

Institut für Praktische Informatik und Medieninformatik
Fakultät für Informatik und Automatisierung
Fachgebiet Telematik / Rechnernetze

Projektseminar

Extraktion eines Bewegungsmodells basierend auf Kartenmaterial von OpenStreetMap

(zur Simulation von Bewegungen innerhalb eines verteilten Kamerasytems)

vorgelegt von: Steve Göring
Matrikel: 43952
Betreuer: Prof. Dr.-Ing. Günter Schäfer
Dipl.-Ing. René Golembewski

Ilmenau, den 1. April 2013

Inhaltsverzeichnis

1 Einleitung	2
2 Grundlagen	3
2.1 OpenStreetMap	3
2.1.1 Export der XML Daten / Großstädte	3
2.1.2 Aufbau des OpenStreetMap Export Formats	4
2.1.3 Transporttypen	5
2.1.4 sphärische Koordinaten	5
2.2 Graphen	6
2.2.1 Schichten	6
2.2.2 Routinggraph	7
2.2.3 Dot	8
2.2.4 Grapheigenschaften/ Metriken	8
2.3 Formales Bewegungsmodell	10
2.3.1 Einschränkungen und Festlegungen	10
2.3.2 Modell	12
2.4 Simulationsframework	13
3 Implementierung/ Integration	14
3.1 OSMConverter	14
3.1.1 Aufbau	14
3.1.2 Ablauf	17
3.1.3 Ausgabeformate	18
3.2 Integration in die Simulation	19
3.2.1 TrafficGenerator	19
3.2.2 Erweiterungen im Bewegungsmodell	20
4 Auswertung	21
4.1 Großstädte vs. generierte Topologien	21
4.1.1 Berlin	21
4.1.2 synthetische Vergleichstopologie- Berlin	23
4.1.3 Moskau	24
4.1.4 synthetische Vergleichstopologie- Moskau	25
4.1.5 Weitere Städte	26
4.2 Qualitätsüberlegungen	27
5 Fazit	29
5.1 Zusammenfassung	29
5.2 Ausblick und offene Punkte	29
Literaturverzeichnis	30

1 Einleitung

Smart Camera Systeme erfreuen sich einer zunehmenden Beliebtheit. Sie sind aktuelles Forschungsobjekt, so wird auch in [GRS11] und [GSG12] ein Simulationframework für verteilte Smart Camera Systeme beschrieben.

Es gibt verschiedene Einsatzszenarien für Smart Camera Systeme, zum Beispiel können sie in Großstädten Verwendung finden. Dabei können zum einen verschiedene Überwachungsanwendungen (Autokennzeichen, Personensuche, vergleiche Abbildung 1.1) betrachtet werden, aber auch intelligente Navigationsaufgaben übernommen werden (vergleiche [GGS13]).



Abbildung 1.1: Videokamera in einer Stadt, Quelle: [[wikib](#)]

Betrachtet man den Einsatz in Großstädten so ist bei der simulativen Untersuchung solcher Camera Systeme eine realistisches modellgetriebenes Abbilden von Großstädten nötig. Aktuell benutzt die Simulation ein einfaches Bewegungsmodell, welches nur zufällige Bewegungen von Menschen zu Fuß betrachtet. Dagegen gibt es aber in Großstädten durchaus mehr Möglichkeiten sich von einem Ort zu einem anderen zu bewegen. In Städten findet man viele verschiedene öffentliche Verkehrsmittel, zum Beispiel U-Bahn-, S-Bahn-, Bus-, Zugverbindungen. Außerdem gibt es auch noch andere Bewegungsmittel (Auto, Fahrrad, City-Roller,...), welche aber für dieses Projektseminar nicht betrachtet werden sollen. Die genannten öffentlichen Verkehrsmittel fließen aktuell nicht in das Bewegungsmodell ein.

Daher soll in diesem Projektseminar aufbauend auf OpenStreetMap ein realistisches Bewegungsmodell zur Simulation von Smart Camera Systemen entwickelt werden. Dabei sollen Bewegungen innerhalb öffentlicher Transportnetze von Großstädten untersucht und modelliert werden. Es sollen nicht nur Kamerapositionen, sondern auch Bewegungsabläufe innerhalb des öffentlichen Verkehrsnets beachtet werden.

Aufbau. Zunächst sollen im Folgenden Kapitel 2 alle (theoretischen) Grundlagen beschrieben werden um mittels OpenStreetMap eine Topologie aufzubauen, welche als Grundlage für das Bewegungsmodell dienen soll. Anschließend werden in Kapitel 3 die beschriebenen Grundkonzepte umgesetzt, zum einen als extra Tool zur Extraktion der Topologie und zum anderen als Erweiterung der bestehenden Simulation. Am Ende in Abschnitt 4, werden die ermittelten Modelle für verschiedene Großstädte mit den aktuell synthetisch ermittelten Topologien verglichen und ausgewertet.

2 Grundlagen

Im folgenden Kapitel sollen Grundlagen zu OpenStreetMap und zur allgemeinen Darstellung von Routinggraphen beschrieben werden, damit anschließend das Bewegungsmodell formal dargestellt werden kann. Das Modell soll später in das Simulationsframework integriert werden, daher wird auch das Framework kurz vorgestellt.

2.1 OpenStreetMap

OpenStreetMap (OSM) ist ein Projekt zum Erzeugen einer freien Weltkarte, die vergleichbar mit Google Maps ist. Dabei bietet OpenStreetMap alle gesammelten Daten frei verfügbar als Rohmaterial oder in vorberechneten Kartenausschnitten an [OSMb, HW08]. Die OSM Daten werden zum größten Teil von Benutzern zur Verfügung gestellt. Freiwillige zeichnen dabei mittels GPS-Empfänger Routen (Straßen, Wanderwege, Fußwege, Zugverbindungen und weitere) auf und ordnen später zusätzliche Informationen den Rohdaten zu. Weiterhin stammen einige Daten auch aus anderen Quellen wie zum Beispiel *bing*.

Zur Erzeugung eines Bewegungsmodells werden zunächst Exportdaten von Großstädten benötigt.

2.1.1 Export der XML Daten / Großstädte

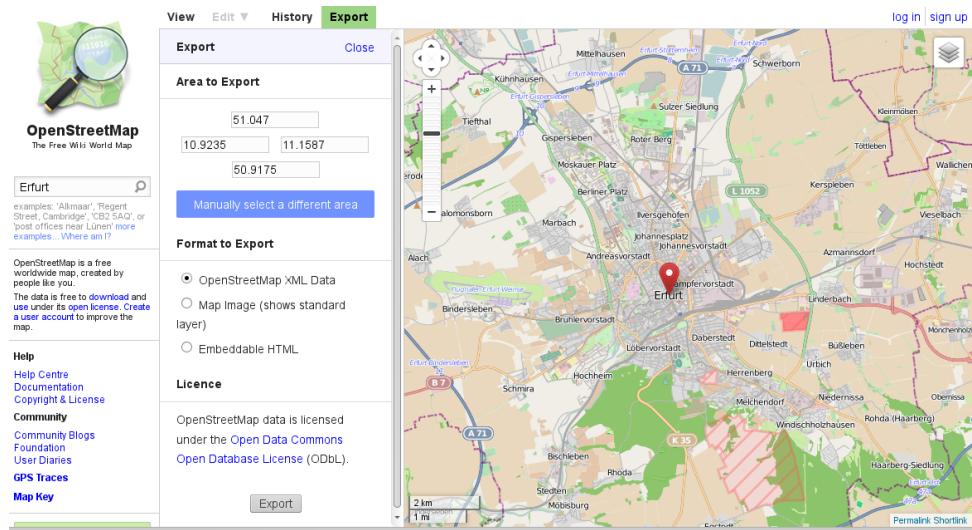


Abbildung 2.1: Manueller Export eines Kartenausschnitts mit der Web-Gui von OpenStreetMap <http://www.openstreetmap.org/?lat=50.982&lon=11.0414&zoom=12&layers=M>

Der Export des Rohmaterials kann bei OpenStreetMap auf verschiedenen Wegen erfolgen. Zunächst kann in der Web-GUI des Projektes direkt ein Export eines aktuell betrachteten Kartenausschnitt erfolgen (vergleiche Abbildung 2.1).

Die Webanfragen sind aber auf 50000 Knoten begrenzt (wie exemplarisch beim Versuch des Exports von Berlin nachgewiesen werden kann), daher ist es nicht direkt möglich OSM-Daten von Großstädten zu exportieren.

OpenStreetMap bietet aber nicht nur Webanfragen, sondern auch die Möglichkeit alle Daten des gesamten Planeten zu betrachten. Dazu kann die “planet.osm” Datei heruntergeladen werden (siehe [OSMe]). Sie umfasst alle weltweiten gespeicherten OpenStreet Daten und hat eine unkomprimierte Größe von ungefähr 250 GB.

Die “planet.osm” Datei könnte als Grundlage zum Ermitteln der jeweiligen Großstädte dienen, dazu muss die Datei mehrmals geparsst werden. Zunächst muss der Bereich der Großstadt in einer Anfrage ermittelt werden, anschließend müssen alle Knoten, Wege und Relationen innerhalb des Bereichs selektiert und exportiert werden, damit die Weiterverarbeitung nicht auf der gesamten “planet.osm” Datei erfolgen muss. Der Prozess gestaltet sich als zeitkritisch und aufwändig, daher bietet OpenStreetMap auch Exportdaten für einige bekannte Großstädte unter [OSMa] an. In [OSMa] sind ungefähr 140 Großstädte zu finden, dabei ist das exportierte Kartenmaterial relativ aktuell (September 2012) und es werden Städte von verschiedenen Kontinenten bereitgestellt. Auch verschiedene Exportformate werden angeboten. Alle exportierten OSM Daten haben dabei den gleichen strukturellen Aufbau.

2.1.2 Aufbau des OpenStreetMap Export Formats

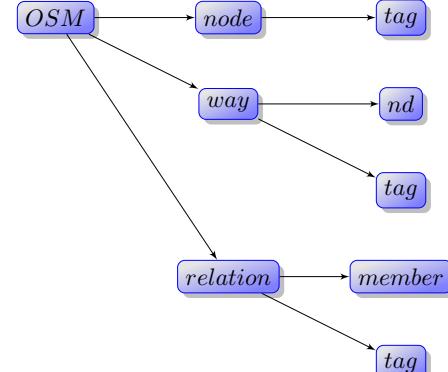
Das bevorzugte Exportformat zur Weiterverarbeitung ist dabei XML. Eine typische OpenStreetMap Datei besteht immer aus mehreren “node”, “way” und “relation“ Tags. Strukturell kann der Aufbau wie in Listing 2.2a dargestellt werden.

```

<osm>
  <node id="$ID" lat="$LAT" lon="$LON">
    <tag k="$KEY" v="$VALUE"/>
  </node>
  <way id="$ID" >
    <nd ref="$REF" />
    <tag k="$KEY" v="$VALUE"/>
  </way>
  <relation id="$ID" >
    <member type="$TYPE" ref="$REF" />
    <tag k="$KEY" v="$VALUE" />
  </relation>
</osm>

```

(a) XML Aufbau (nur wichtige Informationen wurden dargestellt)



(b) logischer Aufbau

Abbildung 2.2: OSM Dateiaufbau

Allgemein besteht ein von OSM exportierter Kartenausschnitt aus den genannten Hauptteilen, die jeweils noch verzweigt sind (vergleiche Abbildung 2.2b und Listing 2.2a):

- ▷ “node“ Tags: erhalten eine eindeutige \$ID, speichern weiterhin die Position(\$LAT, \$LON) und weitere Informationen zu einem GPS-Messpunkt der Karte
- ▷ “way“ Tags: sind eindeutig durch eine \$ID bezeichnet, speichern zusammenhängende Wegmesspunkte, die durch das typische OSM Tracking der Benutzer hervorgehen, dabei werden alle dazugehörigen Knoten (mittels “nd“ Tags und Referenzen auf die jeweiligen Node \$IDs)

und meist auch Informationen über die Art des Weges (Wegtyp, zum Beispiel: Wanderweg, Straße, Schiene) oder Namen gespeichert.

- ▷ “relation“ Tags: speichern Metawissen über die Zugehörigkeit von Wegen und Knoten (mittels “member“ Tag), zum Beispiel Gebäudeumrisse, Wanderwege, spezielle Gebiete, öffentliche Transportlinien

Der “relation“ Tag stellt die wichtigsten Informationen für dieses Projekt bereit. Dabei besteht eine Relation typischerweise aus einer gemischten Folge von Weg-Ids und Knoten-Ids. Anschließend folgen zusätzliche Informationen.

Zur Speicherung verschiedener nicht in jedem OSM Objekt benötigten Informationen wird der “tag“ Eintrag benötigt. Er stellt eine einfache Möglichkeit dar, das genannte Zusatzwissen in Form eines Key-Value Paars zu speichern. Weiterhin werden bei den “relation“ Tags auch Informationen zur Art eines Transportweges mittels Key-Value Paar angegeben.

2.1.3 Transporttypen

Zunächst muss im “relation“ Tag ein Key-Value Paar mit “type”=“route“ vorhanden sein, genauer muss sich `<tag k="type" v="route"/>` innerhalb des “relation“ Tags befinden. Anschließend werden verschiedene Transporttypen unterschieden. Aufbauend auf [OSMF, OSMd] kristallisieren sich folgende wichtige öffentliche Transporttypen (angelehnt an das Openptmap Projekt) heraus:

- ▷ route=train : Eisenbahn
- ▷ route=light_rail oder route=train ref=S : S-Bahn
- ▷ route=subway : U-Bahn
- ▷ route=tram : Straßenbahn
- ▷ route=bus : Bus

Im “relation“ Tag werden auch Namen der jeweiligen Linie gespeichert. Zum Beispiel besteht der “relation“ Tag einer U-Bahn Linie aus Key-Value Paaren für Name, Transporttyp, betreibendes Unternehmen und andere Informationen. Weiterhin werden auch die dazugehörige Schienen (in Form von Weg \$IDs) und Knoten (Haltestellen) gespeichert.

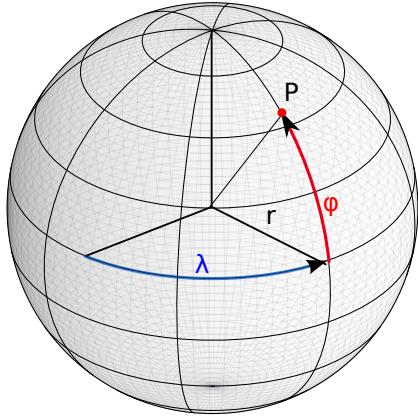
2.1.4 sphärische Koordinaten

Jeder Knoten speichert dabei auch Lageinformationen in Form der GPS-Koordinaten (\$LON,\$LAT).

Typischerweise sind das Longitude ($= \lambda$) und Latitude ($= \phi$) Werte, da in der Kartographie die Koordinaten eines Ortes in diesem Format angegeben werden (vergleiche Abbildung 2.3a). Auf einer Kugel mit dem Radius $r = \text{Erdradius}$ kann mittels der Winkel λ und ϕ , ausgehend von einem Referenzpunkt London $\lambda = 0$, Äquator $\phi = 0$, die Position eines Punktes eindeutig beschrieben werden.

Da sphärische Koordinaten Punkte auf Kugeloberflächen abbilden, aber Karten normalerweise zwei dimensional dargestellt werden (siehe Abbildung 2.3b), wird eine Projektion durchgeführt. Openstreetmap benutzt dazu die bekannte Mercator-Projektion (Zylinderprojektion) [OSMc, Osb08].

$$x = r \cdot \text{arc}(\lambda) \quad y = r \cdot \ln(\tan \phi + \sec \phi) \quad (2.1)$$



(a) Long/Lat Position auf einer Kugel , dargestellt ist ein Punkt $P(\lambda, \phi)$



(b) Projektion der Weltkarte von OpenStreetMap [OSMb]

Abbildung 2.3: Kartendarstellungen

Die sphärischen Koordinaten des Punktes $P(\lambda, \phi)$ können mittels der Formeln 2.1 und dem durchschnittlichen Äquatorradius $r = 6.378.137m$ (siehe [Mor00]) in einen Kartenpunkt $P'(x, y)$ transformiert werden.

Für kleine Kartenausschnitte müsste keine Projektion durchgeführt werden, aber in diesem Projekt sollen auch Entfernungsmessungen zwischen verschiedenen GPS-Positionsknoten durchgeführt werden, durch die Projektion kann die euklidische Distanz zur Messung benutzt werden. Ohne Projektion müssten ständig mittels sphärischer Trigonometrie Distanzen bestimmt werden. Des Weiteren bleiben auch die grafischen Projektionen der Ergebnisse vergleichbar mit den OpenStreetMap Darstellungen.

2.2 Graphen

Die von OSM bereitgestellten Informationen können als mehrschichtiger Graph aufgefasst werden.

2.2.1 Schichten

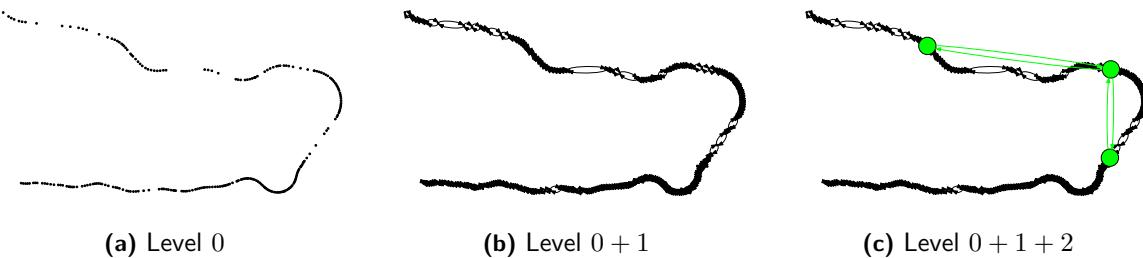


Abbildung 2.4: Logische OSM-Schichten

Dabei stellt die unterste Schicht (Level 0) die reinen GPS-Messpunkte (Node Tag Instanzen) ohne Bezug zueinander dar und ist somit ein Graph bestehend aus positionierten Knoten. Die zweite Schicht (Level 1) sind alle Weg Instanzen und stellt dadurch einen Zusammenhang zwischen den verschiedenen Knoten dar, weiterhin werden auch Kantenklassifizierungen getroffen (Wanderwegkante, Schienenkante, usw.). Level 1 ist daher ein Graph mit klassifizierten Kanten.

Die letzte Schicht (Level 2) stellt Metawissen über die “relation“ Tags bereit, was bedeutet, dass Kanten und Knoten besondere Funktionen erhalten. Mehrere Wege werden in Level 2 zu Transportlinien und besondere Knoten werden zu Halteknoten. Daher kann die Schicht 2 als Graph mit abstrahierten Wissen (klassifizierte und zusammengefasste Knoten/Kanten) aufgefasst werden. Für eine reine Routingaufgabe wäre das Level 1 ausreichend, aber in dem Bewegungsmodell sollen öffentliche Transportmittel betrachtet werden. Nicht jede klassifizierte Schienenkante wird auch für den Transport benutzt (zum Beispiel gibt es häufig Abstellgleise oder unbenutzte Schienen in Level 1), deshalb ist Level 2 von größere Bedeutung und wird im Folgenden als Routinggraph bezeichnet.

Beispielsweise ist in den Abbildungen 2.4a, 2.4b und 2.4c das Eisenbahnnetz von Ilmenau dargestellt. In Level 0 sind nur die Knoten dargestellt, dagegen in Level 0+1 die Schienen und in Level 0+1+2 die Metainformationen (grün) über Haltestellen (Hauptbahnhof, Ilmenau-Pörlitzer Höhe und Ilmenau-Roda) und deren logischem Zusammenhang, insbesondere dass es eine öffentliche Transportlinie vom Hauptbahnhof über alle Zwischenhaltestellen bis nach Ilmenau-Roda gibt. Dabei gibt es keine Kanten in die Außenwelt, da dazu die nächste Haltestation nötig wäre, sie aber nicht in der exportierten OSM-Datei aufgeführt wird.

2.2.2 Routinggraph

Für das Bewegungsmodell ist Level 2 der Schichtinterpretation von bedeutender Rolle, daher soll nun zusammengefasst werden, woraus ein Routinggraph G besteht:

- ▷ V : ausgezeichnete Knotenmenge (Haltestellen)
- ▷ $E \subseteq V^2$: ungerichtete Kantenmenge
- ▷ $pos : V \mapsto (x, y) \in \mathbb{R}^2$: Positionsfunction (projizierte Mercator Koordinaten)
- ▷ $dist : E \mapsto \mathbb{R}^+$: Entfernung zwischen zwei Haltestellen
- ▷ $class : E \mapsto K$: Klassenbestimmung einer Kante ($K = \{ \text{U-Bahn, S-Bahn, usw.} \}$)
- ▷ $line : E \mapsto L$: Transportlinienzugehörigkeit einer Kante ($L = 1, 2, \dots$)

Der Routinggraph wird als ungerichteter Graph modelliert, da in der Regel auf einem öffentlichen Transportweg eine Reise in beide Richtungen möglich ist. Im der OSM-Datei sind auch Richtungsangaben gespeichert, sie werden aber nicht ausgelesen und behandelt.

Weiterhin werden in einem Relationstag Wege und Haltestellen gespeichert. Die Haltestellen werden automatisch zu den Knoten des Routinggraphens. Treffen zwei Haltestellen in einem Relationstag hintereinander, getrennt durch einen Weg, auf, so wird eine Kante in dem Routinggraphen eingefügt. Die tatsächliche Entfernung wird später mittels Dijkstra auf dem darunterliegenden Schienennetz ermittelt. Weiterhin werden die Position der Haltestellen aus den Node Informationen ausgelesen und gespeichert.

Für die jeweiligen verschiedenen Transportlinien gibt es die *line* und *class* Funktionen. Dabei ordnet die *line* Funktion jeder Kante $e \in E$ eine eindeutige Transportliniennummer zu. Die Transportliniennummer ist dabei ohne realen Bezug. Auch die Transportarten (U-Bahn, S-Bahn, ...) werden mittels der *class* Funktion gespeichert, später kann dann im Bewegungsmodell eine Unterscheidung anhand der Transportart stattfinden.

2.2.3 Dot

Zur Darstellung und optischen Validierung der ermittelten Graphen der jeweiligen Großstädte wird das Dot Format benutzt. Dot bietet einfache Möglichkeiten Graphen durch lesbare Formate darzustellen und in verschiedene Grafikformate umzuwandeln. Zum Generieren der jeweiligen PDF Grafiken wurde das “neato”-Plugin des GraphVis Projektes (siehe [Grac, EGK⁺02]) benutzt, dabei ist zu beachten, dass die Knoten des Routinggraphen Positionen besitzen, daher ist ein Ausrichten der Knoten nicht nötig (mittels des Parameter “-n2” kann das Ausrichten deaktiviert werden). Auf die Beschreibung des konkreten Ausgabeformates wird an dieser Stelle verzichtet und auf die Dot-Sprachdefinition des GraphViz Projektes ([Grab]) und deren aufgeführte Beispiele ([Graa]) verwiesen.

2.2.4 Grapheigenschaften/ Metriken

Im späteren Verlauf soll das entwickelte Bewegungsmodell mit dem aktuell benutzten synthetisch generierten Modell verglichen werden. Ein Teilprozess ist dabei die Routinggraphen miteinander zu vergleichen. Eigenschaften von Graphen wurden bereits häufig untersucht. Wichtige Merkmale zum Vergleich verschiedener (Routing-)Graphen sind dabei der Knotengrad, der Clusterkoeffizient und der Durchmesser eines Graphen (vergleiche [AH10][S.193ff]).

Knotengrad (Durchschnitt, Verteilung)

Eine wichtige Eigenschaft eines Graphen ist der Knotengrad $\deg(v)$ eines Knotens. Er gibt an wie viele adjazente Nachbarn ein Knoten hat und kann formal durch

$$\deg(v) = |\{x \in V \mid \{v, x\} \in E\}|$$

für eine ungerichteten Graphen beschrieben werden. Der durchschnittliche Knotengrad ergibt sich dann als Durchschnittswert aller Knotengrade:

$$\deg_{avg} = \frac{1}{|V|} \sum_{v \in V} \deg(v)$$

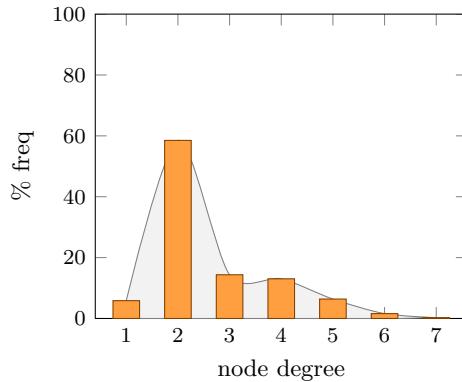


Abbildung 2.5: Beispiel Knotenverteilung

Zum Vergleich verschiedener Graphen ist der \deg_{avg} nicht unbedingt ausreichend, daher soll weiterhin auch die Verteilung der Knotengrade betrachtet werden. Dabei wird für jeden auftretenden Knotengrad die relative Anzahl $H_{deg}(G)$ der Knoten mit dem Grad \deg in einem Histogramm dargestellt (vergleiche Abbildung 2.5).

Besitzen Graphen ähnliche Knotenverteilungen so sind sie auch ähnlich zueinander. Netzwerke weisen dabei verschiedene Knotenverteilungen auf, häufig auftretend sind Power-Law-Verteilungen, oder Poisson/Exponential-Verteilungen (vergleiche [AH10][S.195ff]).

Clusterkoeffizient (lokal, global)

Der Clusterkoeffizient ist eine weitere Möglichkeit verschiedene Graphen zu vergleichen. Dabei wird das Verhältnis der Anzahl von Dreiecken zu verbundenen Tripeln ausgewertet.

Alternativ kann der Clusterkoeffizienten eines Knotens v mittels

$$cc_{local}(v) = \frac{2n(v)}{\deg(v) \cdot (\deg(v) - 1)}$$

berechnet werden, dabei ist $n(v)$ die Anzahl an Kanten die zwischen den Nachbarn des Knotens v tatsächlich verlaufen. Dagegen beschreibt $\frac{\deg(v) \cdot (\deg(v) - 1)}{2}$ die Kantenanzahl in dem Fall, dass alle Nachbarn vollständig mit dem Knoten v und untereinander verbunden wären.

Für den gesamten Graphen folgt dann als globaler Clusterkoeffizient, das arithmetische Mittel aller lokalen Koeffizienten mit

$$cc_{global} = \frac{1}{|V|} \sum_{v \in V} cc_{local}(v)$$

Der Wert des Clusterkoeffizienten liegt im Bereich $[0, 1]$, Werte nahe 0 bedeuten dabei für den Graphen einen geringen Anteil an Cliquen, Werte nahe 1 dagegen einen hohen Anteil, was bedeutet, dass der Graph fast vollständig verbunden ist.

Distanzen/ Durchmesser

Der Durchmesser eines Graphen G ist die größte Entfernung zwischen zwei Knoten des Graphen (vergleiche [HHM][S.18f]) und kann durch

$$diam_G = \max_{u, v \in V} d(u, v)$$

beschrieben werden, wobei $d(u, v)$ die Entfernung des Graphen von u nach v angibt.

Besonders bei Netzwerktopologien ist der Durchmesser eine gute Möglichkeit das Netzwerk zu bewerten, da z.B. die maximale worst case Übertragungszeit abgeschätzt werden kann. Daher bietet sich der Durchmesser auch als Metrik für den Routinggraphen an.

Weiterhin könnte auch die durchschnittliche Entfernung zwischen zwei Knoten in einem Routinggraphen von Bedeutung sein und kann mit

$$d_{avg} = \frac{1}{|V|^2} \sum_{u, v \in V} d(u, v)$$

ermittelt werden.

weitere Metriken

Die aufgeführten Metriken betrachten nur grundlegende Eigenschaften, es gibt weitere interessante Metriken (z.B. Konnektivität, Zentralität) die außerdem betrachtet werden könnten, welche aber für die Bewertung des Routinggraphen und Vergleich zu den synthetisch generierten Topologien nicht unbedingt neue Erkenntnisse bringen. Daher werden sie im Rahmen dieser Arbeit nicht weiter betrachtet.

2.3 Formales Bewegungsmodell

Der bereits beschriebene Routinggraph, der aus den OSM Information generiert wird, dient als Grundlage für das formale Bewegungsmodell. Das Modell besteht aus der Idee, dass sich Menschen mittels öffentlicher Verkehrsmittels in einer Stadt bewegen. Dabei können Menschen wählen, ob sie sich zu Fuß oder z.B. mit der U-Bahn von einem Ort der Stadt zu einem anderen bewegen.

2.3.1 Einschränkungen und Festlegungen

Für das Modell müssen einige Einschränkungen getroffen werden. Zunächst soll als Zug im folgenden alle möglichen Transportmittel einer Transportlinie benannt werden.

Für das Bewegungsmodell soll angenommen werden, dass auf jeder bekannten Transportlinie genau ein Zug zwischen den Endstationen pendelt. Anhand der verschiedenen Transportklassifikationen können verschiedene Geschwindigkeiten für die Züge ermittelt werden.

Eine Bewegung außerhalb der Stadt ist nur bedingt möglich, dabei steigt die Person an der Endstation aus und läuft zu Fuß zu einer nicht ermittelten Transportlinie, die in einer anderen Stadt endet. Diese Festlegung stellt einen Kompromiss dar um die Limitierungen der OSM Daten auszugleichen, denn ein Routing zu einer anderen Stadt würde eine globalere Sicht auf die Karte voraussetzen.

Es wird weiterhin angenommen, dass jeder Haltestation auch zu Fuß erreichbar ist, dabei muss zuvor entschieden werden ob es sich lohnt auf einen Zug zu warten oder nicht.

Im aktuell verwendeten Bewegungsmodell werden Bewegungen zu Fuß betrachtet, daher kann es als Grundlage für das erweiterte Modell benutzt werden und daher muss nur die Funktionalität der Züge hinzugefügt werden.

Jeder Zug besitzt eine gewisse Anzahl an Waggons mit einer bestimmten Ladekapazität. Züge pendeln entlang ihrer Linien von Haltestation zu Haltestation, erreichen sie ihr Ende so bewegen sie sich wieder zum Ausgangsort. Dabei seien die Geschwindigkeiten der Züge als konstant angenommen (es sollen also keine Beschleunigungs- und Abbremsvorgänge betrachtet werden).

An jeder Haltestelle können Personen, sofern noch Platz im Zug ist, einsteigen um ihr potentielles Ziel schneller zu erreichen. Wenn das Ziel einer Person nicht auf der Zuglinie liegt, wartet diese Person oder läuft zum Zielort.

Zur Vereinfachung wird für folgende Situation festgelegt, wenn sich eine Person von *A* nach *B* bewegt, dort eine gewisse Zeit verbringt und anschließend weiter zu *C* reist. So sollen zwei Bewegungsschritte modelliert werden, zunächst bewegt sich die Person von *A* nach *B*, anschließend wird später eine andere Person erzeugt die sich von *B* nach *C* bewegt. Diese Vereinfachung ist nötig, damit nicht alle Personen dauerhaft im der Simulation gespeichert werden müssen.

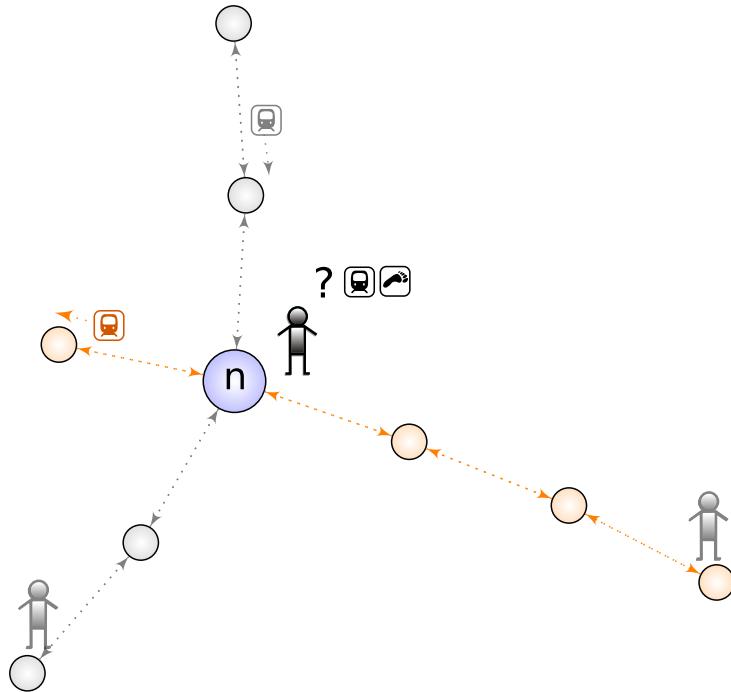


Abbildung 2.6: Haltestelle n mit zwei angedeuteten Transportwegen

In Abbildung 2.6 ist eine typische Situation dargestellt. Eine wartende Person an der Haltestelle n , die sich entscheiden muss ob sie mit dem Zug oder zu Fuß ihr Ziel erreichen will, außerdem ist ein Zug (orange) bereits an der Haltestelle n vorbei und ein weiterer (grau) trifft bald ein. Weiterhin gibt es an allen Haltestellen mehrere Personen, die die gleichen Entscheidungen in der beschriebenen Art und Weise treffen müssen. Die Person muss also die Entscheidung treffen ob es sich lohnt auf den Zug zu warten oder nicht, diese Entscheidung kann einfach durch die Berechnung der Wartezeit und Transportzeit gefällt werden. Falls die Person zu Fuß ihr Ziel schneller erreicht ist es unnötig auf den Zug zu warten, es kann aber dennoch passieren, dass der Zug bereits überfüllt ist, dann muss sie erneut entscheiden ob sie auf den nächsten Zug wartet oder zu Fuß ihr Ziel erreicht.

Zusammenfassend sind für das formale Bewegungsmodell folgende Eigenschaften und Festlegungen nötig:

- ▷ festgelegte Waggonkapazität, Anzahl Waggons und Kapazitätstoleranz (im realen Leben kommen überfüllte öffentliche Transportmittel häufig vor)
- ▷ es können beliebig viele Züge an einer Haltestelle sein, da spezielle Hauptbahnhöfe oder Stationen mit mehreren Gleisen nicht ermittelt wurden
- ▷ Geschwindigkeiten und daraus berechnete Ankunftszeiten der Zugtypen (anhand der Kantenklassifikation und Distanzen)
- ▷ Modellierung des Zugpendelns
- ▷ akzeptable Wartezeit für das Warten auf einen Zug
- ▷ Bewegungen ohne Zwischenauftenthalte und Umsteigevorgänge
- ▷ es gibt an jeder Haltestelle die Möglichkeit potentielle Züge die zum Zielort führen und Ankunftszeiten zu ermitteln (z.B. Ticketautomat)

2.3.2 Modell

Für das formale Modell müssen zunächst einstellbare Parameter festgelegt werden:

- ▷ $wagonCap$: Waggon-Kapazität: 260 Personen pro Waggon (vergleiche: [\[wika\]](#))
- ▷ $wagonCountPerTrain$: Anzahl an Waggons pro Zug,
für jeden Zug zufällig gleich verteilt mit $\sim R[1, maxWagonCountPerTrain]$
- ▷ $trainCapTolerance$: Zugtoleranz um überfüllte Züge zu simulieren,
festgelegt als $\sim R[0, wagonCap]$ explizit für jeden Zug
- ▷ $v(class)$: Geschwindigkeit für jede Zugklasse (z.B. $60 \frac{km}{h}$ für eine U-Bahn, einstellbar)

Weiterhin sind folgende Informationen nötig:

- ▷ R : extrahierter Routinggraph
- ▷ $D[x, y, l]$: Distanzmatrix (erstellt aus dem Routinggraphen), um die Transportzeiten für eine Linie l zu berechnen
- ▷ $L[x, y]$: Linienmatrix, um zu ermitteln welche Linie eine Person am Ort x zu y befördern kann.
- ▷ $P[x, y, l]$: Pfadmatrix, um zu einer Linie l und den Orten x, y einen nächsten Knoten entlang des kürzesten Pfades zu ermitteln

Dabei können alle Matrizen basierend auf einem modifizierten Dijkstra vorher aus dem Routinggraphen ermittelt werden (Dijkstra mit Einschränkung der Knotenwahl, denn es sollen nur Knoten der jeweiligen Zuglinie betrachtet werden).

Initialzustand Alle Züge sind gleich verteilt auf den Knoten, die zu ihrer Transportlinie gehören und wählen eine zufällige Startrichtung. Ihre Waggonkapazität in Form von $cap(x)$ für einen Zug x wurde anhand der Parameter bestimmt:

$$cap(x) = wagonCap \cdot wagonCountPerTrain(x) + trainCapTolerance(x)$$

Es werden außerdem auch die Geschwindigkeiten der jeweiligen Züge festgelegt. Alle benötigen Informationen wurden aus dem Routinggraphen ermittelt.

Simulationsverhalten. Die Züge Pendeln entlang ihrer Zuglinie mit ihrer aktuellen Richtung, wird eine Endstation erreicht, so wird die Richtung gewechselt. Trifft ein Zug auf eine Haltestation ein, steigen zuerst die Personen die am Zielort sind aus, außerdem können Personen, deren Ziel auf der Zuglinie liegt, einsteigen sofern der Zug nicht voll ist und wenn es schneller ist mit dem Zug zu fahren. Die Entscheidung ob ein Ziel auf der Zuglinie liegt kann durch die Linienmatrix und Pfadmatrix einfach ermittelt werden. Die Zugfahrtzeiten können mit der Distanzmatrix und der Geschwindigkeit ermittelt werden.

Wertung. Da das beschriebene Bewegungsmodell, das vorhandene Modell um die Möglichkeiten der Benutzung öffentlicher Transportmittel erweitert, ist eine explizite Evaluierung nicht nötig. Denn die ermittelte geschätzte Topologie im Simulationsframework kann durch diese Erweiterungen nur besser werden.

2.4 Simulationsframework

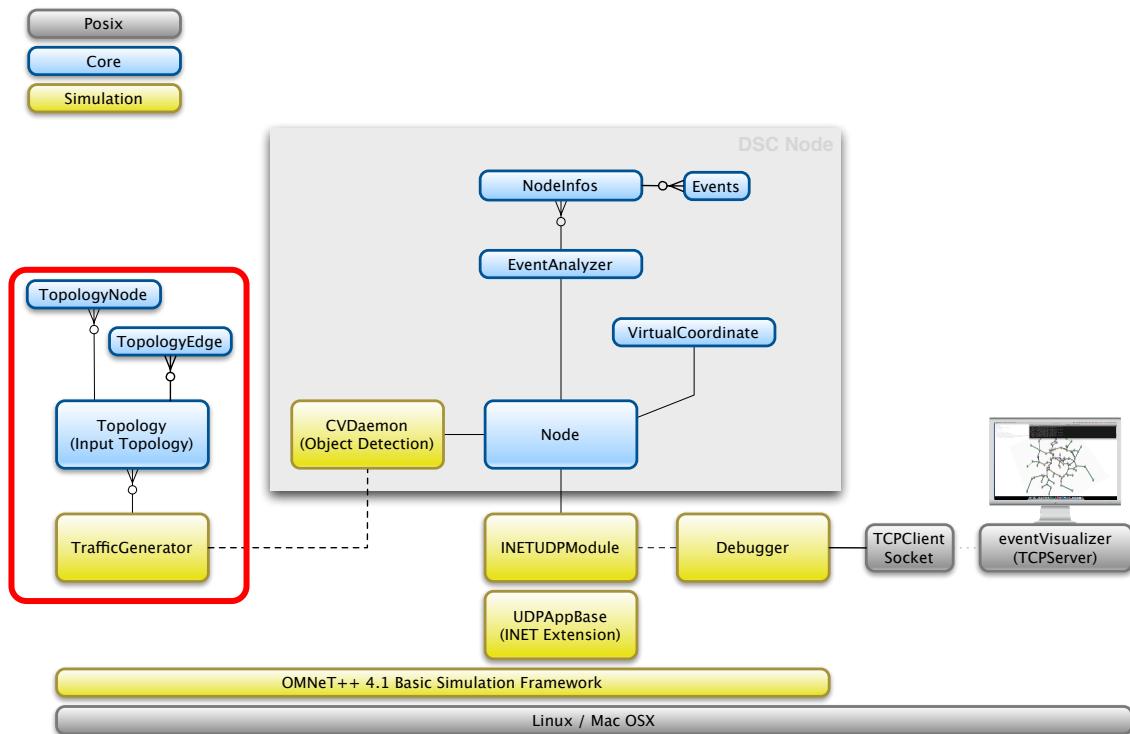


Abbildung 2.7: Klassenstruktur des Simulationsframeworks, Erweiterungsstellen sind rot markiert

In Abbildung 2.7 ist das Framework für die simulative Untersuchung von verteilten Smart Camera Systemen dargestellt. Für die Simulationsanwendung wird das Omnet++ Framework und die Inet Erweiterung benutzt. Das beschriebene Bewegungsmodell soll in den Simulationsteil integriert werden. Es besteht aus mehreren Komponenten. Zunächst die Komponente "DSC-Node", welche alle Funktionalitäten für die Topologieschätzung und Kommunikation der Kameraknoten bereitstellt. Weiterhin sind für die Simulation die rot markierten Teilmodule wichtig. Sie stellen die Grundlage für Bewegungsmuster dar. Im späteren realen Einsatz wird die Komponente durch den "CVDaemon", welcher Bewegungen von Menschen/Autos/.. erkennt, realisiert. Aber für die Simulation ist zunächst ein Bewegungsmodell erforderlich. Daher muss das hier beschriebene Modell durch Erweiterungen des "TrafficGenerators" verwirklicht werden. Der "TrafficGenerator" benutzt dazu eine "InputTopology" bestehend aus Kanten und Knoten. Außerdem werden bisher rein zufällige Bewegungen erzeugt. Aufbauend auf OSM kann direkt die "InputTopology" (Routinggraph) erzeugt werden. Für die Transportwege müssen Erweiterungen im "TrafficGenerator" durchgeführt werden, so dass auch öffentliche Transportmittel bei der Bewegung berücksichtigt werden, dazu werden die Informationen des formalen Bewegungsmodells umgesetzt.

Zusammenfassend muss das bestehende Framework an den rot markierten Stellen erweitert werden. Optional wäre auch eine Erweiterung in der Visualisierungskomponente (EventVisualisierer) möglich, um die neuen Bewegungsabläufe auch während der Simulation verfolgen zu können.

3 Implementierung/ Integration

Die Implementierung des in Abschnitt 2.3 beschriebenen Bewegungsmodells kann in zwei Schritte eingeteilt werden. Im ersten Schritt werden die nötigen Informationen aus den OSM Daten extrahiert (Aufbau des Routinggraphs). Der erste Schritt wird komplett durch den OSMConverter realisiert und die Ergebnisse in ein einfaches Zwischenformat überführt. Im zweiten Schritt werden die exportierten Informationen in die Simulation integriert, dazu müssen sie ausgelesen werden und entsprechende Erweiterungen im Simulationsframework getroffen werden.

3.1 OSMConverter

Der OSMConverter ist ein Tool zum Extrahieren des Routinggraphens aus einer OSM-Datei. Die Implementierung erfolgte in Python 3.

3.1.1 Aufbau

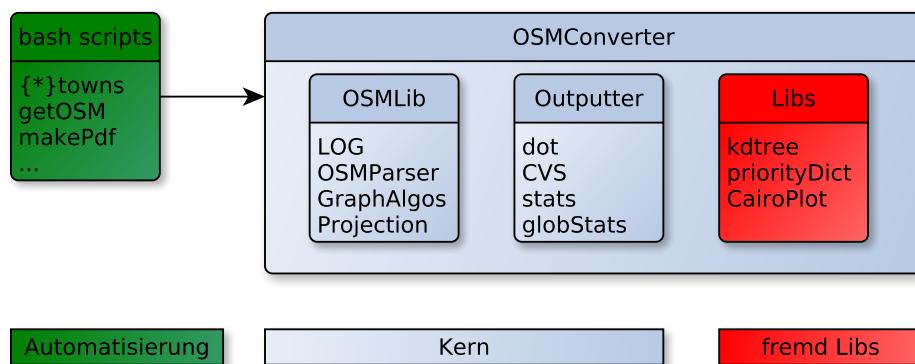


Abbildung 3.1: Komponenten des OSMConverters

In Abbildung 3.4 sind die Komponenten des OSMConverters dargestellt.

Der OSMConverter besteht aus den Modulen OSMLib, Outputter und Libs. Wobei das Libs Modul alle Fremdbibliotheken zusammenfasst. Die Fremdbibliotheken stellen Funktionen zur Erzeugung von Grafiken (CairoPlot), zum Suchen von benachbarten Punkten im zweidimensionalen Raum (kdtree) und effiziente Datenstrukturen (prioDic) für Dijkstra bereit.

Das Outputtermodul bietet die Möglichkeit verschiedene Ausgabeplugins zu entwickeln. Sie können dynamisch hinzugefügt werden und stehen anschließend als gültiges Ausgabemodul im OSMConverter zur Verfügung. Die wichtigsten Ausgabemodule sind dabei das dot Modul zur späteren Erzeugung der PDF Grafiken und das stats bzw. globStats Modul, welche die aufgeführten Grapheigenschaften und Metriken berechnen, abspeichern und zusätzlich Grafiken erzeugen. Weiterhin ist für die Simulation

das CSV Ausgabemodul erforderlich, da es den ermittelten Routinggraphen für die Weiterverarbeitung in der Simulation speichert.

Das Modul OSMLib stellt alle benötigten Graph Algorithmen (Graph Datenstruktur, Dijkstra, Breitensuche, Zusammenhangskomponenten,...), die Mercator-Projektion, Debug Logging Ausgaben und die OSMParser-Klassen bereit.

Die OSMParser-Klassen sind spezielle SAX-XML Parser Klassen und realisieren das Auslesen der jeweiligen XML Informationen einer OSM Datei. SAX-XML bietet dabei eine effiziente Methode große XML-Dateien zu verarbeiten, dabei wird ereignisbasiert die XML-Datei sequentiell durchlaufen. Beim Start und eines Ende XML-Tags wird ein Ereignis ausgelöst (*startElement* und *endElement*), damit man die gelesenen Informationen verarbeiten kann, als Basisklasse dient dabei *sax.handler.ContentHandler*.

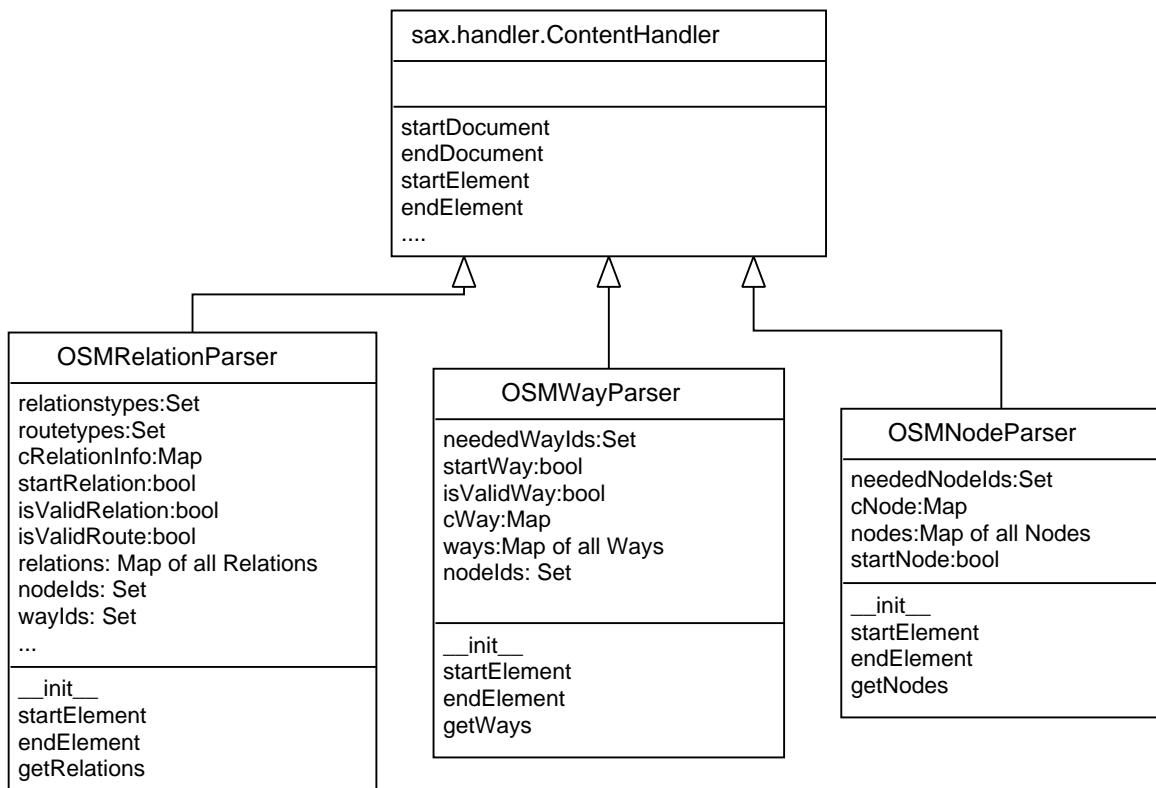


Abbildung 3.2: OSMParser Klassendiagramm

Der OSMConverter benutzt drei solcher Parserklassen (vergleiche Abbildung 3.2):

- ▷ `OSMRelationParser`: Auslesen der Relationsinformationen
- ▷ `OSMWayParser`: Ermitteln des Wegaufbaus
- ▷ `OSMNodeParser`: Knoteninformationen extrahieren

Der Parsingvorgang muss auch in der Reihenfolge Relationen, Wege und Knoten erfolgen, um möglichst wenig Speicher zu verschwenden (leider muss dagegen genau drei mal die OSM-Datei durchlaufen werden, was eine höhere Laufzeit verursacht). Alle drei Parserklassen sind im Verhalten ähnlich, daher soll nur der `OSMRelationParser` genauer betrachtet werden.

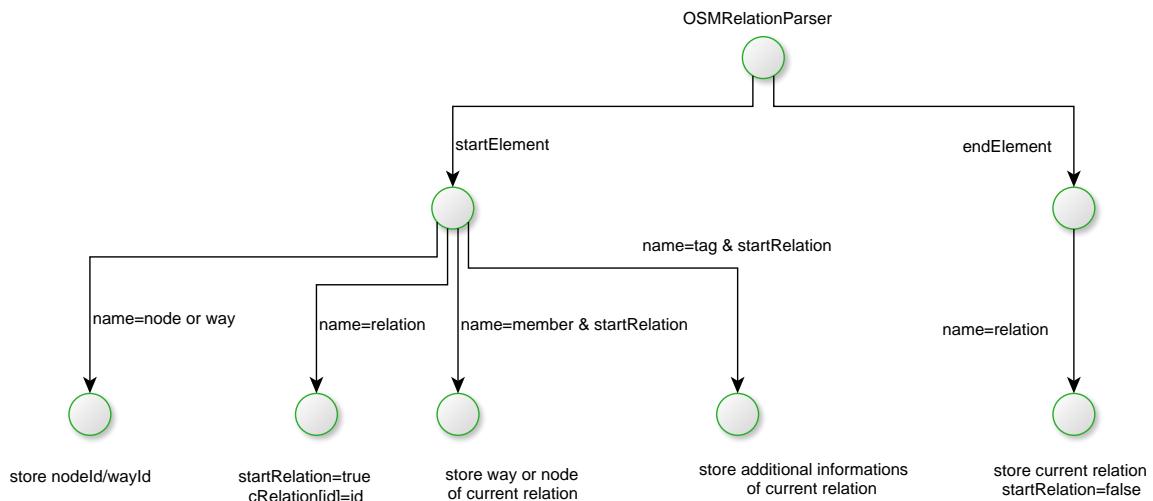


Abbildung 3.3: OSMRelationParser Ablauf

Abbildung 3.3 zeigt den schematischen Ablauf des OSMRelationParser. Wird beim Lesen der XML-Datei ein XML-Start-Tag gefunden so wird die Methode `startElement` aufgerufen, anschließend kann überprüft werden, um welches Start-Tag es sich handelt. Falls es sich um ein Node oder Way Tag handelt, so wird zur Optimierung die Knoten/Weg ID gespeichert. Interessant ist aber der Fall in dem ein Relation-Tag beginnt, denn anschließend wird der Zustand “`startRelation=true`” eingeleitet. Innerhalb eines Relation-Tags befinden sich verschiedenen Member-Tags (Knoten oder Wege). Falls nun ein Member-Start-Tag beginnt und eine Relation begonnen wurde, so wird der jeweilige Knoten bzw. Weg der aktuellen Relation zwischengespeichert (in der Map “`cRelation`”). Analog werden auch weitere Informationen der Relation gespeichert, die mittels Key-Value über das XML-Element “`Tag`” zur Verfügung stehen. Ist die aktuelle Relation nun vollständig verarbeitet, so folgt ein End-Relation-Tag und die Methode `endElement` wird aufgerufen. Anschließend werden die gesammelten Informationen der aktuellen Relation in einer Map abgelegt, um sie später weiterzuverarbeiten und “`startRelation`” wird auf “`false`” gesetzt. Weiterhin werden natürlich Relationen die nicht den gewünschten Typ besitzen beim Parsen direkt übersprungen. Das Speichern der Weg- und Knotenids ist notwendig, damit an geeigneter Stelle auch überprüft werden kann, ob eine Knoten/Weg-Id einer Relation auch in der aktuellen OSM-Datei gespeichert ist, falls dies nicht der Fall ist, so kann der Knoten/Weg ignoriert werden (da sowieso keinerlei Informationen in der OSM-Datei über den Knoten/Weg vorhanden wären).

Zur Automatisierung der Untersuchung von verschiedenen Städten besteht das Toolkit weiterhin aus einer Reihe von Bash Skripten, erwähnenswert ist hier zum Beispiel das “`bigtowns.sh`“ Script, welches automatisch alle gepackten Städte des Metro-Projektes entpackt, mit dem OSMConverter die nötigen Informationen ermittelt und anschließend verschiedene Grafiken und Statistiken erzeugt. Weiterhin gibt es auch ein Bashscript, welches alle Städte des Metro-Projektes automatisch herunterlädt.

3.1.2 Ablauf

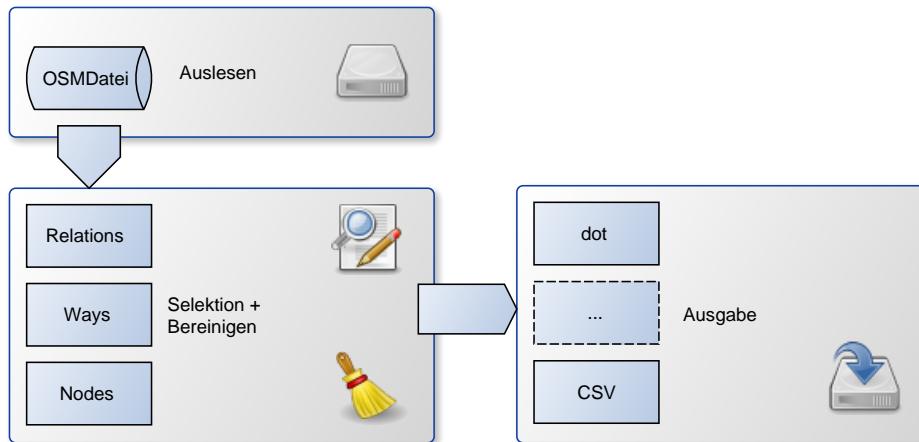


Abbildung 3.4: Ablauf des OSMConverters

Der Ablauf des OSMConverters kann in drei Schritte eingeteilt werden (vergleiche Abbildung 3.4). Im ersten Schritt wird die benötigte OSM-Datei drei mal durch die OSMParse-Klassen verarbeitet. Das mehrmalige Parsen der OSM-Datei ist nötig, da die Relationsinformationen am Ende stehen und man aber zum Zeitpunkt des Relationsparsens nicht alle vorherigen Informationen im Ram haben sollte. Es können daher auch größere OSM-Daten verarbeitet werden. Beispielsweise ist die *Berlin.osm* knapp 1GB groß und beim Parsen werden aktuell ca. 2-3 GB an Ram verbraucht und bei Berlin ergibt sich eine Bearbeitungszeit (mit allen Ausgabemodulen) von ca. 3-4 Minuten. Eine C++ Implementierung könnte durchaus effizienter arbeiten, aber Python 3 bietet viele vorgefertigte Datenstrukturen und kompakten funktionalen Quellcode.

Im Parsingvorgang werden zunächst alle benötigten Relationen ausgelesen (anhand der Relationstypen und Straßentypen). Während des Auslesens der Relationen, werden alle nötigen Weg und Knoten-IDs gespeichert.

Im zweiten Parsing Schritt, werden alle notwendigen Weg-Informationen extrahiert, dabei erfolgt ein Sammeln der benötigten Knoten-IDs (zusätzlich zu den bereits im Relationparsingvorgang ermittelten).

Am Ende des Parsing Vorgangs werden alle Knoteninformationen ermittelt und in speziellen Datenstrukturen (Python 3 Hash Maps) gespeichert. Anschließend ist die erste Phase beendet und es stehen nun alle selektierten Informationen bereit (Knoten, Wege und Relationen).

Im zweiten Schritt wird anhand der gesammelten Informationen ein Graph aufgebaut, der dann bereinigt wird. Die Bereinigung beinhaltet hauptsächlich das Zusammenfassen dicht benachbarter Haltestellen (dicht heißt dabei, dass sich die Knoten innerhalb eines Radius von 200m befinden) mittels KD-Baum. Das Knotenkontrahieren ist nötig um am Ende einen zusammenhängenden Graphen zu erhalten, auch wenn verschiedene Transportwege betrachtet werden. Zum Beispiel befindet sich meist in der Nähe eines Hauptbahnhofs auch eine Straßenbahnhaltestelle. In der Bereinigungsphase wird zusätzlich auch die größte Zusammenhangskomponente ermittelt, die projizierten Koordinaten und alle Entferungen zwischen den Haltestellen berechnet und im Graphen abgespeichert.

Die letzte Phase besteht darin, den ermittelten Routinggraph in verschiedene Export- und Ausgabeformate mittels der Outputer-Module umzuwandeln, dabei sind die wichtigsten Ausgabemodule das dot, stats und CSV Modul.

3.1.3 Ausgabeformate

Der nun ermittelte Routinggraph wird durch mehrere erweiterbare Ausgabemodule in verschiedene Formate konvertiert. Die Module werden beim Start des OSMConverters dynamisch aus dem Verzeichnis „outputter“ gelesen. Ein Ausgabemodul muss dabei lediglich eine Funktion:

```
def output(OSMFilename, completeGraph, routingGraph, infos={}):
    ...
```

implementieren, dabei werden die Parameter *OSMFilename*, *completeGraph*, *routingGraph*, *infos* übergeben. *OSMFilename* ist der Dateiname der aktuell zu bearbeitenden OSM-Datei, *completeGraph* speichert den gesamten Level 0 Graphen, *routingGraph* speichert den ermittelten Routinggraphen (Level 2) und *infos* stellt einige zusätzliche Informationen bereit (z.B. max/min Koordinaten).

Stats/ Dot

Das stats-Modul dient zu Berechnung der in Abschnitt 2.2.4 beschriebenen Metriken, die berechneten Informationen werden dabei lesbar in einer Textdatei *.stats* abgespeichert.

Beispielauszug für Berlin:

```
# infos:
    relationsnames : 37
    relations : 37
    ...
# scale faktor (1 ^= scale meter): 93938.7556817
relationsnames:
    S-Bahnlinie S9: S Pankow => S Flughafen Schoenefeld
    ...
```

Das Dot-Modul generiert eine grafische Ausgabe des Routinggraphens in Form von Dot-Language Anweisungen und dient später zum Erzeugen der in diesem Dokument verwendeten Graphiken.

CSV

Für die Weiterverarbeitung des Routinggraphens in der Simulation ist das CSV-Modul von entscheidender Rolle. Zunächst werden zwei Textdateien *.nodes* und *.edges* angelegt. Die *.nodes*- Datei beinhaltet alle Knoteninformationen, d.h. die Knoten-ID, die x-Position und die y-Position.

Dagegen speichert die *.edges*-Datei Kanteninformationen des Routinggraphens. Genauer werden dort Startknoten, Endknoten, Entfernung, Transportlinien-ID und Transportart einer Routingkante abgelegt.

Zum Beispiel ergibt sich für Berlin auszugsweise folgende *.nodes/.edges* Dateien:

```
berlin.nodes:
#nodeID posX                  posY
0      0.30092152049696697    0.4079085225477145
1      0.3995473749274487    0.4419561093463904
...
berlin.edges:
#src      dest      dist          line      class
0        290      987.126169235446  0         light_rail
0        69       2494.5386928370094  1         light_rail
...
```

3.2 Integration in die Simulation

Um den ermittelten Routinggraphen in der Simulation zu verwenden mussten Erweiterungen am TrafficGenerator durchgeführt werden. Zunächst soll die Struktur des unveränderten TrafficGenerators beschrieben werden (nur die für die Erweiterungen wichtigen Stellen) um danach die Erweiterungen darzustellen. Im Anschluss werden die Erweiterungen für das neue Bewegungsmodell beschrieben.

3.2.1 TrafficGenerator

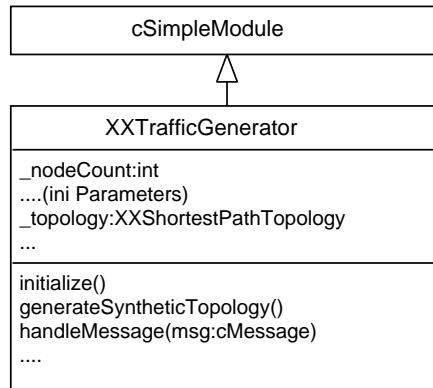


Abbildung 3.5: Klassendiagramm TrafficGenerator

Die Abbildung 3.5 zeigt die Klassenstruktur des TrafficGenerators. Als Basisklasse wurde die OmNet++ SimpleModule-Klasse festgelegt. Zunächst soll der typische Ablauf des TrafficGenerators beschrieben werden. Über Ini-Parameter der *omnetpp.ini* Datei wird die Topologieart, Knotenzahl und weitere Parameter festgelegt. OmNet++ kümmert sich dann um das Erzeugen einer Instanz des TrafficGenerators, dabei wird die Methode *initialize()* aufgerufen, welche alle internen Objektvariablen und Verhalten festlegt. Dazu zählt zum Beispiel auch die Knotenzahl *_nodeCount*. Es wird beim initialisieren auch eine Auswahl zwischen der gewählten Topologieart getroffen. Wurde zum Beispiel die Topologie "syntheticTopologie" ausgewählt, so ruft *initialize()* die Methode *generateSyntheticTopology()* auf. Dabei wird eine synthetische Topologie erzeugt (Knoten und Kanten) und in der Objektvariable *_topology* gespeichert.

Um nun eine andere Topologie zu erzeugen mussten mehrere Erweiterungen vorgenommen werden. Zunächst wurde eine neue Klassenmethode *importOSMTopology()* angelegt. Die Initialisierungs methode wurde um einen neuen Fall "OsmTopologie" erweitert, welche dann anschließend die angelegte *importOSMTopology()* Funktion aufruft und die gewählte OSM-Topologie lädt. Da das beschriebene Dateiformat recht einfach ist, sei an dieser Stelle auf eine genaue Beschreibung des Ladens verzichtet. Es sei nur angemerkt, dass auch in der Topologie Klasse zum Speichern der zusätzlichen Informationen, wie Liniennummer, Transportklasse Erweiterungen getroffen werden mussten (zusätzliche Map Datenstrukturen).

Damit verschiedene Städte geladen werden können musste auch die *trafficGenerator.ned* Datei erweitert werden, dazu wurde ein neuer Parameter namens *osmfile* ergänzt, welcher den Basisnamen der Eingabetopologie darstellt (für die exportierten Dateien *osm/berlin.nodes*, *osm/berlin.edges* zum Beispiel "osm/berlin").

Weiterhin stellte auch die Ermittlung der Knotenzahl eine Erweiterung dar, vorher musste die Knotenzahl immer als Ini-Parameter angegeben werden. Da es recht unpraktisch ist, für jede

Stadt explizit eine eigene *omnetpp.ini* Datei anzulegen, welche die Knotenanzahl der Importtopologie fest eingetragen hat, wurde eine neue "NED_Function" erzeugt, welche für eine übergebene OSM-Topologie die Knoten zählt und wie zum Beispiel die *exponential(x)*, *normal(x)* Funktionen in der *omnetpp.ini* verwendet werden kann.

3.2.2 Erweiterungen im Bewegungsmodell

Zunächst mussten alle Parameter des formal beschriebenen Bewegungsmodells in der Datei *trafficGenerator.ned* ergänzt werden. Zusätzlich dazu mussten für die Bewegung der Züge neue Nachrichten mit den Namen *trainMessage* hinzugefügt werden. Zu Beginn der Simulation wird nachdem die OSM-Topologie ausgelesen wurde für jede dort aufgelistete Zuglinie ein *trainMessage* Nachricht angelegt. Dabei werden in der Nachricht die Zuglinie, Zugklasse, der momentane, vorhergehende Knoten, Personenkapazität und Personen (als Nachrichten huckepack), Linienstartknoten/Endknoten gespeichert. Der TrafficGenerator kümmert sich um die Behandlung dieser speziellen Nachricht ergänzend zu den normalen Bewegungsnachrichten des alten Bewegungsmodells. Dabei werden die *trainMessages* am Anfang mit einem zufälligen Startknoten (welcher auf ihrer eigenen Zuglinie liegt) initialisiert. Die Bewegung erfolgt, dann getaktet. Für jeden Zug x kann der nächste Halteknoten (basierend auf dem aktuellen Knoten und dem Knoten davor) und eine Ankunftszeit, nach folgender Berechnung:

$$t_{next_arrival}(x) = simtime() + D[curr(x), next(x), line(x)] \cdot v(class(x))$$

ermittelt werden, dabei ist $D[curr(x), next(x), line(x)]$ die physikalische Entfernung entlang der Zuglinie und $v(class(x))$ die Geschwindigkeit der Zugklasse. Die *trainMessage* wird dann nach ihrer nächsten Ankunftszeit geplant.

Trifft im TrafficGenerator eine *trainMessage* ein, also ein Zug x , so müssen mehrere Entscheidungen getroffen werden. Sind aktuell Personen im momentanen Knoten des Zuges erzeugt wurden, die mittels des aktuellen Zuges (und dessen Zuglinie) ihren Zielort schneller erreichen? Für eine Person p bedeutet schneller dabei eine einfache Entscheidung. Wenn der Zug nicht voll ist:

$$size(x) + 1 \leq cap(x)$$

und der Zielort schneller als zu Fuß:

$$D[start(p), target(p), line(x)] \cdot v(class(x)) < D_{geo}[start(p), target(p)] \cdot v_p$$

erreicht werden kann, dann wird natürlich der Zug benutzt (v_p ist die im ursprünglichen Modell festgelegte Geschwindigkeit einer Person, ca. $5 \frac{km}{h}$, $size(x)$ gibt die Anzahl der sich aktuell im Zug befindenden Personen an und D_{geo} ist die geographische Distanz zwischen zwei Knoten).

Weiterhin wird für alle im Zug gespeicherten (verpackten) Personen überprüft, ob sie ihren Zielort erreicht haben. Tritt dieser Fall ein, so wird die Person aus dem Zug entfernt. Und kann anschließend die neu gewonne Freizeit sinnvoll einsetzen ;).

Da sich die *trainMessage* Objekte während der gesamten Simulation im System befinden, müssen Vorfahrten getroffen werden, damit diese Züge entlang ihrer Linien pendeln. Es kann vorkommen, dass ein Zug am Ende der Zuglinie ankommt, in diesem Fall bewegt sich der Zug einfach in umkehrter Richtung zurück. Im Fall das die Zugstrecke zirkulär ist, wird einfach nie ein Ende erreicht und der Zug kreist die gesamte Simulationszeit entlang der Zuglinie.

4 Auswertung

Es wurde das U-Bahn und S-Bahn Netz verschiedener Städte untersucht. Dabei wurden die Metriken (außer Distanzen/ Durchmesser), welche in Abschnitt 2.2.4 eingeführt wurden berechnet. Auf die Metriken für die Distanz und dem Durchmesser wurde verzichtet, da sie die physikalischen Entfernungen mit einbeziehen, aber die untersuchten Städte verschiedene Areale aufweisen und diese Metriken dann schwer zu vergleichen wären. Anschließend wurde exemplarisch für zwei Großstädte synthetisch generierte Topologien erzeugt und Vergleiche zwischen den Topologien durchgeführt.

4.1 Großstädte vs. generierte Topologien

Im folgenden Abschnitt sollen zwei Großstädte betrachtet werden und die mittels OSM extrahierten Topologien mit synthetischen verglichen werden. Für die ermittelten synthetischen Topologien sind immer Mittelwerte über 32 generierte Topologien mit fester Knotenanzahl dargestellt.

4.1.1 Berlin

Die Abbildung 4.1 stellt das ermittelte Netz von Berlin dar, gut erkennbar ist die Überdeckung des gesamten Schienennetzes (Level 0) durch den ermittelten Routinggraphen (Level 2).

Für den dargestellten Graphen von Berlin wurden folgende Werte ermittelt:

deg_{avg}	cc_{global}	$ E $	deg_{max}	deg_{min}	$ V $
2.61	0.14	983	7	1	376

Weiterhin zeigt Abbildung 4.2 die ermittelte Knotengradverteilung für Berlin.

Auswertung. Zunächst ist deutlich in der Knotengradverteilung und anhand des deg_{avg} erkennbar, dass am häufigsten mit 60% der Knotengrad 2 vertreten ist. Für eine Großstadt ist diese Eigenschaft auch gut zutreffend, da am häufigsten kleinere Haltestationen (mit Ein- und Aussteigemöglichkeiten) vorhanden sind.

Weiterhin ist der größte vorkommende Knotengrad 7, was durch einen zentralen Haupt/ U-Bahn/ Straßenbahnhof gut erklärbar ist.

Auch der kleinste Knotengrad 1 ist schlüssig, da die OSM-Daten nur Kartenausschnitte beinhalten, müssen Verbindungslien zu anderen Orten irgendwann enden. Der ermittelte globale Clusterkoeffizient fällt gering aus, da ein Transportnetz wenig Redundanz aufweisen sollte.

Optisch könnte die Knotengradverteilung von Berlin am ehesten als Normal-/Poissonverteilt mit dem Mittelwert $\lambda = deg_{avg}$ approximiert werden (vergleiche 4.3).



Abbildung 4.1: Berlin

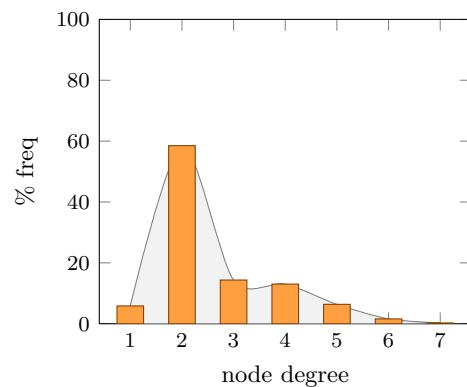


Abbildung 4.2: Knotengradverteilung: Berlin

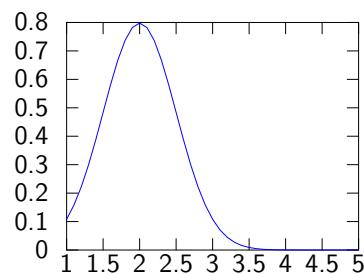


Abbildung 4.3: Normalverteilung $\lambda = 2, \sigma = 0.5$

4.1.2 synthetische Vergleichstopologie- Berlin

Zum Vergleichen der Topologie von Berlin wurden synthetisch generierte Topologien mit 376 Knoten erzeugt. Es wurden die genannten Metriken als Durchschnittswerte über 32 erzeugte Topologien ermittelt und ergaben:

$\overline{deg_{avg}}$	$\overline{cc_{global}}$	$\overline{ E }$	$\overline{deg_{max}}$	$\overline{deg_{min}}$	$ V $
2.36	0.03	890	4.5	1	376

Auffallend ist die hohe Ähnlichkeit der ermittelten synthetischen Topologien mit Messwerten von Berlin. Die durchschnittlichen Knotengrade sind sehr dicht beieinander, außerdem ist auch der Clusterkoeffizient sehr gering. Auch die Kantenanzahl ist vergleichbar groß. Nur der größte vorkommende Knotengrad fällt etwas kleiner aus.

Die Abbildung 4.4 zeigt weiterhin die ermittelte durchschnittliche Knotengradverteilung der 32 Ersatztopologien für Berlin. Auch hier kann festgestellt werden, dass die Verteilung recht ähnlich zur

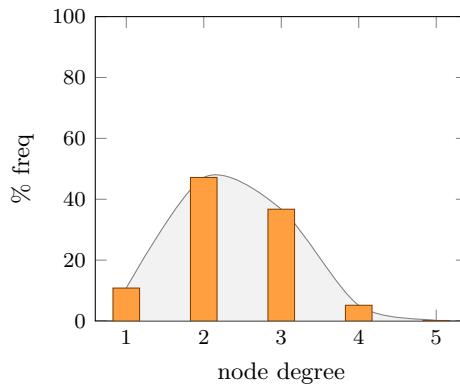


Abbildung 4.4: durchschnittliche Knotengradverteilung der synthetischen Topologie für Berlin

Knotengradverteilung von Berlin ist. Auffällig ist das Hoch bei Knotengrad 2 – 3 und der Abfall zu den benachbarten Knotengraden. Selbst die Häufigkeit des Knotengrades 2 – 3 mit ca. 50% ist vergleichbar mit der Häufigkeit bei Berlin mit ca. 60%.

4.1.3 Moskau

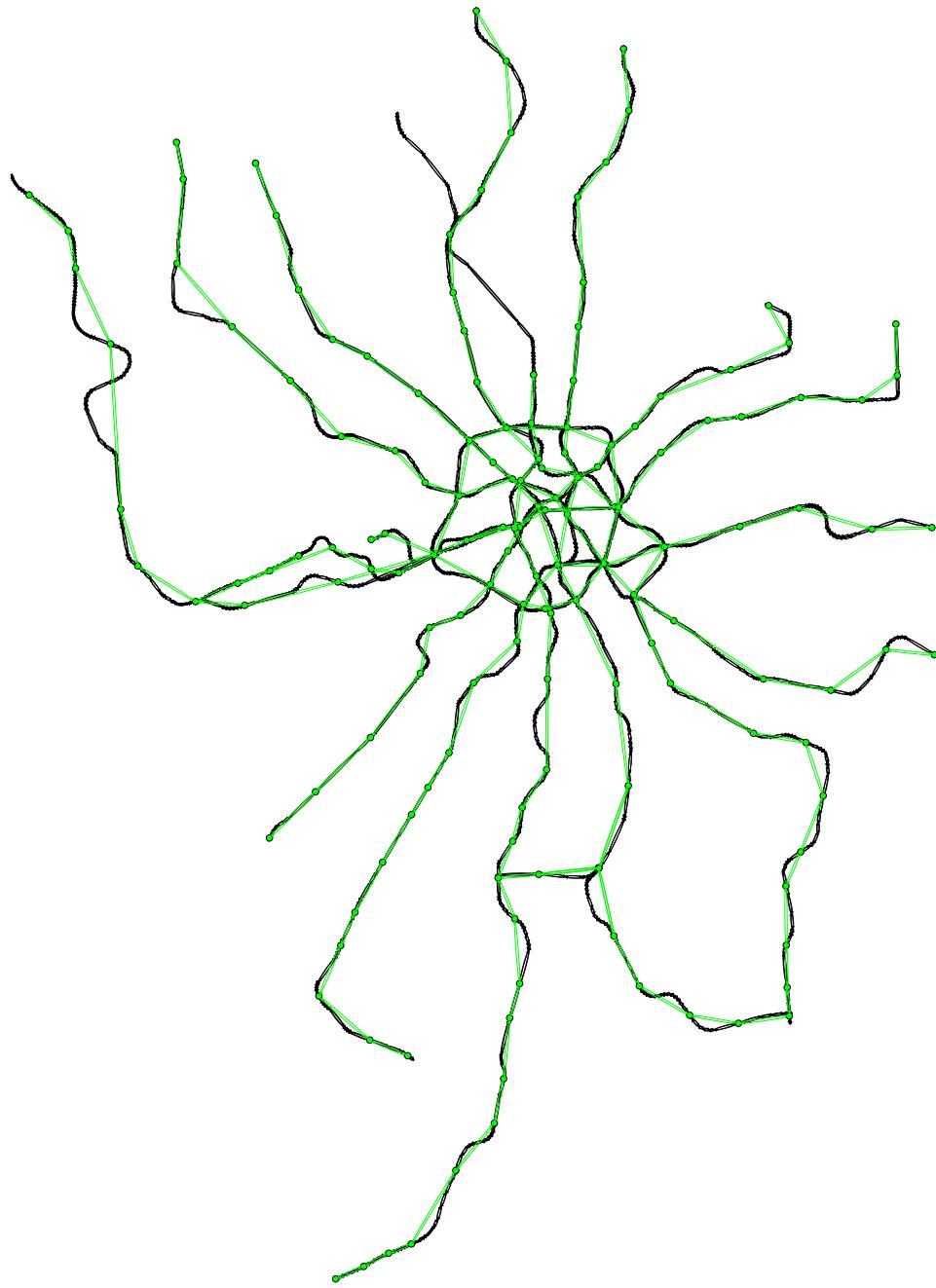


Abbildung 4.5: Moskau

Abbildung 4.5 zeigt die ermittelte Topologie für Moskau. Genau wie bei Berlin ist deutlich die Überdeckung des Schienennetzes erkennbar.

Für den dargestellten Graph ergaben sich folgende Werte:

deg_{avg}	cc_{global}	$ E $	deg_{max}	deg_{min}	$ V $
2.18	0.01	352	4	1	161

Auch diese Werte sind ähnlich zu den Werten von Berlin. In Diagramm 4.6 ist die Knotengradverteilung von Moskau dargestellt.

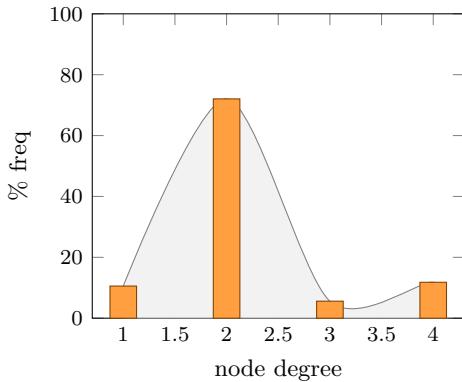


Abbildung 4.6: Knotengradverteilung: Moskau

Auswertung. Obwohl das Netz von Moskau nur aus ca. 1/2 so vielen Knoten wie Berlin besteht, weißt es doch eine große Ähnlichkeit in allen ermittelten Metriken und der Knotengradverteilung auf. Bei der Knotengradverteilung ist nur auffällig, dass Knoten mit Grad 4 häufiger als mit Grad 3 sind. In Abbildung 4.5 ist eine deutliche Ringstruktur in dem Netz zu erkennen, jeder der Knotenpunkte wird dabei von einer anderen Linie gekreuzt, daher entstehen so viele Knoten mit Grad 4. Auch wenn Berlin eine ähnliche Ringstruktur aufweist, so fließt dieser Fakt nicht so stark in die Knotengradverteilung bei Berlin ein.

4.1.4 synthetische Vergleichstopologie- Moskau

Auch für Moskau sollen synthetisch generierte Topologien ermittelt und mit der Topologie von Moskau verglichen werden. In der folgenden Übersicht sind die Mittelwert über 32 generierte Topologien mit fester Knotenanzahl von 161 zusammengefasst:

$\overline{deg_{avg}}$	$\overline{cc_{global}}$	$\overline{ E }$	$\overline{deg_{max}}$	$\overline{deg_{min}}$	$ V $
2.30	0.02	371.87	4.2	1	161

Beim Vergleich der ermittelten Werte mit den Moskau-Werten wird wiederum eine große Ähnlichkeit deutlich. $deg_{vg}, cc_{global}, |E|, deg_{max}$ und deg_{min} sind liegen noch näher an den Werten von Moskau als bei dem Vergleich der generierten Topologien mit Berlin.

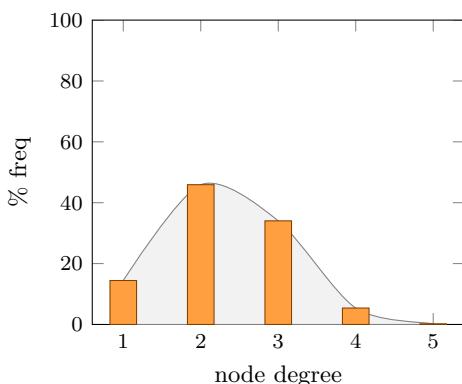


Abbildung 4.7: durchschnittliche Knotengradverteilung der synthetischen Topologie für Moskau

Diagramm 4.7 zeigt die Knotengradverteilung der generierten Topologien, welche auch ähnlich zur Verteilung von Moskau ist, d.h. in etwa gleicher Hochpunkt, ähnliches Verhalten zu Nachbarknotengrade und gleiche Min/Max Knotengrade.

4.1.5 Weitere Städte

In der folgenden Tabelle sind weitere berechnete Metriken verschiedener ausgewählter Städte dargestellt:

Stadt	deg_{avg}	cc_{global}	$ E $	deg_{max}	deg_{min}	$ V $
Paris	2.4	0.04	763	12	1	314
Rom	2.0	0.02	152	4	1	75
Frankfurt	2.5	0.09	526	11	1	208
Boston	2.0	0.03	98	5	1	49
Amsterdam	2.0	0.03	104	4	1	52
Shanghai	2.1	0.01	452	5	1	210
Karlsruhe	2.2	0.11	553	5	1	250
New-York	2.7	0.09	603	13	1	219
Sankt Petersburg	2.0	0.01	117	5	1	58
London	2.3	0.15	125	8	1	54

Wie zu erwarten war, ist der globale Clusterkoeffizient für alle ermittelten Städte sehr gering, da das Vorkommen von Clustern in einem Transportnetzwerk nicht ökonomisch wäre. Cluster würden redundante Transportwege darstellen, welche in öffentlichen Transportnetzen wenig gewünscht sind.

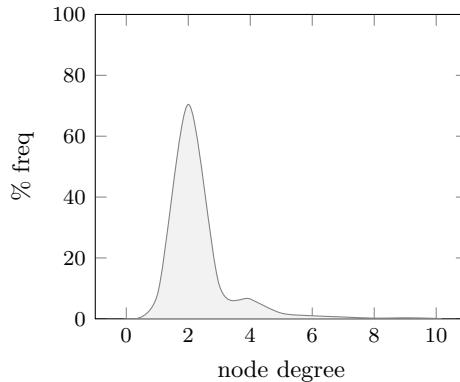


Abbildung 4.8: Kumulativ ermittelte Knotengradverteilung aller weiteren Großstädte

Weiterhin ist deutlich erkennbar, dass der durchschnittliche Knotengrad und auch deren Verteilung aller Städte recht ähnlich ist, das spiegelt sich auch in der kumulativ ermittelten Knotengradverteilung (siehe Abbildung 4.8) wieder. Auffallend ist auch die starke Ähnlichkeit zu Moskau und Berlin.

Aus den bisherigen Erkenntnissen lässt sich schlussfolgern, dass Großstädte topologisch ähnliche Eigenschaften besitzen, selbst wenn die Knotenzahl stark variiert. Ein ähnliches Phänomen tritt bei der Untersuchung von sozialen Netzwerkbeziehungen auf (vergleiche [AH10][S.195ff]), benannt als Kleines-Welt-Phänomen.

4.2 Qualitätsüberlegungen

Bei den untersuchten Städten, gab es auch Fälle in denen der ermittelte Graph keinesfalls brauchbar war. Daher wurde eine grafische Evaluierung der Städte mittels der generierten Dot-Pdfs vorgenommen. Als negatives Beispiel sei zum Beispiel die Stadt Toronto oder Istanbul erwähnt, siehe Abbildung 4.9 und 4.10. Deutlich erkennbar ist bereits, dass das Schienennetz (extrahiert wurde das U-Bahn und S-Bahn Netz) sehr klein ausfällt. Weiterhin scheinen nicht viele Haltestellen getaggt zu sein, da der grüne Graph nur einen kleinen Teil des Netzes darstellt. Dieser Umstand kann sich in naher Zukunft durchaus noch ändern, wenn alle Großstädte besser bei OSM getaggt sind. Unter Umständen sind auch die verwendeten OSM-Dateien des Metro-Projektes qualitativ schlecht.

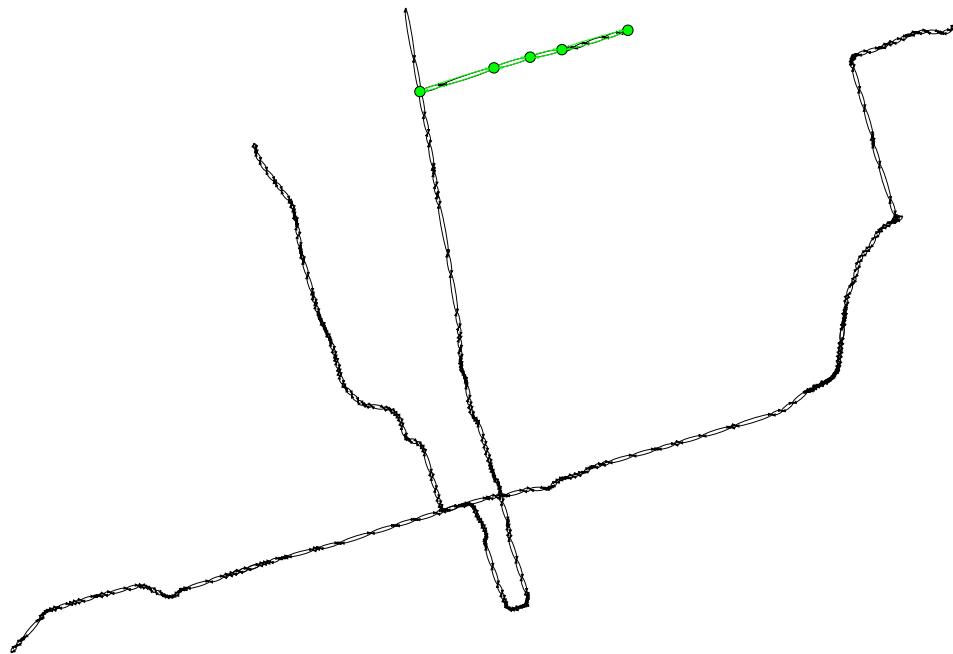


Abbildung 4.9: Toronto- schlechte OSM Daten

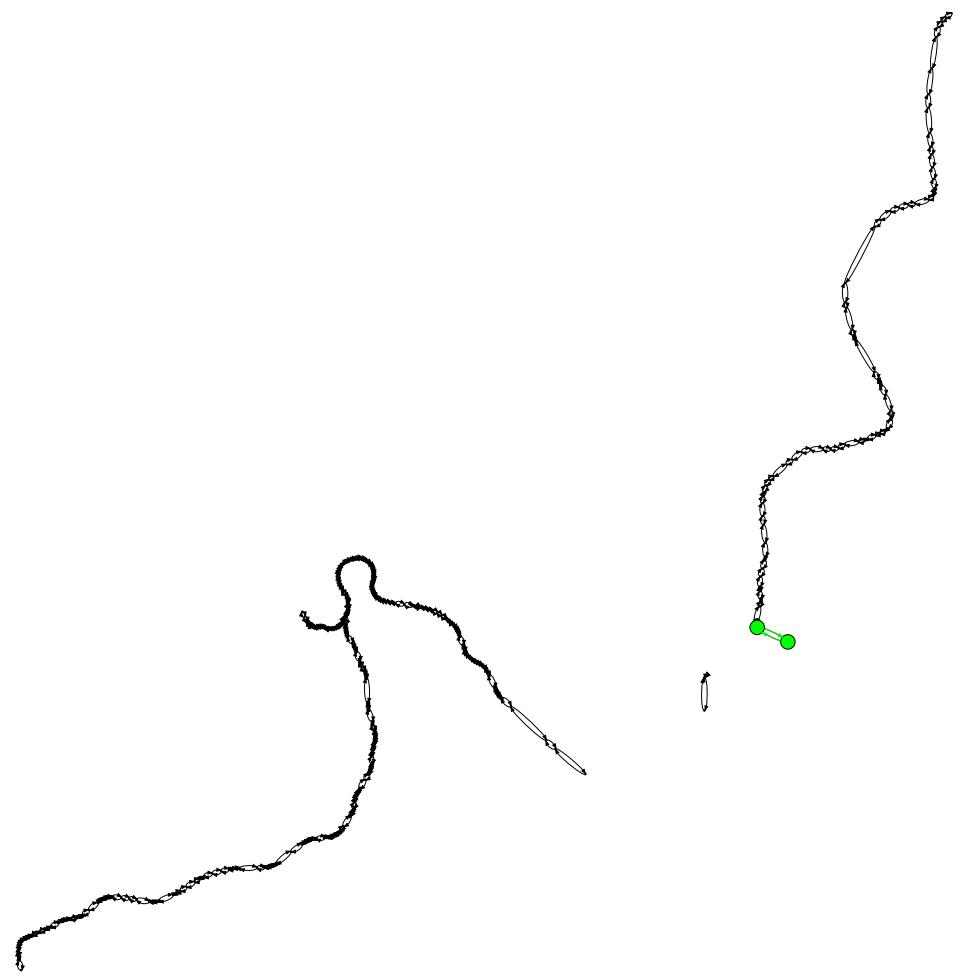


Abbildung 4.10: Istanbul- schlechte OSM Daten

5 Fazit

Im folgenden Abschnitt werden die Erkenntnisse, die in diesem Projektseminar ermittelt wurden, zusammengefasst. Außerdem werden offene Punkte und Verbesserungsmöglichkeiten dargestellt.

5.1 Zusammenfassung

Zur Simulation von verteilten Kamera Systemen können mittels OpenStreetMap realistische Topologien ermittelt werden. Zunächst wird das OSM-Format beschrieben und Techniken dargestellt, wie diese Informationen extrahiert werden können. Anschließend wird ein Routinggraph anhand der OSM-Daten erzeugt. Er dient als Grundlage für komplexere Bewegungsmodelle, welche öffentliche Transportwege mit einbeziehen. Das eingeführte Bewegungsmodell stellt dabei zunächst einen ersten Schritt dar, da es verschiedene Vereinfachungen annimmt, welche durchaus realitätsnah sind. Aber wie jedes Modell müssen Kompromisse getroffen werden. Für eine bestehende Simulation wurden Erweiterungen beschrieben und implementiert, welche für das neue Bewegungsmodell nötig waren. Formal eingeführte Graphmetriken können für die Evaluierung der Topologien herangezogen werden. Aufbauen auf diesen Metriken wurden Vergleiche zwischen Großstädten und synthetisch generierten Topologien angestellt. Es wurde festgestellt, dass die ermittelten Graphen verschiedener Großstädte ähnliche Eigenschaften aufweisen, die durchaus auch synthetisch generiert werden können.

5.2 Ausblick und offene Punkte

In dem Projektsseminar konnten einige Ziele nicht erfüllt werden. Zum Beispiel war eine graphische Repräsentation des neuen Bewegungsmodell eine optionale Anforderung gewesen. Weiterhin wurden nur die synthetischen Topologien mit den Großstadraphen verglichen. Ein Vergleich des aktuellen Bewegungsmodells mit dem neuen ist hinfällig, da das neue Modell lediglich Erweiterungen vornimmt. Dennoch wäre eine dynamische Evaluierung der Bewegungsmodelle interessant, aber in dem Rahmen dieses Projektseminars nicht realisierbar.

Das aktuell implementierte Projekt könnte auch einfach dahingehend erweitert werden um Straßen, Buslinien, Fahrradwege, usw. zu betrachten. Auch die angesprochenen OSM-Qualitätsprobleme könnten weiter untersucht werden.

„I love deadlines.
I like the whooshing sound they make as they fly by.“

DOUGLAS ADAMS

Literaturverzeichnis

- [AH10] ABRAHAM, A.; HASSANIEN, A.E.: Computational Social Network Analysis: Trends, Tools and Research Advances. Springer London, 2010 (Computer communications and networks). <http://books.google.de/books?id=-S1KiURSfRAC>. – ISBN 9781848822290
- [EGK⁺02] ELLSON, John; GANSNER, Emden; KOUTSOFIOS, Lefteris; NORTH, Stephen; WOODHULL, Gordon: Graphviz—open source graph drawing tools. In: Graph Drawing Springer, 2002, S. 594–597
- [GGS13] GOLEMBEWSKI, René; GÖRING, Steve; SCHAEFER, Guenter: Capabilities and Objectives of Distributed Image Processing on Smart Camera Systems. In: 18th IEEE Symposium on Computers and Communications (IEEE ISCC 2013). Split, Croatia, Juli 2013
- [Graa] GRAPHVIZ: GraphViz Dot Example Gallery. <http://www.graphviz.org/Gallery.php>, . – Stand: 24.02.2013
- [Grab] GRAPHVIZ: GraphViz Dot Language Definition. <http://www.graphviz.org/doc/info/lang.html>, . – Stand: 24.02.2013
- [Grac] GRAPHVIZ: GraphViz Hauptseite. <http://www.graphviz.org/>, . – Stand: 24.02.2013
- [GRS11] GOLEMBEWSKI, René; ROSSBERG, Michael; SCHAEFER, Guenter: Towards Smart Infrastructures for Modern Surveillance Networks. In: Future Security, 2011
- [GSG12] GOLEMBEWSKI, René; SCHAEFER, Guenter; GERLACH, Tobias: Efficient communication for large-scale robust image processing with smart camera devices. In: IEEE Symposium on Computational Intelligence for Security and Defence Applications. Ottawa, Canada, 2012, S. 1–8
- [HHM] HARRIS, J.; HIRST, J.L.; MOSSINGHOFF, M.J.: Combinatorics and Graph Theory (Undergraduate Texts in Mathematics). – ISBN 9780387797106
- [HW08] HAKLAY, Mordechai; WEBER, Patrick: Openstreetmap: User-generated street maps. In: Pervasive Computing, IEEE 7 (2008), Nr. 4, S. 12–18
- [Mor00] MORITZ, Helmut: Geodetic reference system 1980. In: Journal of Geodesy 74 (2000), Nr. 1, S. 128–133
- [Osb08] OSBORNE, Peter: The Mercator Projections. <http://mercator.myzen.co.uk/mercator.pdf>. Version: 2008
- [OSMa] OSM: Metro Extracts. <http://metro.teczno.com/>, . – Stand: 01.02.2013
- [OSMb] OSM: OpenStreetMap. <http://www.openstreetmap.org/>, . – Stand: 01.02.2013
- [OSMc] OSM: OpenStreetMap Mercator Projection. <http://wiki.openstreetmap.org/wiki/Mercator>, . – Stand: 01.02.2013
- [OSMd] OSM: OpenStreetMap Openptmap. <http://wiki.openstreetmap.org/wiki/Openptmap>, . – Stand: 01.02.2013
- [OSMe] OSM: OpenStreetMap Planet.osm. <http://wiki.openstreetmap.org/wiki/Planet.osm>, . – Stand: 01.02.2013
- [OSMf] OSM: OpenStreetMap Öpnvkarthe. <http://wiki.openstreetmap.org/wiki/%C3%96pnvkarte>, . – Stand: 01.02.2013
- [wika] Gelenktriebwagen. http://de.wikipedia.org/wiki/Gelenktriebwagen_NGT_D12DD. – 03.01.2013
- [wikb] Videoüberwachung. <http://de.wikipedia.org/wiki/Video%C3%BCberwachung>. – 03.01.2013