings using relations, and provide no benefit for people using OSM as a Google Maps substitute. However, it would be a boon to data analysts, so the adjustment may be worth it depending on how often and by how many people/organisations the data is analysed.

Amenities

I believe the data would be more easily navigated if amenities were only ever represented as nodes. Amenities added as ways would be automatically converted to a node at the centre of the way, and an associated way with a building tag. This would trade off user freedom and simplicity against ease of analysis, like the change in the previous subsection.

If adding an amenity tag automatically added a building tag with the same value unless the contributor actively marked it as not being a building, more buildings in general (and hospitals in particular) would be marked. However, this would increase the rate of false positives: there are a few amenities, like public phones and punching bags, which are not buildings, and my proposed change would probably see a few of them misclassified as buildings.

General: Unicode

Having unicode characters in this data makes it harder to navigate, both because a sufficiently rigorous analysis would require them to be translated, and because they could be problematic under certain schemas.

Requiring Chinese contributors to never contribute in Chinese would, however, be unreasonable and deter users. Having a separate type of tag for Chinese-language and English-language contributions seems like the best compromise.