OSM History Analysen

Intrinsische Datenqualitäts-Analyse von Geodaten

Einer unserer Forschungs-Schwerpunkte dreht sich um intrinsische Datenqualitätsanalysen von Geodaten. Intrinsisch bedeutet, dass die Datenqualität durch nicht-vergleichende Methoden bestimmt wird. Im Gegensatz dazu stehen extrinsischen Methoden, wo Referenz-Datensätze verwendet werden.

- **Vollständigkeit**
- **Konsistenz (Integrität)**
- **Positions Exaktheit**
- **Temporale Exaktheit**
- **Thematische Exaktheit**

Beim Thema Datenqualität interessiert man sich für verschiedene Aspekte; eine ISO-Norm spezifiziert z.B.: Vollständigkeit, Konsistenz (das Fehlen von Inkonsistenzen wie z.B. widersprüchlichen oder formal fehlerhafte Daten), sowie Genauigkeit bezüglich Position, Zeit und Thema.

Der Punkt der Daten-Vollständigkeit sei mal hervorgehoben, weil dieser im Vergleich schon am besten erforscht ist was intrinsische Analysemethoden angeht: Hierbei ist die These, dass wenn man eine Sättigung der

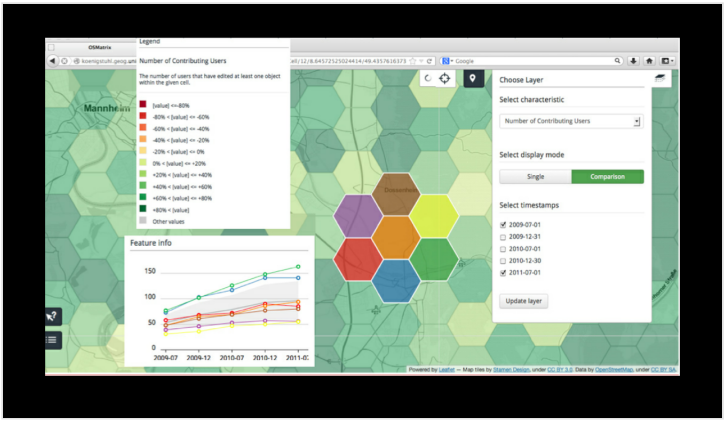
Menge einer bestimmten Objekt-Klasse beobachtet und es gleichzeitig weiter eine hohe Mapping-Aktivität bei ähnlichen Objekt-Typen gibt, dann kann man davon ausgehen, dass die erste Objekt-Kategorie bereits vollständig erfasst ist. Das kanonische Beispiel ist hierbei immer die Evolution von Straßennetzen, wo man in der Regel beobachten kann, dass übergeordnete Straßen (motorway bis primary) vor Nebenstraßen vollständig gemappt sind. Diese Methode kann aber einfach auf weitere Objekttypen übertragen werden.

Daten-Analyse-Platform

Dafür benötigen wir eine Daten-Analyse-Platform die ein effizientes Arbeiten mit OSM-History-Daten ermöglicht.

Methode 1: Alte OSM Planet Dumps

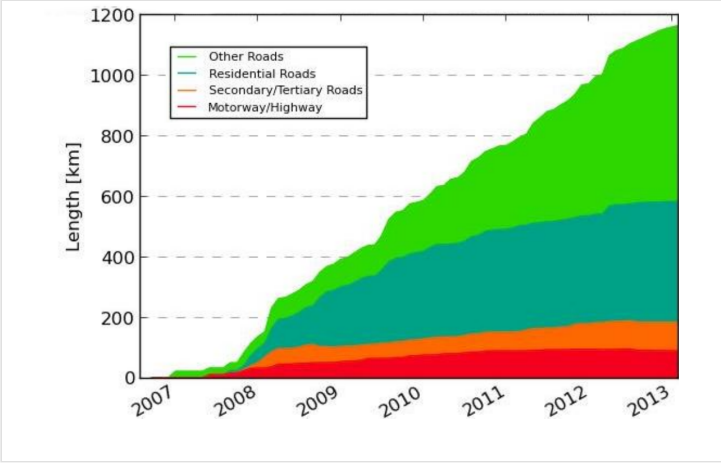
Der einfachste Weg zu solchen Auswertungen zu gelangen, ist einfach auf das OSM-Planet Dump Archiv zurückzugreifen.



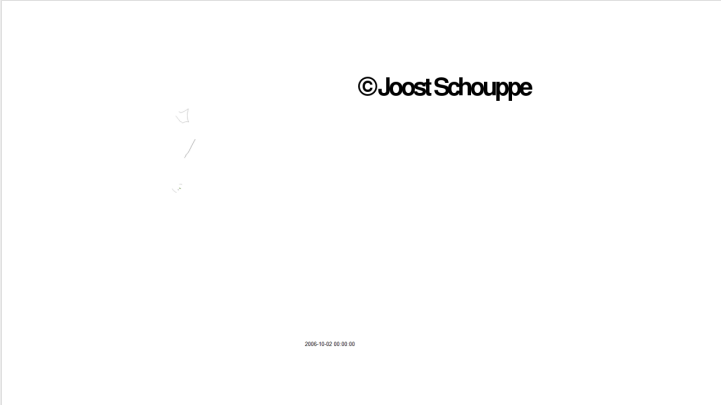
Schon deutlich länger gibt es das OSMatrix-Tool, welches bei uns am Institut entwickelt wurde. Die Technik dahinter ist relativ ähnlich.
http://osmatrix.uni-hd.de

Dabei haben wir allerdings festgestellt, dass dieser Ansatz recht unflexibel ist und die Datenverarbeitung unnötig lange dauert. Zum Beispiel ist es nur mit sehr viel Aufwand möglich nachträglich weitere Auswertungen hinzuzufügen.

Ein effizienterer Weg ist es direkt auf OSM History Dumps zu bauen. Zum Beispiel unter Verwendung der osm-history-renderer toolchain.
https://github.com/MaZderMind/osm-history-renderer



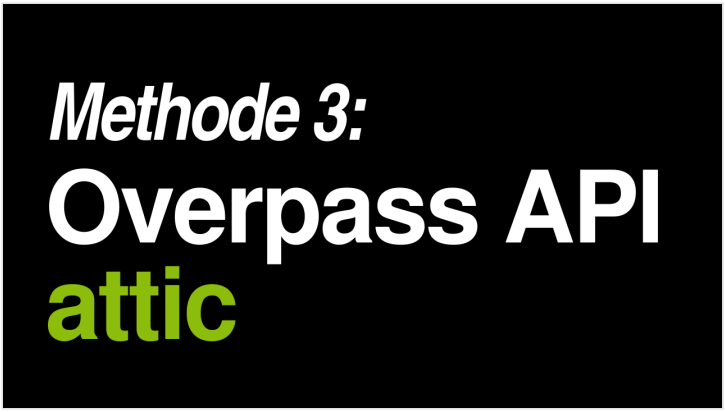
Was intern zum Beispeil von iOSMAnalyzer (ebenfalls entwickelt in der GIScience Gruppe) verwendet wird und neben den bereits erwähnten Vollständigkeit-Analysen auch verschiedene weitere Auswertungen anbietet (z.B. Positionsgenauigkeit von Kreuzungspunkten).
https://github.com/zehpunktbarron/iOSMAnalyze



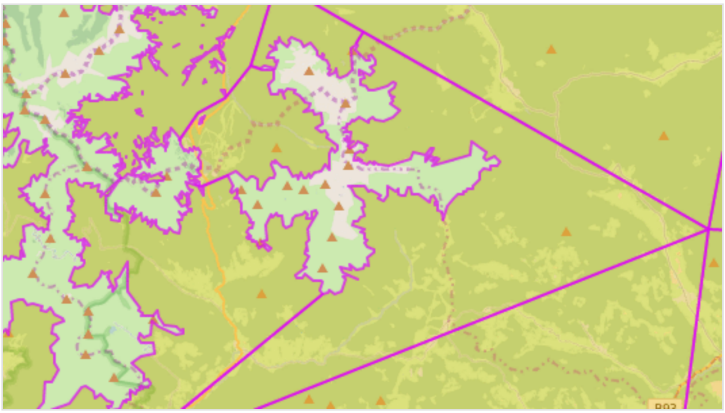
Mit dem osm-history-importer kann man aber auch sehr tolle animierte Karten erstellen, z.B. diese relativ aktuelle von Brüssel erstellt von Joost Schouppe.
http://www.openstreetmap.org/user/joost%20sch

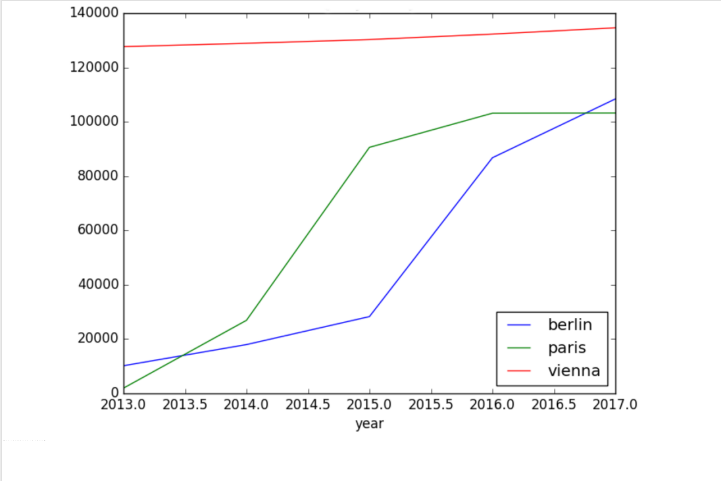
Der Nachteil mit dieser Methode ist leider, dass man diese nicht realistisch auf den gesamten Planet anwenden kann.

Außerdem funktioniert der Ansatz leider auch nicht mit OSM-Relations.



Wenn man schon auf relativ kleine Ausschnitte beschränkt ist, kann man auch gleich auf die History-Funktionen der Overpass-API ausweichen, womit es auch recht einfach möglich ist einfache vorher/nachher Bilder zu zeichnen oder Statistiken zu berechnen.





(Gezeigt sind hier die Anzahl der Bäume (natural=tree) im Verlauf der Zeit seit 2013 in verschiedenen europäischen Städten.) https://github.com/mocnik-science/osm-python-tools

Das Haupt-Problem bei diesem Ansatz ist, dass die Daten der Overpass History Funktionen nur bis zum Zeitpunkt der ODbL-Umstellung zurückgehen.

Außerdem ist diese Methode in der Regel leider auch recht langsam und ebenfalls nur für kleine Gebiete anwendbar.

Für unsere Forschungsaufgaben und -Ziele ist es wichtig, möglichst dynamische Abfragen zu formulieren zu können, z.B.:

- beliebige Untersuchungsgebiete (bis hin zu weltweiten)
- beliebige Objekttypen (nach Tag, Geometrien)
- Zugriff auch auf rohe OSM Daten

Außerdem ist es auch wichtig, dass diese dann möglichst performant ausgeführt werden können.

Deshalb haben wir uns entschlossen, zu versuchen dieses Problem mit etwas „gröberen“ Werkzeugen anzugehen.

Im "big-geo-data"-Umfeld gibt es bereits ein paar mehr oder weniger bekannte Player:

geomesa, geowave

Beide können gut mit großen Mengen von Geodaten umgehen, jedoch sind sie für (historische) OSM-Daten nur bedingt geeignet. Ein Grund dafür ist, dass diese jeweils mit bereits aufbereiteten Objekt-Geometrien (Simple Features) arbeiten und man dadurch keinen direkten Zugriff mehr auf die rohen OSM-Daten hat.

osm2orc, osm-qa-tiles + tile-reduce

Weiters seine diese beiden näher an OSM-Daten orientierten Projekte erwähnt:

osm2orc bietet OSM-History-Rohdaten in einer big-data Datenbank an.

osm-qa-tiles ist ein Ansatz der es über Vektor-Tiles ermöglicht parallelisierte Verarbeitung von OSM-Daten zu ermöglichen.

HeiGIT DB

OSM Daten

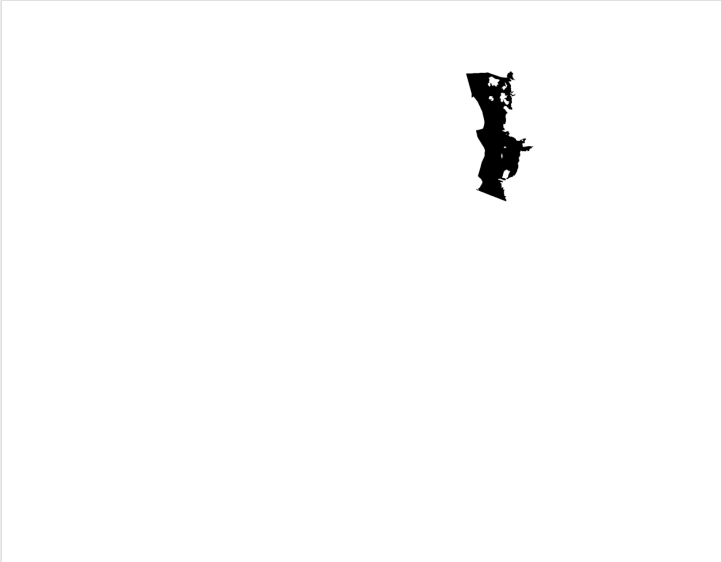
OSM ist in sofern relativ einzigartig, als dass Daten sehr viel häufiger aber auch unregelmäßiger modifiziert werden als z.B. kommerzielle Datensätze.

Deshalb haben manche OSM Objekte hunderte Versionen. Durch die getrennte Speicherung von Ways und Nodes können Ways z.B. noch mal mehr Zwischenversionen besitzen, da eine Koordinaten-Änderung eines Nodes sich auch auf die Geometrie des Ways auswirkt, aber dort nicht

notwendigerweise eine neue Versionsnummer erzeugt wird.

OSM's Datenmodell besitzt also eine recht kompakte Darstellung des historischen Verlaufs (das außerdem mit relativ einfachen Mitteln leicht weiter optimierbar ist).

Hier zwei Beispiele (gifs) von Objekten, mit denen man in OSM zurecht kommen muss.



Historische Geometrien

Jede einzelne (Zwischen)-version der Geometrien jedes OSM Objekts würde den gesamten Datensatz sehr stark aufblähen.

Unser Ansatz ist es deshalb weiterhin relativ rohe OSM-Daten zu speichern und Geometrien dynamisch je nach Bedarf zusammenzubauen.

Dadurch vermeindet man außerdem Verlust von gewissen Teilen der OSM-Daten (z.B. Node-Id-Referenzen von Ways oder "höhere" OSM Datenstrukturen wie Abbiegebeschränkungen), die eventuell für komplexere Auswertungen benötigt werden.

Verteiltes System

Bei einem verteilten System ist es von großer Wichtigkeit, so weit wie möglich jegliche Interkommunikation zwischen den einzelnen Maschinen des Clusters zu minimieren.

Das heißt man muss die Daten so partitionieren, dass oft zusammen verwendete Daten (z.B. die Nodes eines Ways) auch immer am selben Knoten zur verfügung stehen.

Daten in Zellen

Unser Ansatz ist es deshalb die OSM Daten in Zellen organisiert zu speichern. Wie diese Zellen im Detail aussehen ist dabei im Prinzip unerheblich, der einfacheren Handhabung halber verwenden wir ein simples quadratischen lon/lat Grid.

Diese Zellen bilden dabei automatisch einen Index über den gesamten Datensatz.

Im Grunde ist das sehr Ähnlich zu Vektor-Tiles, mit dem Unterschied, dass wir OSM Rohdaten als Inhalt speichern.

Ways/ Relations

Für Punkt-Daten (Nodes) ist es trivial diese in Zellen (eines vorgegebenen Zoom-Levels) zu speichern.

Für Linien oder Flächen ist das aber nicht so einfach, da sich diese potentiell über einen extrem großen Bereich erstrecken können. Denn in den OSM-History-Daten kann man nicht ausschließen solche in der Regel an sich fehlerhafte Daten (z.B. interkontinentale Ways) zu finden! Außerdem müssten die Geometrien von Polygonen und Linien pro Zelle geschnitten werden, was pro (Zwischen)version gemacht werden müsste, was wiederum einen höheren Rechen- und Speicherbedarf voraussetzt.

Lösung: "Tierd Storage"

Ähnlich wie ~~geomesa~~ geowave, nur nicht mit den Geometrien.

Auch können wir nicht einfach die Objekte splitten -> nur einmal speichern, dafür komplett

Nach etwas genaueren Hinsehen bemerkt man außerdem, dass die Overpass API einen Index verwendet der ebenfalls dieser Struktur entspricht.

Geogr. Index

input = bbox, polygon, o.Ä.

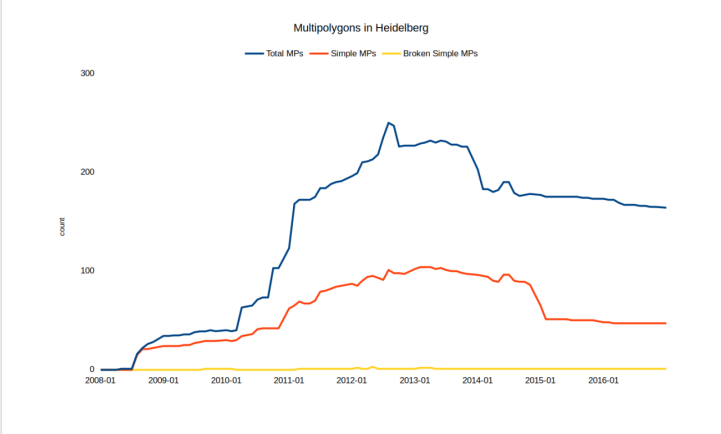
output = Menge an Zellen-Ids, die potentiell durchsucht werden müssen

map-reduce

map = Filtern von objekten (z.B. nach tags, timestamp, o.Ä.) + Zusammenbauen der Objekt-Geometrien + Filtern nach Geometries + Berechnung von statistiken (z.B. Länge von Straßen nach highway-Typ)

reduce = Aggregieren der Einzelergebnisse

Sehr flexibel, allerdings streng genommen auf Einzelobjekt-Analysen beschränkt -> komplexere Analysen (z.B. Objekt-Objekt Korrelationen) benötigen u.U. mehrere map-reduce Schritte



ODbL

Aktuelle full history planet dumps haben nicht alle Daten von vor ~2012.

Das ist potentiell problematisch beim Zusammenbauen von (älteren) Geometrien, da evtl. nicht alle Referenzen aufgelöst werden können.

Die einzige gangbare Lösung ist es für Daten von vor Mitte 2012 auf einen alten cc-by-sa 2.0 History-Dump zurückzugreifen. Man kann dafür zwei parallele DBs für die jeweiligen Datensätze vorhalten.

martin.raifer@uni-heidelberg.de
osm / twitter: tyr_asd
github: tyrasd