

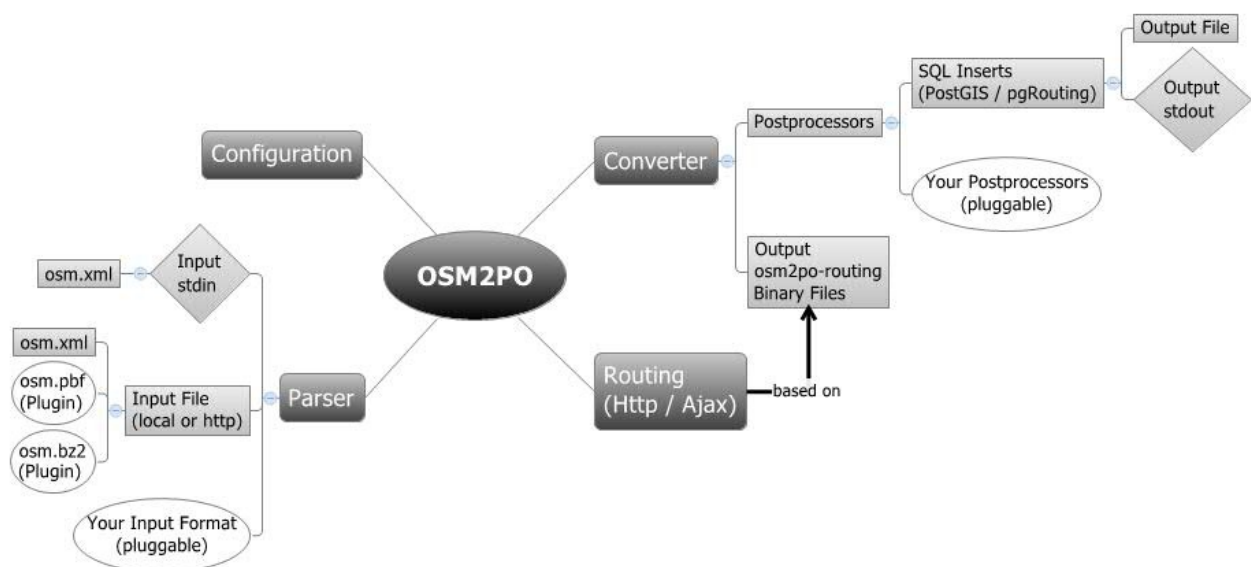
OSM2PO

OpenStreetMap-Data To Topology Converter and Routing Engine
for Java™ 6+

© 2011 Carsten Möller, Pinneberg, Germany
info@osm2po.de

- Schnelleinstieg
- Kurzdokumentation
- Referenz

für Version 4.x / 2012-04-10



This work (document) is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Inhaltsverzeichnis

1) Einleitung.....	3
1.1 Die Geburt.....	3
1.2 Das Team.....	3
1.3 Danksagungen.....	3
1.4 Was osm2po ist.....	3
1.5 Was osm2po nicht ist.....	3
1.6 Version 4.....	3
2) Lizenzen.....	4
3) Installation.....	5
4) Schnelleinstieg.....	6
4.1 Funktionstest und erster Kontakt.....	6
4.2 Was sonst noch passiert ist.....	7
4.3 Daten mit psql in die Datenbank einspielen.....	7
4.4 Daten unter QuantumGIS visualisieren.....	8
4.5 Kürzesten Weg mit pgRouting finden.....	9
5) Probleme bei der Konvertierung.....	11
5.1 Prinzipielle.....	11
5.2 OSM-Daten.....	11
5.3 OSM Relationen.....	12
5.3.1 Abbiegevorschriften in OSM.....	12
5.4 Speicher.....	12
6) Referenz.....	13
6.1 Aufruf.....	13
6.2 Konfiguration.....	13
6.3 Die wichtigsten Parameter.....	13
6.3.1 tileSize und -Xmx.....	13
6.3.2 cmd.....	14
6.4 Schnellübersicht weiterer Parameter.....	16
7) Tipps und Beispiele.....	17
7.1 Analyse und Vorgehensweisen.....	17
7.2 Richtwerte für -Xmx und tileSize.....	17
7.3 Beispiele.....	18
8) osm2po Routing.....	19

1) Einleitung

1.1 Die Geburt

osm2po wurde im Herbst 2009 als kleines Hobby-Tool geboren. Ziel war es, OpenstreetMap-Daten irgendwie für einen einfachen Routingalgorithmus brauchbar zu machen. Es gab derzeit wenige Lösungen zu diesem Thema und wenn, dann musste ich einen riesigen Aufwand betreiben, um diese zu installieren oder überhaupt zum Laufen zu bekommen. Ich entschloss mich daher eine eigene Lösung zu schaffen, die a) performant, b) einfach zu handhaben und c) vom Betriebssystem unabhängig ist.

1.2 Das Team

Das Programmiererteam ist nachwievor eine One-Man-Show. Nämlich meine Wenigkeit. Das komplette Resultat rundum **osm2po** entsteht in meiner Freizeit. Trotz dieser Tatsache habe ich einen recht hohen Anspruch an Software und deren Qualität. **osm2po** ist daher kein Schnellschuss, sondern steht auf stabilen Fundamenten.

1.3 Danksagungen

Einen großen Dank gilt meiner Frau Chrissie und meiner Tochter Mia, die so einige Abende unter meinem Programmier-Wahn zu leiden hatten.

1.4 Was osm2po ist

osm2po ist zweierlei, ein Konvertierungsprogramm, welches das Straßennetzwerk aus den OpenStreetMap-Daten extrahiert, dieses für ein Routing aufbereitet und ein kleiner Routingsservice, der auf diesen Daten arbeitet. Zudem werden im Standardumfang SQL-Dateien generiert, welche direkt in eine PostGIS-Datenbank eingelesen werden können. Das Tabellenlayout ist so gewählt, dass diese Daten ohne Umwege unter QuantumGIS (qgis.org) visualisiert werden können und zudem ein Routing mittels pgRouting (pgrouting.org) für die Algorithmen *ShortestPath* und *ShortestPathAstar* möglich ist.

osm2po ist komplett unter Java™ 6 entwickelt und besitzt keine Abhängigkeiten zu Fremdbibliotheken. Der Funktionsumfang ist jedoch erweiterbar. Hierzu besitzt das Programm diverse Schnittstellen, deren Implementationen per Konfiguration ausgetauscht werden können. Im Download-Paket bereits enthalten sind z.B. zwei Parser, die es ermöglichen, Daten nicht nur aus reinen XML-Quellen zu lesen, sondern auch Bzip2 und Pbf (Google's Protobuf Binary) zu verarbeiten. Diese Erweiterungen werden in **osm2po** als Plugins bezeichnet, sind aber lediglich weitere vom Java-Classloader einzubindende Jar-Dateien. Für diesen Zweck besitzt **osm2po** seinen eigenen Classloader, welcher Fremdbibliotheken automatisch aus dem plugins-Verzeichnis einbindet.

1.5 Was osm2po nicht ist



osm2po kümmert sich ausschließlich um OpenStreetMap-Daten, die für ein Routing relevant sind. Es ist nicht das Ziel dieses Programms den kompletten Datenbestand in Tabellen zu schreiben oder zu interpretieren.

1.6 Version 4

osm2po ist mittlerweile in der Version 4. Die Vorgängerversion 3 wurde komplett überarbeitet und im Kern komplett neu geschrieben. Daher sind die Versionen auch nicht mehr kompatibel zueinander. Dokumentation, die sich auf Version 3 bezieht ist nicht für die Version 4 gültig und andersherum.

2) Lizenzen

Das **osm2po**-Paket (zip-Datei) enthält mehrere Dateien mit unterschiedlichen Lizenzmodellen. Für alle Modelle gilt, dass die Ersteller keine Garantie für mögliche Schäden, die durch diese Werke entstehen, übernehmen. Die Ausführung der Programme geschieht ausschließlich auf eigenes Risiko.

osm2po-core-4.x.jar <i>Hauptprogramm (Kern)</i>	FreeWare	D.h. Der Quellcode ist nicht offen. (ClosedSource). Das Programm kann jedoch frei (auch kommerziell) verwendet werden. Es darf aber nicht verändert, dekompiert oder entpackt werden oder nur in Teilen zum Einsatz kommen. Bei einer kommerziellen oder öffentlichen Nutzung sind die Namen des Programms und des Autors dem Endanwender in einem geeigneten Kontext sichtbar anzuzeigen. Das Programm darf weder weiterverkauft noch vermietet werden.
osm2po-plugins-4.x.jar <i>Erweiterungn (Plugins)</i> osm2po-plugins-4.x-src.zip <i>Quellcode</i>	OpenSource	Diverse OpenSource-Lizenzen. Die Lizenzen sind jeweils im Kopfbereich der Quelldateien einsehbar und nicht im Paket als gesonderte Dateien mitgeliefert.
Dieses Dokument	<i>Creative Commons Attribution-ShareAlike 3.0</i>	
Generierte Daten	<i>Aktuell gültige OSM-Lizenz. Derzeit Creative Commons Attribution-ShareAlike 2.0</i>	 Alle generierten Daten verbleiben unter der aktuellen OpenStreetMap-Lizenz und sind entsprechend kenntlich zu machen. Ebenso ist ein Hinweis anzubringen, dass die Daten mit osm2po konvertiert wurden.

3) Installation

- **osm2po** muss nicht installiert werden. Es wird lediglich in ein beliebiges Verzeichnis entpackt und ist dann sofort lauffähig.
- **osm2po** setzt ein installiertes Java™ 6 (oder höher) voraus.
- Liegt der Fokus auf den eigentlichen Daten, so sollte ebenfalls PostgreSQL (postgresql.org) mit einem PostGIS-Aufsatz (postgis.org) installiert sein.
- Sollen die Daten visualisiert werden, so ist der Einsatz eines GIS-Programms z.B. QuantumGIS (qgis.org) zu empfehlen.
- Liegt der Fokus zudem auf reinem Datenbank-Routing, so ist zusätzlich noch pgRouting (pgRouting.org) notwendig.
- Nach der Installationen sollte sicher gestellt sein, dass sich *java* und ggf. *psql* (PostgreSQL) im Systempfad befinden und von überall her per Kommando aufgerufen werden können.
- Sollten sich mehrere Java-Versionen auf dem Rechner befinden, so kann man mit dem Kommando `java -version` herausfinden, ob tatsächlich die Version 6 (1.6.x) am Start ist.

4) Schnelleinstieg

4.1 Funktionstest und erster Kontakt

Ich werde jetzt einen sehr kurzen Funktionstest durchführen, den jeder in nicht einmal 5 Minuten nachvollziehen kann. Voraussetzung ist lediglich, dass **osm2po** entpackt in einem beliebigen Verzeichnis vorliegt und Java per Kommandozeile aufrufbar ist.

Ferner ist bei den folgenden Beispielen genau darauf zu achten, dass keine Leerzeichen vor und nach den Gleichheitszeichen stehen und natürlich, dass hier mehrzeilig dargestellte Kommandos in Wirklichkeit eine zusammenhängende Zeile bilden.

Ich öffne jetzt eine Kommandozeilen-Konsole und, falls noch nicht geschehen, navigiere mittels **cd** (change directory) in das Verzeichnis, in dem sich die `osm2po-core-4.x.x.jar` selbst befindet. Alle weiteren Aufrufe finden relativ zu diesem Verzeichnis statt.

Folgendes wird **osm2po** gleich in einem Rutsch für mich erledigen:

1. Download der Daten fuer Hamburg.
2. Entpacken und konvertieren der Daten fuer pgRouting und osm2po-Routing
3. Erstellung von SQL-Insert Skripten.
4. Starten des osm2po-Service (Routing)

```
java -jar osm2po-core-4.xxx.jar prefix=hh  
http://download.geofabrik.de/osm/europe/germany/hamburg.osm.pbf
```

Am Ende der Verarbeitungskette sollte nun folgende Meldung erscheinen:

Waiting for requests at http://localhost:8888/Osm2poService

Jetzt können wir einen Browser öffnen und die Adresse einfach eingeben.

```
http://localhost:8888/Osm2poService
```

Wichtiger Hinweis:

Sollte sich der Browser jetzt weigern, überhaupt irgendwas anzuzeigen, dann liegt es womöglich an der lokalen Firewall-Einstellung. Diese muss zwingend lokale Services auf http://localhost:8888 durchlassen. Falls Port 8888 bereits belegt ist, so kann man diesen in der Konfiguration (*osm2po.config*) ändern.



Soll der Service zu einem späteren Zeitpunkt erneut gestartet werden, so müssen wir natürlich nicht jedes Mal vorher konvertierten. Dies ist ja bereits geschehen.
Es muss lediglich der Service neu gestartet werden:

```
java -jar osm2po-core-4.xxx.jar prefix=hh cmd=r
```

4.2 Was sonst noch passiert ist

hierfür sollten wir einen Blick in das aktuelle, sowie in das soeben generierte Verzeichnis `hh` werfen.
Folgende Text-Dateien lohnen einen kurzen Blick:

- `osm2po.config` Die Konfiguration für **osm2po**.
- `hh_2po.log` Die Log-Datei, die bei der Konvertierung geschrieben wurde.
- `hh_2po_4pgr.sql` Die konvertierten Daten als CreateTable und Insert SQL für PostGIS.

4.3 Daten mit `psql` in die Datenbank einspielen

Grundvoraussetzung hierfür ist ein installiertes PostgreSQL/PostGIS.

Wenn man die Konsolenmeldungen während der Verarbeitung aufmerksam verfolgt hat, so wird man eine Kopiervorlage für den nötigen Befehl entdeckt haben. Alternativ kann man auch die PostgreSQL-Dokumentation zum Kommandozeilenbefehl `psql` lesen oder im Nachgang diese Vorlage aus der Log-Datei heraus in die Konsole kopieren. Die Vorlage sieht in etwa wie folgt aus:

```
psql -U [username] -d [dbname] -q -f "/path/to/my/dir/hh/hh_2po_4pgr.sql"
```

Wir brauchen lediglich `[username]` und `[dbname]` gegen die von uns verwendeten Namen zu ersetzen. Bei mir heißt die Datenbank `osm2po` und der User `postgres`. Als Passwort wird ebenfalls `postgres` angenommen. Weitere `psql`-Parameter, so z.B. die Angabe eines Passwortes, können mit dem Befehl `psql --help` abgefragt werden.

Hier nun der Befehl, mit dem ich die korrespondierenden Daten in die Datenbank einspiele:

```
psql -U postgres -d osm2po -q -f "/path/to/my/dir/hh/hh_2po_4pgr.sql"
```

4.4 Daten unter QuantumGIS visualisieren

Hierzu öffnen wir das Programm und müssen einen Postgis-Layer hinzufügen
Ein Dialog erscheint, in dem wir unsere soeben erzeugte Hamburg-Tabelle auswählen können.



Sollte dieser Dialog keine Verbindungen zur Auswahl anbieten, so müssen wir diese im Vorwege anlegen. In Hierzu einfach auf **Neu** klicken und ein weiterer Dialog erscheint, in dem wir die Parameter eingeben können.

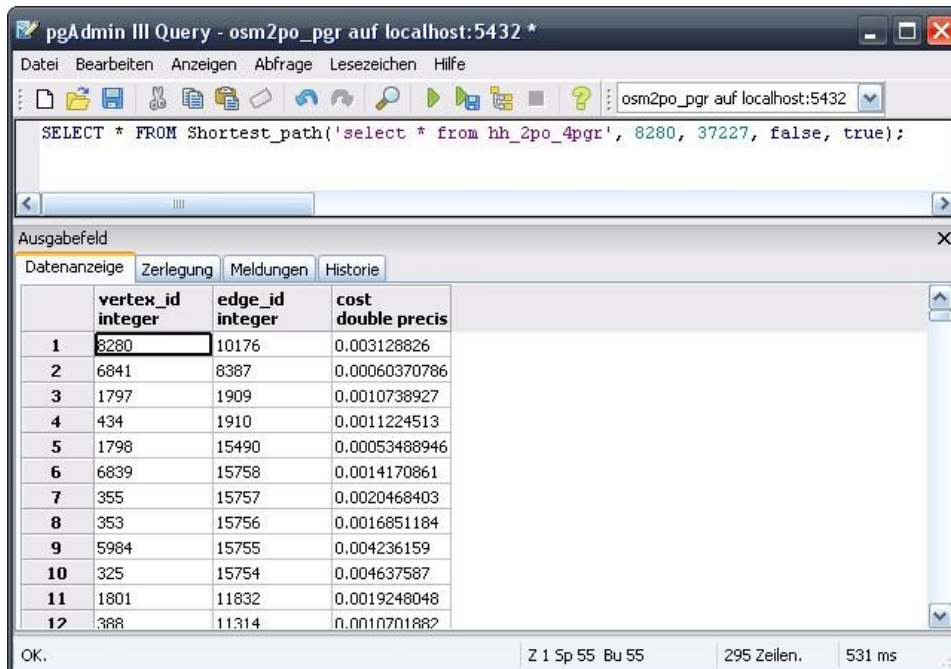


4.5 Kürzesten Weg mit pgRouting finden

Ist pgRouting bereits installiert, so können wir direkt auf der Datenbankebene routen. **osm2po** hat dafür die gleichen IDs herausgeschrieben, die es für sein eigenes Routing verwendet und ist somit vergleichbar.

Ich öffne hierzu das PostgreSQL-Tool **PgAdminIII** und führe folgenden Befehl aus, wobei ich die gleichen IDs für Source und Target wie im ersten Beispiel verwende.

```
SELECT * FROM Shortest_path('select * from hh_2po_4pgr', 8280, 37227, false, true);
```

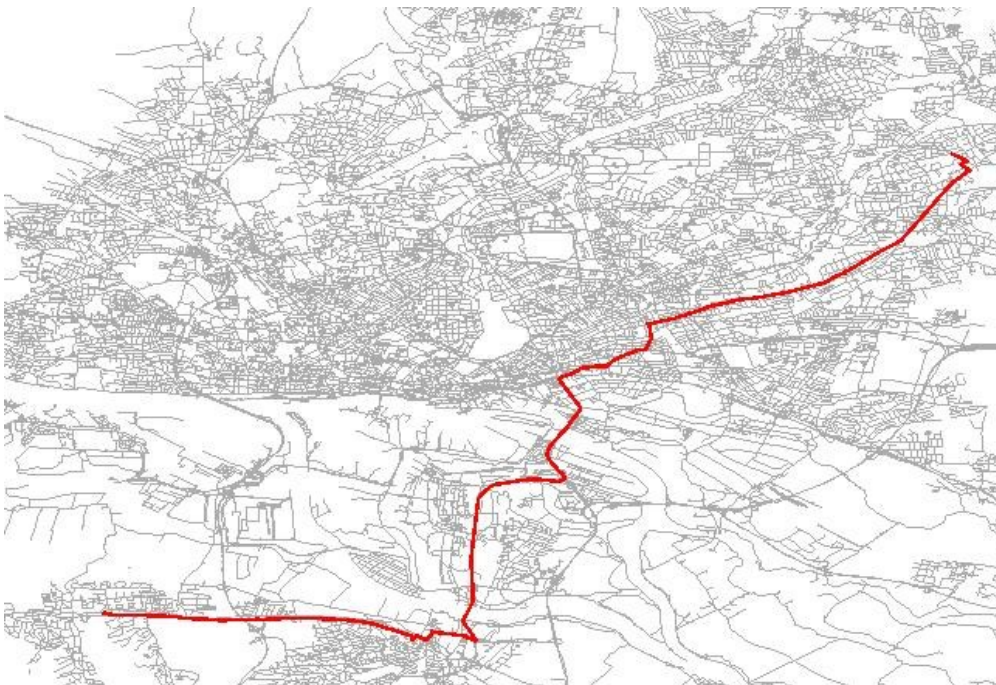
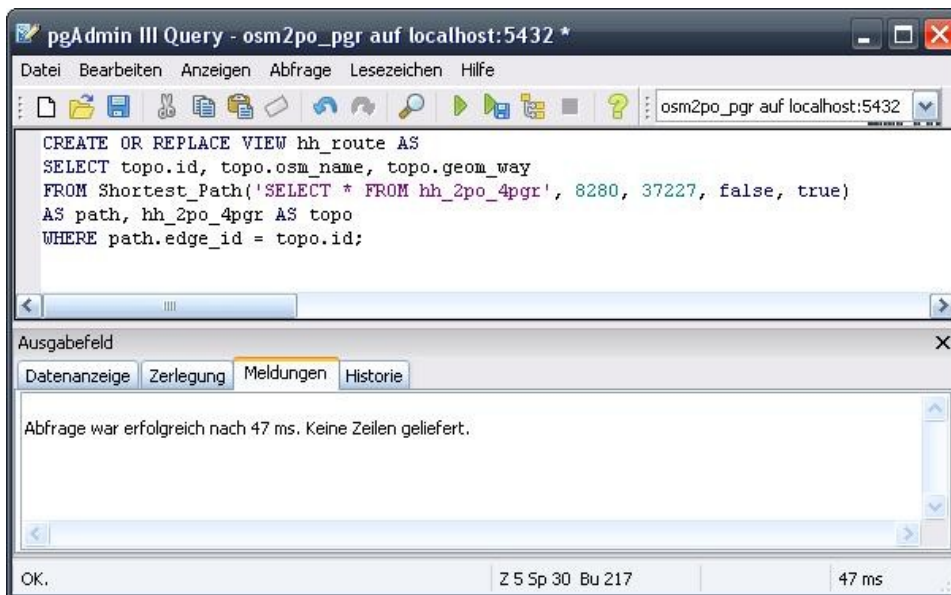


The screenshot shows the PgAdmin III Query tool interface. The query editor at the top contains the SQL command: `SELECT * FROM Shortest_path('select * from hh_2po_4pgr', 8280, 37227, false, true);`. Below the editor, the 'Ausgabefeld' (Output Field) tab is active, displaying a table of results. The table has four columns: 'vertex_id' (integer), 'edge_id' (integer), and 'cost' (double precis). The results are listed in 12 rows, with the first row highlighted. The status bar at the bottom indicates 'OK.', 'Z 1 Sp 55 Bu 55', '295 Zeilen.', and '531 ms'.

	vertex_id integer	edge_id integer	cost double precis
1	8280	10176	0.003128826
2	6841	8387	0.00060370786
3	1797	1909	0.0010738927
4	434	1910	0.0011224513
5	1798	15490	0.00053488946
6	6839	15758	0.0014170861
7	355	15757	0.0020468403
8	353	15756	0.0016851184
9	5984	15755	0.004236159
10	325	15754	0.004637587
11	1801	11832	0.0019248048
12	388	11314	0.0010701882

Für die Visualisierung unter QuantumGIS benötigen wir allerdings noch eine View, welche die gefundene Route mit den Geometrien verbindet. Hierzu müssen wir erneut PgAdminIII bemühen und folgendes SQL ausführen.

```
CREATE OR REPLACE VIEW hh_route AS  
SELECT topo.id, topo.osm_name, topo.geom_way  
FROM Shortest_Path('SELECT * FROM hh_2po_4pgr', 8280, 37227, false, true)  
AS path, hh_2po_4pgr AS topo  
WHERE path.edge_id = topo.id;
```



Tipp:

Natürlich können die Daten in QuantumGIS näher analysiert werden. Das heißt, wir können uns z.B. alle IDs oder Straßennamen anzeigen lassen. Ferner ist es möglich OSM-Daten direkt als Layer über unsere Daten zu legen, um so Unterschiede oder Fehler zu entdecken.

5) Probleme bei der Konvertierung

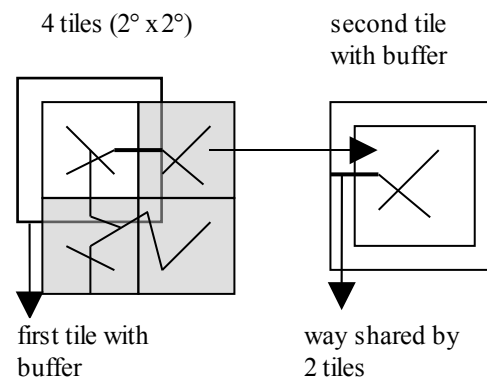
5.1 Prinzipielle

osm2po konvertiert in mehreren Schritten. Dies ist komfortabel und übersichtlich - man kann die Schritte als Stapel oder einzeln ausführen. Der eigentliche Grund liegt aber in der Anordnung der Daten in den OSM-Extrakten selbst. Es handelt sich hier schließlich nicht um eine Datenbank, die man beliebig befragen kann, stattdessen erhalten wir der Reihe nach eine Sammlung von Knoten, Wegen mit Referenzen auf die Knoten und Relationen, wiederum mit Referenzen auf Knoten und Wege. Die Größenordnung einer solchen Datei kann leicht 10 Gigabyte übersteigen.

osm2po geht bei der Konvertierung daher grob folgendermaßen vor:

1. Alle Knoten werden geografisch in Quadranten einsortiert (Tiling)
Wege und Relationen werden vorgefiltert und unsortiert extrahiert.
2. In einer Schleife werden möglichst viele Quadranten in den Speicher geladen und gegen die Wege aufgelöst. Das Resultat sind nach Quadranten sortierte Wege, wobei Wege die solche Quadranten kreuzen, extra behandelt werden. (Joining)
3. In einem letzten Schritt werden die Wege gelesen, auf Kreuzungen untersucht und ggf. segmentiert (Segmenting)

Dabei entsteht folgendes Problem: Wege kümmern sich nicht um Grenzen. Daher muss jedem Quadranten ein Puffer bzw. Überlappungsbeich zugeordnet werden. Ist dieser zu klein, so kann ein Weg ggf. nicht aufgelöst werden. Die Standardeinstellung für diesen Puffer beträgt 0.1 Grad. Dies kann aber per Konfiguration vergrößert werden. Je größer also dieser Wert, desto genauer arbeitet das Programm. Ist der Wert hingegen zu groß, so müssen unnötig viele Bytes bewegt und gespeichert werden. Ein Puffer von 0.1 ist ein guter Richtwert, bei dem z.B. die *germany.osm* verlustfrei konvertiert werden konnte.



Per Konfiguration lässt sich auch eine alternative Verhaltensweise einstellen, in der in Schritt 1 Knoten extrahiert und danach erst sortiert werden. Der Vorteil liegt hier in der sehr frühen Erkennung von relevanten Knoten und der drastischen Reduktion der Menge auf selbige. Somit können größere Gebiete auf einmal verarbeitet werden, was wiederum die Fehlerquote an den Grenzen der Quadranten mindert. Allerdings leidet die Geschwindigkeit darunter und es können Speicherprobleme in Schritt 3, also zu einem späten Zeitpunkt, eintreten.

Zudem ist es möglich, komplett auf Tiling zu verzichten, was bis zu Größen einer *germany.osm* auch möglich ist. Und dies sogar auf einem älteren PC mit lediglich 2 Gigabyte Speicher. Für moderne Rechner mit 6 Gigabyte und mehr ist selbst die *europa.osm* so verlustfrei konvertierbar.

Die Art des Tilings bestimmt also maßgeblich die Qualität der Daten und die Geschwindigkeit der Konvertierung. Der entsprechende Parameter hierfür heißt `tileSize` und wird weiter unten noch genauer vorgestellt.

5.2 OSM-Daten

OpenStreetMap (OSM) ist im Prinzip ein großes WIKI für GEO-Daten. Jeder kann mitmachen. Jeder kann Dinge richtig machen, aber auch falsch. Es gibt also eine Reihe falscher oder unlogischer Daten, die nicht

mehr interpretierbar sind. So zum Beispiel ist das Taggen (Anbringen von Informationen) für Fähren oder Autozüge nicht einheitlich geregelt. **osm2po** kann an solchen Stellen nur Annahmen treffen. Dies geht sogar so weit, dass Dinge, die gestern noch funktionierten, heute plötzlich völligen Unsinn produzieren. „Krumme Touren“ sind zumeist Fehler in OSM-Daten und kein Fehler von **osm2po**.

5.3 OSM Relationen

Die Relationen in OSM bündeln Knoten und Wege oder wiederum andere Relationen und erlauben die Angabe weiterer Meta-Informationen für eine derartige Zusammenstellung. Klassische Beispiele hierfür sind Transitstrecken, Buslinien oder Bundesstraßen, welche gewöhnlich aus mehreren Wegen bestehen. Auch Haltestellen für Bus- oder Bahnlinien werden hier, zumeist als Knoten, erwähnt.

Relationen können sinnvoll sein, allerdings zeigt die Realität, dass diese nicht ganz so einfach zu editieren oder interpretieren sind wie ursprünglich gedacht. Das Editieren einer Relation ist für viele OSM-Freiwillige ein Grauen und wird daher weitestgehend gemieden. Entsprechend sehen teilweise die Daten aus. Auch **osm2po** ist nur begrenzt in der Lage, Informationen aus Relationen zu extrahieren. Es bedurfte eines halben Kraftakts alleine die Abbiegevorschriften zu verarbeiten.

5.3.1 Abbiegevorschriften in OSM

Die Abbiegevorschriften (engl. turn restrictions) werden in OSM bis auf wenige Ausnahmen richtig angegeben. Allerdings existieren auch hier Exoten, die **osm2po** nicht interpretieren kann und will. **osm2po** kümmert sich ausschließlich um Relationen, bei denen folgende vier Informationen zusammentreffen:

1. ID des Weges auf dem man sich befindet (from)
2. ID des Knotens (Kreuzung), an dem die Vorschrift (Straßenschild) angebracht ist. (via)
3. ID des Weges, in den abgelenkt werden soll (to)
4. Die Vorschrift selbst, so z.B. *no-right-turn* oder *only-left-turn*.

Einbahnstraßen sind implizite Vorschriften und werden in **osm2po** extra behandelt.

5.4 Speicher

Wie bereits weiter oben erwähnt, muss **osm2po** viele Daten cachen. Je mehr Speicher wir also reservieren, desto besser und schneller ist die Verarbeitung. Allerdings ist der Speicher physisch durch die Hardware begrenzt. Zudem ist die Kontrolle dieses Speichers unter Java nur bedingt möglich. Java kennt den Parameter **-Xmx**, der diesen Heap-Speicher für uns reserviert.

Unter einem 32-Bit Java ist die Voraussage des noch zur Verfügung stehenden Speichers recht zuverlässig. Sobald allerdings ein Java mit 64-Bit zum Einsatz kommt, ist eigentlich kaum noch etwas vorherzusagen. Nichtsdestotrotz ist es möglich, Werte größer 1.4 Gigabyte anzugeben. Ein **-Xmx4g** ist also auch für **osm2po** möglich.

Im Joining-Lauf (Schritt 2) kann diese Fehleinschätzung auch per Konfiguration mittels des Parameters **reservedXmx** verhindert werden. Dies passiert allerdings auch automatisch, wenn **osm2po** hier in einen OutOfMemory-Fehler läuft, feststellt, dass eigentlich noch 800 Megabyte hätten übrig sein müssen und dann diesen Wert neu kalibriert. Sollte das Programm dennoch mit einem OutOfMemory abbrechen, dann hilft nur noch die Erhöhung von **Xmx** oder die Verringerung von **tileSize**.

6) Referenz

6.1 Aufruf

Das grundsätzliche Aufrufmuster sieht wie folgt aus:

```
java [-Xmx] -jar osm2po.jar [param=value]* [InputFile]
```

Die Quelldatei (InputFile) wird nur in der ersten Verarbeitungsstufe, dem Tiling benötigt. Wird der Parameter nicht angegeben, so wartet **osm2po** auf Eingaben von der Standardeingabe (stdin). Befindet sich die Quelldatei im Internet unter einer http-Adresse, so wird dies ebenfalls berücksichtigt und eine entsprechende Verbindung aufgebaut. Der Parameter -Xmx ist ein Java- und kein osm2po-Parameter, ist aber einer der wichtigsten Schalter, von dem das Gelingen einer Konvertierung maßgeblich abhängt. Er sorgt nämlich für die Reservierung des Speichers.

6.2 Konfiguration

Parameter werden in **osm2po** immer als Name=Wert Paare angegeben. Es wird zwischen Groß- und Kleinschreibung unterschieden. `TileSize` ist also nicht gleich `tileSize`. Sämtliche Parameter können per Konfiguration voreingestellt und auf der Kommandozeile überschrieben werden. Setzt man `Name=` (ohne Wert), so wird der Parameter intern gelöscht.

Werden die Parameter auf der Kommandozeile eingegeben, so dürfen sie keine Leerzeichen enthalten, also auch nicht vor und hinter dem Gleichheitszeichen.

Die Verarbeitungsreihenfolge ist geläufig:

1. Auslesen der Konfigurationsdatei.
2. Übersteuerung der Einstellungen durch Kommandozeilen-Parameter.

Nach der Konfigurationsdatei wird wie folgt gesucht:

1. Externe Konfigurationsdatei, die durch den Parameter `config=<Pfad>` bei Aufruf festgelegt wird. Es dürfen hier keine Leerzeichen eingegeben werden. Hochkommas werden nicht erkannt.
2. Datei namens `osm2po.config`, wenn diese sich im aktuellen Verzeichnis befindet.
3. `osm2po.config` innerhalb des verpackten Programms selbst.

6.3 Die wichtigsten Parameter

6.3.1 TileSize und -Xmx

`tileSize` bestimmt die Größe eines Quadranten (Tile) und des Überlappungsbereichs (Puffer).

Bei der Konvertierung größerer Gebiete, also Länder und Kontinente, sollte dieser Wert zwingend erhöht werden. Dies setzt allerdings auch einen größeren Cache voraus. Dieser Speicher wird unter Java als Heap bezeichnet und kann mittels des Parameters **-Xmx** eingestellt werden.

Beispiele für Xmx: `-Xmx512m`, `-Xmx4g`, `-Xmx1280m`

Der höchste Wert für ein 32 Bit-Java liegt bei ungefähr **-Xmx1408m**.

Wenn wir mit höheren Werten, z.B. `-Xmx4g`, unter einem 64-Bit-Java arbeiten, dann kann zusätzlich der osm2po-Parameter `reservedXmx` auf z.B. 1024 (1 Gigabyte) gesetzt werden, was zumindest im sehr kritischen zweiten Verarbeitungsschritt eine Fehleinschätzung des noch zur Verfügung stehenden Speichers verhindert.

`tileSize` selbst kennt folgendes Parameter-Muster:

```
tileSize = [<Y>x<X>|x][,<Puffer>][,<Schalter>]
```

Alle Parameter sind optional und die Reihenfolge ist beliebig.

[<Y>x<X> x]	Größe eines Quadranten (Tile), also Y mal X, bzw. Breite mal Länge. Hier dürfen nur ganze Zahlen verwendet werden. Optimalerweise sollten diese Werte Teiler von 90 bzw. 180 (Grad) sein. Dies ist zwar nicht zwingend, jedoch bei der Konvertierung der ganzen Welt aufgrund der Überschneidungen an den Meridianen sinnvoll. Soll osm2po komplett ohne Tiling arbeiten, so ist einfach ein kleines x zu setzen. Die Standardeinstellung ist 1x1 (Grad).
[<Puffer>]	Dies ist der oben bereits erwähnte Überlappungsbeich der Quadranten. Die Standardeinstellung ist 0.1 (Grad), was ungefähr 10 km bedeutet. osm2po berücksichtigt dabei, dass aufgrund der Erdkrümmung dieser Wert zu den Polen hin vergrößert werden muss. Auf dem 60. Breitengrad wird bei 0.1 also intern mit 0.2 für den X-Wert, also der geografischen Länge gerechnet.
[<Schalter>]	Eine Zeichenkette bestehend aus Kleinbuchstaben, die weitere Einstellungen ermöglicht. Derzeit existiert hier nur ein einziger Zusatzschalter, nämlich das kleine c . c steht für Compress, was dazu führt, dass in Schritt 1 der Verarbeitungskette Knoten erst extrahiert und danach sortiert werden. Beim Lesen der Wege werden dann alle verwendeten Referenzen vermerkt und anschließend nur die Knoten weiterverarbeitet, die tatsächlich die Wege und nur diese bilden. Dies reduziert die Menge für den zweiten Verarbeitungsschritt erheblich.

6.3.2 cmd

osm2po ist im Kern eine Aneinanderreihung vordefinierter und frei definierbarer Verarbeitungsschritte. Dies wird auch Stapel (engl. Batch) genannt. Dieser Schalter weist **osm2po** an, bestimmte Schritte auszuführen. Einige dieser Schritte bauen zwingend aufeinander auf, andere, zumeist die frei definierbaren, sind eher im Nachlauf (Postprozess) angesiedelt. Jedem Verarbeitungsschritt ist ein Kleinbuchstabe zugeordnet. So bedeute das kleine **t** Tiling, also den ersten Verarbeitungsschritt.

Im Gegensatz zu den Folgeschritten benötigt das Tiling natürlich die Angabe einer Quelldatei. Alle weiteren Schritte arbeiten dann verzeichnisbasiert. Dieses Arbeitsverzeichnis wird mit dem Parameter `workDir` oder `prefix` festgelegt.

Der Standardwert für `cmd` ist **tjsppr**. Die Bedeutung wird in der folgenden Tabelle ersichtlich:

Kommando	Bedeutung	Abhängigkeit	Kombinierbar	erfordert Quelle	Erläuterung
t	Tiler		ja	ja	Liest die Quelldatei aus OSM entweder lokal oder direkt aus dem Internet, wenn der Pfad mit <code>http://</code> beginnt. Wird hierfür kein Parameter angegeben, dann wartet das Programm auf Eingaben von der Konsole (stdin). Der eigentliche Verarbeitungsschritt wurde oben bereits zum Thema <i>tileSize</i> bzw. <i>Tiling</i> ausführlich erläutert.
j	Joiner	t	ja	nein	Führt die Informationen der Knoten, Wege und Relationen zusammen.
s	Segmenter	j	ja	nein	Erkennt Kreuzungen und teilt Wege entsprechend neu. Ferner werden hier neue, fortlaufende Ids vergeben, welche die Grundlage für das spätere Routing bilden.
p	Postprocessors	s	ja	nein	Ruft alle konfigurierten, nachgelagerten Prozesse auf. Hierunter fällt z.B. auch die Erstellung der SQL-Script-Dateien für PostGIS.
g	Graph-Builder	s	ja	nein	Erstellt den Graphen für das osm2po-Routing.
r	Routing	g	ja	nein	Startet das osm2po-Routing als Http-Service.
c	Kurzform. Setzt <code>tjsp</code>		ja	ja	Für Schreibfaule.
d	Debug / Dump		nein	ja	Gibt *.2po Dateien in lesbarer Form auf der Konsole (stdout) aus.

6.4 Schnellübersicht weiterer Parameter

tileThreshold	Bestimmt die Anzahl der Knoten, die mindestens in einem Quadranten liegen müssen, damit überhaupt eine weitere Verarbeitung für diesen Quadranten angestoßen wird. Der Standardwert ist 0, also alle berücksichtigen.
excludeWrongWays	Beim Aufbau des Graphen für das osm2po-routing werden Segmente in Kanten umgewandelt. Jedes Segment erhält so eine Hin- und eine Rückrichtung. Handelt es sich allerdings um eine Einbahnstraße, so ist es nicht zwingend erforderlich die entsprechende Rückwärts-Kante zu generieren. Für ein Fahrrad-Routing kann dies allerdings sinnvoll sein. Der Standardwert ist <code>false</code> , also berücksichtigen.
log.console.level	Einstellung des Schweregrads von Meldungen in der Konsole. Mögliche Werte sind der Reihe nach <code>debug</code> , <code>progress</code> , <code>warn</code> , <code>info</code> , <code>error</code> , <code>fatal</code> Der Standardwert ist <code>info</code> .
log.console.to	Konsolenmeldungen an folgendes Standardgerät schicken. Mögliche Werte sind <code>out</code> , <code>err</code> , oder <code>null</code> für keine Meldungen.
pipeOut	Weist Postprozesse, die normalerweise Dateien schreiben an, die Ausgaben nach <code>stdout</code> zu schreiben. Kombiniert man also <code>log.console.to=err</code> und <code>pipeOut=true</code> , so lässt sich <code>osm2po</code> in einer Verarbeitungskette wie z.B. <code>curl bzcata osm2po psql</code> integrieren. Zu empfehlen ist hier <code>curl</code> , <code>bzcata</code> und <code>psql</code> im Quiet-Modus zu fahren und nur <code>osm2po</code> erlauben, Meldungen auszugeben.
prefix	Sorgt für die Eindeutigkeit sehr vieler Eingabe- und Ausgabeinformationen. Das Prefix ist frei wählbar. Es macht aber Sinn dieses kurz und beschreibend zu wählen. z.B. <code>hh</code> für Hamburg oder <code>de</code> für Deutschland. Wird dieser Parameter nicht explizit gesetzt, so wird <code>osm</code> als Prefix verwendet.
workDir	Dies ist das Arbeitsverzeichnis, in dem Cache-Dateien und Ergebnisse, z.B. SQL-Skripte erzeugt werden. Der Standardwert ist <code>workDir=prefix</code>
config	Pfad/Dateiname zu einer anderen auch andersnamigen Konfigurationsdatei
postp.n.class	Pluginmechanismus für die Implementation eigener Postprozessoren.

Hinweis:

Es existieren noch weitere Parameter, die hier nicht aufgelistet sind.

Diese sind in der `osm2po.config` nachzulesen und auch beschrieben.

7) Tipps und Beispiele

7.1 Analyse und Vorgehensweisen

Ich habe nun ausführlich über die Probleme bei der Konvertierung berichtet. Ferner habe ich den Leser mit Informationen regelrecht bombardiert. Es muss jetzt so wirken, als sei **osm2po** kompliziert. Dies ist aber nicht so. Es ist kein Hexenwerk. Es existiert nur leider kein Königsweg bei der Konvertierung. Deswegen muss **osm2po** in der Lage sein, verschiedene Strategien zu fahren. Schließlich gibt es dicht besiedelte Gebiete, z.B. Deutschland und riesige Landflächen mit wenigen Straßen, z.B. Norwegen. Andere Gebiete sind zwar dicht besiedelt und groß, jedoch kaum kartografiert.

Bevor wir Daten aus OSM konvertieren, sollten wir uns also einen Überblick über die Größe eines Extrakts und der tatsächlichen Flächenabdeckung machen.

Wir sollten uns an bereits erfolgreich konvertierten Gebieten orientieren und deren Größen und Datenmengen relational gegenüberstellen. Dann sollten wir wissen, wieviel Speicher uns für die Konvertierung zur Verfügung steht. Mit diesem Wissen können wir eine gute Vorhersage treffen, so dass die Konvertierung schon beim ersten Durchlauf optimale Ergebnisse liefert.

Ich ziehe grundsätzlich zwei Extrakte heran. Zum einen ist dies die *germany.osm*, zum anderen ist dies die *europa.osm*, die sogar größer als die *united-states.osm* ist. Beide Extrakte kann **osm2po** problemlos konvertieren. Und dies sogar auf einem alten Rechner mit gerade einmal 2 Gigabyte Speicher.

Da die Datenmengen allerdings stetig wachsen, sollten wir die Eckwerte nicht zu sehr ausreizen, uns also noch ein wenig Luft nach oben lassen.

Gerade beim ersten Durchlauf sollten die Meldungen während der Konvertierung verfolgt werden und hier im Speziellen die Ausgaben für den noch verfügbaren Speicher. Werden diese Werte kaum an die Grenze gefahren, dann sollten wir evtl. einen zweiten Anlauf mit einer größeren **tileSize** wagen.

Ein Blick in das Arbeitsverzeichnis **workDir** lohnt sich ebenfalls. Hier werden für jedes zu cachende Tile temporäre Dateien erzeugt. Haben wir eine zu kleine **tileSize** gewählt, so kann die Zahl leicht in die Hunderte, gar Tausende steigen. Letzteres sollte optimalerweise nicht passieren.

Auch die Konfiguration spielt hier eine Rolle. Wenn wir diese so wählen, dass nur Autobahnen und Landstraßen konvertiert werden, dann sind die Vorzeichen wiederum andere.

7.2 Richtwerte für -Xmx und tileSize

32 Bit Rechner mit 2 Gigabyte RAM

- *germany.osm* -Xmx512m 10x10
- *germany.osm* -Xmx1408m x
- *germany.osm* -Xmx1408m x,c
- *europa.osm* -Xmx1408m 5x5
- *europa.osm* -Xmx1408m 10x10,c

64 Bit Rechner mit 8 Gigabyte RAM

- *germany.osm* -Xmx2g x
- *europa.osm* -Xmx3g 45x45,c
- *europa.osm* -Xmx5g x,c

7.3 Beispiele

Landmasse	Besiedlung	System / RAM	Beispiel	Tiles	Verlustquote
sehr klein	sehr dicht	32 Bit / 1 Gig	java -jar osm2po.jar prefix=hh hamburg.osm		0
mittel	dicht	32 Bit / 2 Gig	java -Xmx512m -jar osm2po.jar prefix=de germany.osm.pbf tileSize=10x10,0.2 cmd=c		0
			java -Xmx1408m -jar osm2po.jar prefix=de tileSize=x,c germany.osm.pbf	1	0
groß	dicht	32 Bit / 2 Gig	java -Xmx1408m -jar osm2po.jar prefix=eu europe.osm.pbf tileSize=5x5,0.2 cmd=tjsp	125	37 auf 10 Mio
			java -Xmx1280m -jar osm2po.jar prefix=eu europe.osm.pbf tileSize=5x5,1,c cmd=tjsp	93	2 auf 10 Mio
			java -Xmx1408m -jar osm2po.jar prefix=eu tileSize=10x10,0.2,c cmd=tjs europe.osm.pbf	41	16 auf 10 Mio
			curl -s --location http://download.geofabrik.de/osm/europe.osm.bz2 bzipcat -c java -Xmx1280m -jar osm2po.jar prefix=eu tileSize=5x5,0.2 cmd=c	125	37 auf 10 Mio
		64 Bit / 4 Gig	java -Xmx3g -jar osm2po.jar prefix=eu tileSize=10x10,0.2 europe.osm.pbf cmd=c	41	16 auf 10 Mio
			java -Xmx3g -jar osm2po.jar prefix=eu tileSize=45x45,0.2,c europe.osm.pbf cmd=c	4	7 auf 10 Mio
		64 Bit / 6 Gig	java -Xmx5g -jar osm2po-core-4.jar prefix=eu tileSize=x,c europe.osm.pbf cmd=tjsp	1	0

Anmerkung:

Die Verlustquote beschreibt die Anzahl der Wege, die aufgrund des zu geringen Überlappungsbeichs (Puffer) nicht konvertiert werden konnten. Es ist ersichtlich, dass die Qualität der Daten mit der Reduktion der Anzahl der Tiles und / oder der Erhöhung des Puffers wächst.

8) osm2po Routing

Das Routing in **osm2po** arbeitet völlig autark und ist seit der Version 4 nicht mehr von *PostGIS* abhängig. Es basiert im Kern auf einem simplen Dijkstra-Algorithmus, berücksichtigt jedoch auch Einbahnstraßen und Abbiegevorschriften. Ebenso kann zwischen einer Kürzesten- und einer Schnellsten-Weg-Suche gewählt werden. Das **osm2po-Routing** dient hauptsächlich der Daten-Validation. Es ist allerdings nicht zu unterschätzen, denn für Java-Verhältnisse ist es recht schnell und begnügt sich mit relativ wenig Speicher. Es kann direkt als Http-Ajax-Service verwendet und so ganz einfach in bereits bestehende Server-Landschaften integriert werden. Es ist natürlich auch möglich, osm2po als Bibliothek einzubinden und so das Routing direkt anzusprechen. Von einer Servlet-Schnittstelle habe ich allerdings bisher abgesehen, obwohl dem technisch eigentlich nichts im Wege stehen würde. Es ist auch nicht das Ziel, sämtliche Anwendungsfälle, die ein Routing ermöglicht, abzubilden. Das wäre zu viel des Guten. Ich habe hier eine Basis geschaffen, auf der viele Dinge in Zukunft möglich sein werden.

Ich möchte an dieser Stelle jedoch nicht weiter auf dieses Thema eingehen. Bis dato wird osm2po überwiegend als Konvertierer verwendet. Sollte ich mich hier täuschen, so darf der interessierte Leser mich gerne per email interviewen.