UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

Bachelorarbeit

von

Tobias Knerr

Graphdarstellung von OpenStreetMap-Wegedaten

Prof. Dr. Franz J. Brandenburg
Lehrstuhl für Informatik
Fakultät für Informatik und Mathematik
Universität Passau

Inhaltsverzeichnis

1 Zusammenfassung **4**

2 Einleitung **5**
2.1 Motivation und Ausgangssituation 5
2.1.1 Das OpenStreetMap-Projekt 5
2.1.2 Herausforderungen für die Datenqualität 6
2.1.3 Zielsetzung der Arbeit 7
2.2 Vergleich mit und Abgrenzung zu existierender Software 8
2.2.1 Vorgerenderte Karten 8
2.2.2 Validator-Plugin 8
2.2.3 Routinganwendungen und -services 9

3 OpenStreetMap-Daten **11**
3.1 Aufbau der Kartendaten 11
3.2 Relevante Tags und Relations 12
3.2.1 Verkehrswege 12
3.2.2 Attribute von Verkehrswegen 12
3.2.3 Barrieren 16
3.2.4 Abbiegebeschränkungen 16

4 Algorithmen **18**
4.1 Auswertung von Zugangsrechten in der Fahrzeugklassenhierarchie 18
4.2 Transformation der OpenStreetMap-Daten in eine Graphstruktur 19
4.2.1 Abstraktionsverfahren 19
4.2.2 Graphaufbau 21

5 Implementierung **27**
5.1 Technologie 27
5.2 Benutzerschnittstelle und Features 27
5.2.1 Benutzerschnittstelle des JOSM-Editors 28
5.2.2 Benutzerschnittstelle und Featureumfang des Graphview-Plugins . 29
5.3 Dauerhaft gespeicherte Daten 32
5.3.1 Konfigurationsoptionen 32
5.3.2 Ruleset-Dateien 33
5.4 Programmarchitektur 34
5.4.1 Modul core 35

5.4.2 Modul plugin 35

6 Fazit **36**
6.1 Schwachstellen im OpenStreetMap-Projekt 36
6.2 Ausblick 37

A Konfigurationseinträge **38**

B Installation und Start des Plugins **39**

C Literaturverzeichnis **41**

D Referenzierte Software und Webservices **42**

1 Zusammenfassung

Diese Arbeit präsentiert einen Ansatz für die Qualitätssicherung im OpenStreetMap-Projekt: Visualisierung eines aus Wegedaten des Projekts erzeugten Routinggraphen. Die zentralen Aspekte der Datenstruktur des OpenStreetMap-Datensatzes für Zwecke des Routings werden beschrieben und mögliche Lösungsalgorithmen für verschiedene Herausforderungen bei der Verarbeitung der Daten vorgestellt. Eine Umsetzung des Konzepts erfolgt in Form eines Plugins für den OpenStreetMap-Dateneditor JOSM.

2 Einleitung

2.1 Motivation und Ausgangssituation

2.1.1 Das OpenStreetMap-Projekt

Karten und Geodaten sind ein wichtiger Bestandteil vieler populärer Anwendungen und Webangebote. Besonders auf Webseiten werden die zugrundeliegenden Karten dabei oft um individuelle Daten angereichert, um einen speziellen Einsatzzweck zu erfüllen – Anfahrtsbeschreibungen für Unternehmen, die Darstellung der Wandertouren des letzten Sommerurlaubs auf privaten Webpräsenzen und individuelle Verzeichnisse von Points of Interest sind nur einige wenige Beispiele. Üblicherweise werden die Basiskarten dabei nicht selbst erstellt. Stattdessen wird auf vorhandenes Material zurückgegriffen – Dienste wie Google Maps bieten hierfür eigens APIs an.

Die Möglichkeiten, die sich bei der Verwendung eines solchen Dienstes bieten, sind jedoch verhältnismäßig beschränkt. Das Aussehen der Karte lässt sich kaum verändern, die Darstellung von Daten bleibt auf Marker und Overlays beschränkt. Weder kann bei der Integration in eine Webseite eine Anpassung an den Stil und die Farbwahl erfolgen, noch kann eine Auswahl getroffen werden, welche Elemente der Karte überhaupt relevant sind – Spezialkarten etwa für Wanderer oder Radfahrer lassen sich so nicht herstellen. Für derartige Vorhaben reicht die Auslieferung fertiger Bilder nicht aus, es wäre Zugang zu den Rohdaten nötig. Dies gilt umso mehr für Routing, Verkehrssimulationen oder ein Durchsuchen der vorhandenen Daten.

Einen für dererlei Zwecke uneingeschränkt tauglichen Datenbestand weltweit bereitstellen möchte das Projekt OpenStreetMap. Durch die Mitarbeit zahlreicher Freiwilliger werden hier aus GPS-Informationen und mancherorts verfügbaren Luftbildern umfangreiche Kartendaten gesammelt. Die Wahl einer Lizenz, die die Erstellung und Weitergabe abgeleiteter Werke für beliebige Zwecke erlaubt, soll auch rechtliche Hindernisse für eine möglichst vielfältige Anwendung beseitigen. Mit derzeit etwa 120 000 angemeldeten Nutzern [sta] und einem stetig wachsenden Datenbestand hat das Projekt durchaus Aussicht auf Erfolg.

2.1.2 Herausforderungen für die Datenqualität

Ein zentrales Problem bei Projekten in dieser Größenordnung – gerade auch, wenn die Daten von Laien geliefert werden – ist allerdings die Qualitätssicherung. Wiki-Projekte setzen hier üblicherweise auf ein Grundprinzip, das ursprünglich als Erfolgsfaktor für Open-Source-Software ausgemacht wurde (prominent formuliert etwa als „Linus' Law" von Eric S. Raymond [Ray01]): Fehler lassen sich leicht finden und beheben, wenn eine große Mitarbeiter- und Nutzerbasis daran aktiv teilnehmen kann. Mit menschlicher Arbeitskraft und Intelligenz sind in der Tat auch bei Datensammlungen viele Probleme zu erkennen und zu beheben, die einer automatischen Prüfung leicht entgangen wären, und dies gilt umso mehr in Fällen, in denen die Korrektheit einer Information ausschließlich im Vergleich mit der Realität zweifelsfrei festgestellt werden kann.

In einem Bereich kann sich dieser Ansatz bei OpenStreetMap bereits entfalten: Beim Aufspüren von Fehlern in den gerenderten Karten. Darstellungsprobleme fallen oft schon beim groben Betrachten einer Karte auf und lassen sich auch ohne Kenntnis der zugrundeliegenden Datenstruktur ausmachen. Dass Mängel in der grafischen Darstellung offensichtlich sind, macht sich OpenStreetMap durchaus zunutze: So setzt etwa das Bug-Reporting-Tool OpenStreetBugs [osb] darauf, dass Nutzer Fehlermeldungen direkt auf der Karte anbringen.

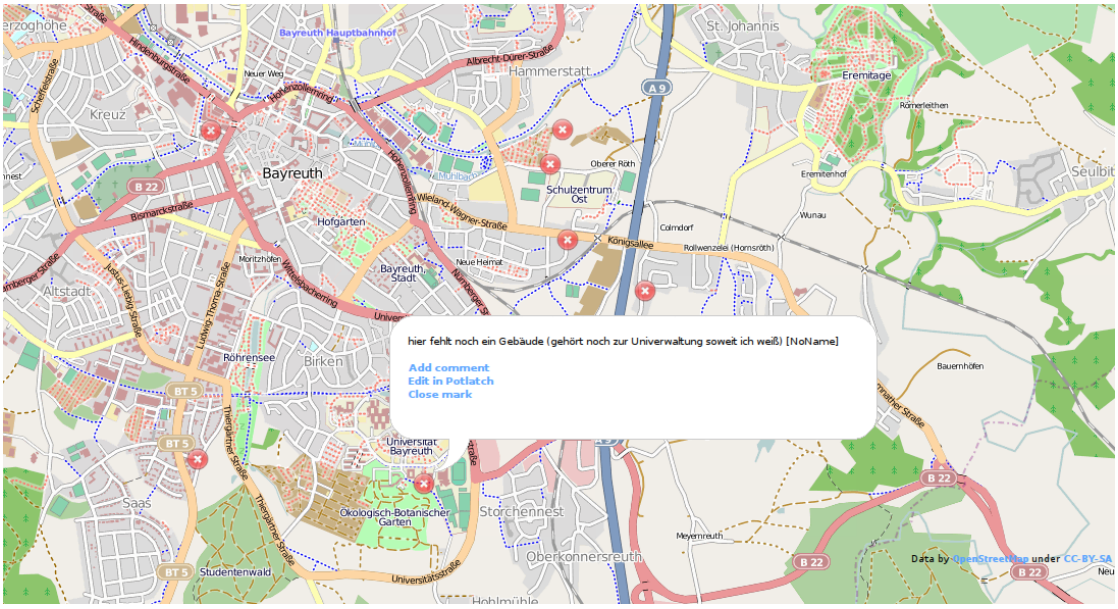


Abbildung 2.1: OpenStreetMap-Kartenmaterial mit OpenStreetBugs-Fehlermeldungen

Schwer möglich ist eine überblicksartige Kontrolle des Datenbestandes hingegen für Aspekte, die in Kartenansichten gemeinhin nicht dargestellt werden. So beeinträchtigen übereinanderliegende Duplikate von Wegen die Kartendarstellung ebensowenig wie fehlende, veraltete oder falsche Informationen zu Höchstgeschwindigkeiten, Abbiegeverboten und vergleichbaren Restriktionen. In ähnlicher Weise können Wege ohne gemeinsamen Punkt im Kartenbild eine Kreuzung vortäuschen, wenn ihre Enden hinreichend nahe beieinander liegen. Für eine Verwendung von OpenStreetMap-Daten etwa in Routingsoftware ist all dies aber von großer Bedeutung.

2.1.3 Zielsetzung der Arbeit

Als naheliegender Lösungsansatz für die zuvor beschriebenen Probleme ergibt sich, die Vorgehensweise bei der Kontrolle des Renderings, nämlich die optische Begutachtung durch Menschen, auch für Routingdaten anwendbar zu machen. Hierzu ist es nötig, diese Information zu visualisieren. Dies ist ohne prinzipielle Schwierigkeiten möglich: Die zentrale Datenstruktur für Routingalgorithmen, der Graph, lässt sich unmittelbar optisch umsetzen. Die Ähnlichkeit zu anderen im Verkehrskontext verwendeten Darstellungsformen (etwa Linienplänen von öffentlichen Verkehrsmitteln), die einfache Grundstruktur aus wenigen Elementtypen (Knoten und Kanten) und nicht zuletzt im vorliegenden Anwendungsfall auch die Ähnlichkeit zum OpenStreetMap-Datenmodell (vgl. 3.1) lassen erwarten, dass eine derartige Darstellung für Nutzer leicht verständlich ist. Zusätzliche Information über die Knoten und Kanten des Routinggraphen kann durch die Verwendung von Farben vermittelt werden.

Ziel der vorliegenden Arbeit ist die Entwicklung eines Werkzeugs, das die Mitglieder des OpenStreetMap-Projekts in dieser Weise zur Visualisierung der Routinginformationen aus OpenStreetMap-Daten nutzen können.

Als Plattform für die Umsetzung wurde der OpenStreetMap-Dateneditor JOSM gewählt, der durch Plugins leicht erweiterbar ist [jos]. Die Integration in einen Editor bietet verschiedene Vorzüge, so ermöglicht sie etwa, neu erzeugte Daten bereits vor der Integration in die OpenStreetMap-Datenbank zu überprüfen, was nachträgliche Korrekturen erspart. Es ist auch nicht nötig, eine zentrale Infrastruktur zu betreiben, wie sie eine webbasierte Lösung erfordern würde. Gegenüber dem Anbieten einer eigenständigen Software entfällt die Installation. In Kauf genommen wird, dass Nutzer anderer Editoren nicht aus ihrer gewohnten Umgebung heraus auf die Funktionalität zugreifen können.

Voraussetzung für die Visualisierung ist auch eine Umsetzung der OpenStreetMap-Rohdaten in einen Routinggraphen. Dadurch bietet sich die Gelegenheit, Ansätze für den Umgang mit komplexeren Aspekten der OpenStreetMap-Daten zu entwerfen und darzustellen, siehe hierzu besonders Kapitel 4. Lösungen für diese Problemstellungen sind für jede Software erforderlich, die OpenStreetMap im Kontext des Routings zum Einsatz bringen will.

2.2 Vergleich mit und Abgrenzung zu existierender Software

Die Community um OpenStreetMap hat bereits zahlreiche Programme geschaffen, die zur Qualitätssicherung dienen können – bisweilen ist dies der eigentliche Einsatzzweck, in anderen Fällen ein Nebennutzen. In diesem Abschnitt soll ein kurzer Überblick über diese Softwarelandschaft geboten werden. Insbesondere wird auch auf die Schwächen der bestehenden Ansätze eingegangen.

2.2.1 Vorgerenderte Karten

Das Konzept, routingrelevante Straßenattribute optisch darzustellen, wird teilweise bereits verfolgt: Für Höchstgeschwindigkeiten existieren etwa Darstellungen als vorgerenderte Kacheln, die in einer Online-Karte per OpenLayers angezeigt werden können. [max]

Eine solche Umsetzung ist allerdings wenig flexibel: Um sowohl die unterschiedlichen Wegattribute auszudrücken als auch der Tatsache gerecht zu werden, dass sich die Werte der Attribute für unterschiedliche Fahrzeugklassen und Situationen unterscheiden können, müsste eine Vielzahl von Kachelsets erstellt und ausgeliefert werden. Prinzipiell decken Karten mit derartigen Einfärbungen auch nur einen Teil der in dieser Arbeit vorgesehenen Funktionalität ab, es fehlt etwa die Veranschaulichung der Bewegungsmöglichkeiten an komplexen Kreuzungen mit Abbiegebeschränkungen und sonstigen Vorschriften.

2.2.2 Validator-Plugin

Ein verbreitetes Hilfsmittel für die Qualitätssicherung ist das Validator-Plugin für den JOSM-Editor [val]. Dieses Werkzeug führt automatische Tests nach bestimmten Regeln durch und listet potentielle Fehler auf. Es kann diese Probleme in der Bearbeitungsansicht des Editors hervorheben und sogar versuchen, sie eigenständig zu beheben.

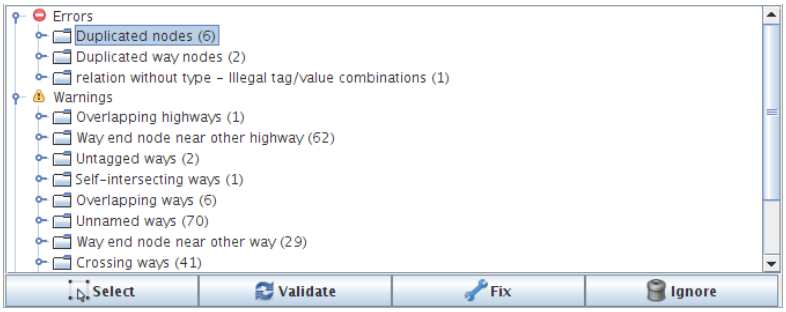


Abbildung 2.2: Fehlermeldungen des Validator-Plugins

Die Möglichkeiten des Validators sind natürlich auf solche Bereiche beschränkt, die maschinelle Kontrolle ohne Kenntnis der Realität erlauben. Ein großer Teil der Fehlerquellen erfüllt diese Kriterien jedoch nicht, was andere Werkzeuge erforderlich macht.

2.2.3 Routinganwendungen und -services

Nicht zuletzt taugen auch Programme, die tatsächliches Routing durchführen, für eine experimentelle Kontrolle der Information. Ermittelte Strecken, die nicht den Erfahrungen aus der Wirklichkeit entsprechen, weisen auf fehlerhafte oder ungenügende Daten hin, die dann korrigiert werden können. Beispiele für Software mit Routingfunktionalität auf OpenStreetMap-Daten sind die Java-Anwendung Traveling Salesman [tra] und das Webangebot OpenRouteService [ors].

Systematische Prüfung durch Projektmitglieder wäre auf diese Weise allerdings sehr aufwendig, da mit jeder Routenberechnung nur ein kleiner Teil der Daten verwendet wird und Abweichungen durch Fehler nicht notwendigerweise auffällig genug sind. Die Verwendung in realer Routingsoftware kann in größerem Umfang dann Ausgangspunkt für Fehlermeldungen sein, wenn Nutzer im Alltagseinsatz der Karten Probleme entdecken. Dies setzt allerdings eine größere Verbreitung der OpenStreetMap-Nutzung voraus, als momentan gegeben ist, und erfordert geeignete Feedbackkanäle, die die Anwender auch zu nutzen bereit sind.

Eine Sonderstellung unter den routingfähigen Programmen nimmt das Routing-Plugin für JOSM ein [rou]: Die Experimentier- und Kontrollfunktion wird hier durch die Integration in Bearbeitungssoftware zum Haupteinsatzzweck. Neben den allgemeinen Problemen eines solchen experimentellen Ansatzes fällt hier allerdings auch auf, dass das Plugin kaum Wegeattribute beachtet und keine Auswahl etwa des Fahrzeugtyps für das Routing erlaubt.

Da sich Routingfunktionalität aber mit der Grapherstellung aus dieser Arbeit gut ergänzen kann, ist eine dahingehende Erweiterung des im Rahmen dieser Arbeit entwickelten Plugins in Zukunft denkbar – möglicherweise auch unter Verwendung vorhandenen Codes des Routing-Plugins.

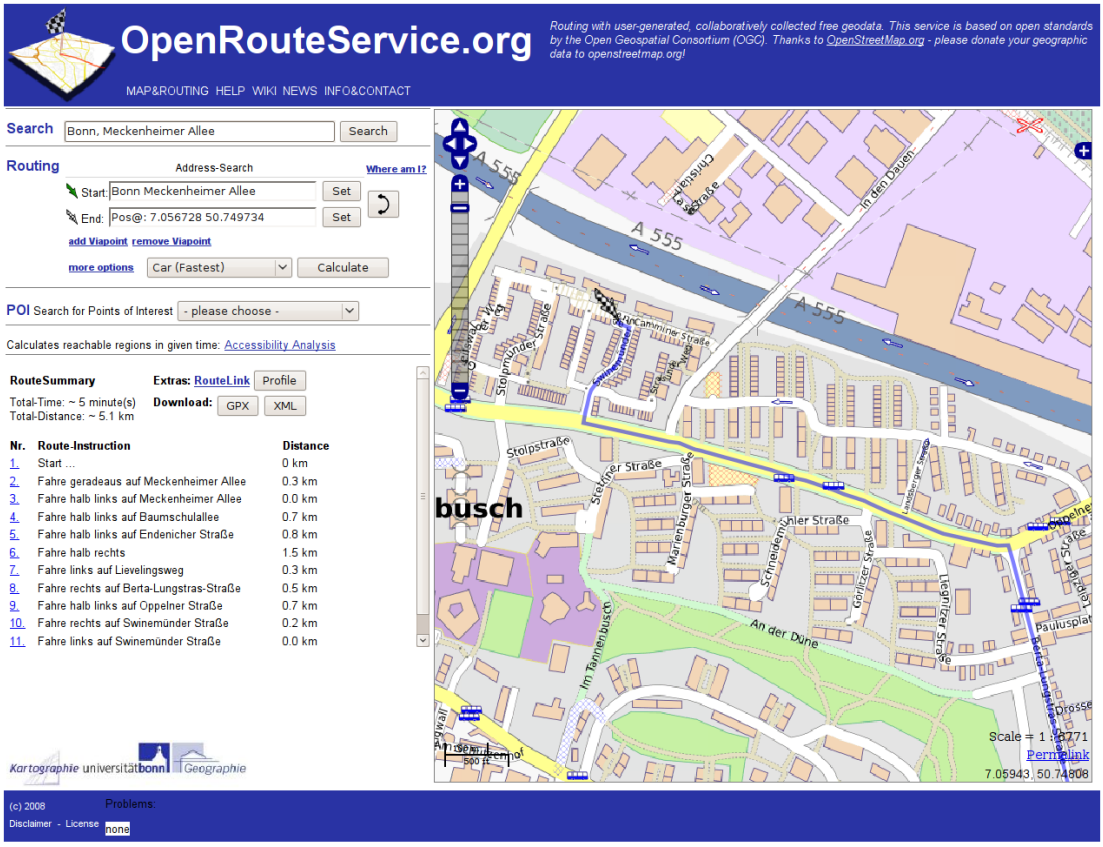


Abbildung 2.3: OpenRouteService [ors], eine Webanwendung für das Routing auf OpenStreetMap-Daten (Software Research Group Cartography, Department of Geography, Universität Bonn; Daten von OpenStreetMap, CC-BY-SA 2.0)

3 OpenStreetMap-Daten

Das folgende Kapitel dient dazu, den Aufbau der Daten, mit denen gearbeitet werden soll, darzustellen. Als Quellen dienen hier die Dokumentation im Wiki von OpenStreetMap ([wik]) sowie das Buch „OpenStreetMap“ von Frederik Ramm und Jochen Topf ([RT09]).

Besonders bei der Auswahl und Interpretation der zu berücksichtigenden Informationen stellt allerdings die instabile Natur eines Wikis eine Schwierigkeit dar. Hinzu kommt, dass die vorhandene Dokumentation im Projekt OpenStreetMap weithin nicht als bindend angesehen wird; die Möglichkeit, eigene Vorgehensweisen beim Eintragen von Daten zu entwickeln, wird betont – diesbezügliche Hinweise finden sich etwa in [RT09: S. 59] und [wik: Map Features]. Aus diesen Gründen ist bei einigen Fragen eine eigene Abwägung und Interpretation nötig. Hier kommt auch das Werkzeug „Tagwatch“ ([tag]) zum Einsatz: Es bietet Statistiken über die tatsächliche Häufigkeit der Verwendung von Tags. In dieser Arbeit werden die europäischen Nutzungshäufigkeiten als Grundlage genommen – anders als etwa in den USA, wo auf frei verfügbare Behördendaten als Ausgangsmaterial zurückgegriffen werden konnte [wik: TIGER], sind die Werte nicht durch flächendeckende Importe beeinflusst und infolge des europäischen Ursprungs des Projekts ist die Datenbasis umfangreich.

3.1 Aufbau der Kartendaten

Sämtliche OSM-Daten bestehen aus den drei Objekttypen *Nodes*, *Ways* und *Relations*.

Node Punkt mit geographischer Koordinate (Länge und Breite). Kann Teil von Ways sein.

Way Geordnete Folge von Nodes. Nodes können auch wiederholt in einem Way auftreten. Sind erster und letzter Node identisch, handelt es sich um einen geschlossenen Way. Geschlossene Ways werden auch zur Darstellung flächiger Objekte verwendet.

Relation Geordnete Folge von Paaren aus *Role* und *Member*-Objekt. Eine Role ist ein Unicode-String, *Member* können beliebige Objekte (Nodes, Ways oder Relations) sein. Weder Roles noch Members sind auf ein einzelnes Vorkommen pro Relation beschränkt.

Was ein konkretes Objekt darstellt, ergibt sich aus seinen Attributen, den *Tags*. Tags sind Paare aus einem *Key*- und einem *Value*-String von Unicode-Zeichen. Jeder Key kann

pro Objekt nur einmal auftreten. Per Konvention werden Tags meist als `key = value` geschrieben, diese Schreibweise wird auch in dieser Arbeit verwendet.

Siehe auch [wik: Elements].

3.2 Relevante Tags und Relations

Für den Zweck, landgebundene Verkehrswege auszuwerten, ist nur ein kleiner Teil der existierenden Tags und Relations relevant. Diese werden im Folgenden dargestellt.

3.2.1 Verkehrswege

Verkehrswege in OSM werden grundsätzlich als Ways dargestellt. Üblicherweise tragen solche Ways ein Tag mit dem Key `highway`, dies ist aber nicht zwingend. Als Ausnahme seien hier etwa Bahnsteige (`railway = platform`) genannt, die wegen ihrer Zugehörigkeit zur Schienennetz-Infrastruktur den `railway`-Key tragen, aber für das Fußgängerrouting relevant sind.

Flächen

Um neben linienartigen Verkehrswegen auch Flächen und Plätze abdecken zu können, existiert das ergänzende Tag `area = yes`, das einen geschlossenen Way als Fläche ausweist. Geschlossene Ways ohne das Tag würden lediglich ringförmig verlaufende Straßen darstellen.

Für die Erreichbarkeit etwa von Abzweigungen hat `area = yes` keine Auswirkungen, es beeinflusst nur die Entfernung zu einer Abzweigung durch den Längenunterschied zwischen der direkten Überquerung einer Fläche und dem Weg entlang des Flächenrandes.

3.2.2 Attribute von Verkehrswegen

Es gibt zahlreiche verkehrsrechtliche und physikalische Einschränkungen der Nutzbarkeit vorhandener Verkehrsverbindungen, die durch Tags in OSM abgebildet werden können. Solche Tags ergänzen stets die grundlegende Auszeichnung als Verkehrsweg, ersetzen sie nicht.

Verkehrsmittel und Nutzergruppen

Um Einschränkungen der Nutzbarkeit für bestimmte Gruppen von Fahrzeugen oder Nutzungsarten bzw. Nutzergruppen darzustellen, dienen in OSM Tags, deren Key die

betroffene Fahrzeuggruppe und dessen Value die zulässige Nutzungsart ist. Als einige wenige Beispiele seien genannt:

- `foot = no`: kein Zugang für Fußgänger
- `motor_vehicle = delivery`: Nutzung mit motorisierten Fahrzeugen nur für Lieferverkehr
- `hgv = destination`: LKW (heavy goods vehicles) nur für Anlieger frei

Statt einer Fahrzeuggruppe kann auch der Key `access` verwendet werden, bei dem die Beschränkung für alle Fortbewegungsarten gilt. Verschiedene Fahrzeuggruppen sind Teilmengen von anderen, wobei die genauen Zuordnungen nicht einheitlich definiert sind.

Listen von Verkehrsmitteln und Nutzungsarten werden definiert auf [wik: Key:access]. In dieser Arbeit werden als Nutzungsarten akzeptiert:

- `designated` – uneingeschränkt nutzbar, offiziell für das Verkehrsmittel ausgewiesen
- `yes` – uneingeschränkt nutzbar
- `permissive` – Nutzung geduldet
- `destination` – für Anlieger nutzbar
- `delivery` – für Lieferverkehr nutzbar
- `agricultural` – für landwirtschaftlichen Verkehr nutzbar
- `forestry` – für forstwirtschaftlichen Verkehr nutzbar
- `private` – privat, nur mit Genehmigung des Eigentümers nutzbar
- `no` – nicht nutzbar

Die verfügbaren Fahrzeuggruppen können regional verschieden sein und sind konfigurierbar, siehe hierzu 5.3.2.

Geschwindigkeitslimits

Zur Darstellung von Geschwindigkeitsbeschränkungen werden die beiden Keys `maxspeed` (Höchstgeschwindigkeit) und `minspeed` (Mindestgeschwindigkeit) genutzt. Die Values sind oft rein numerisch, in diesem Fall wird als Einheit km/h angenommen. Alternativ kann eine Einheit (üblicherweise `mph`) nach dem numerischen Wert angegeben werden.

Limits für Fahrzeugattribute

OSM bietet eine Gruppe von Keys zur Beschreibung von Obergrenzen von Fahrzeugeigenschaften. In dieser Arbeit verwendet werden die folgenden:

Key	Bedeutung	Value: vorgegebene Einheit
`maxweight`	maximales zulässiges Gewicht	Tonnen
`maxaxleload`	maximale zulässige Achslast	Tonnen
`maxheight`	maximale zulässige Höhe	Meter
`maxwidth`	maximale zulässige Breite	Meter
`maxlength`	maximale zulässige Länge	Meter

Einbahnregelungen

Da Ways *geordnete* Folgen von Nodes sind, haben sie stets eine Richtung. In der Regel sind Verkehrswege jedoch in beide Richtungen nutzbar. Einbahnregelungen werden als `oneway`-Tag vermerkt:

- `oneway = yes`: der Way ist nur in seine Richtung nutzbar
- `oneway = -1`: der Way ist nur entgegen seiner Richtung nutzbar
- `oneway = no`: der Way ist in beide Richtungen nutzbar

Für die meisten Straßentypen ist `oneway = no` als Standard vorgegeben und wird daher nicht explizit gesetzt.

Neben `yes/no` existieren noch `true/false` und `1/0` als alternative, bedeutungsgleiche Wertepaare. Die Alternativen werden seltener verwendet, sind aber mit einem Anteil von etwa einem Viertel der `oneway`-Tags dennoch verbreitet. [tag: oneway]

Von den Auswirkungen der `oneway`-Tags sind Fußgänger laut [wik: DE:Key:oneway] grundsätzlich ausgenommen. Für die bei Einbahnstraßen verbreiteten Ausnahmeregelungen für Radfahrer existieren die Tags `cycleway = opposite`, `cycleway = opposite_track` und `cycleway = opposite_lane`. Bei jedem der Tags können sich Fahrräder entgegen der Einbahnrichtung bewegen, abhängig vom Tag entweder ohne gesonderte Spur oder auf einem baulich bzw. durch Fahrbahnmarkierung abgetrennten Radweg.

Physische Eigenschaften des Weges

Neben gesetzlichen Vorgaben für Wege wird deren Nutzbarkeit für manche Fahrzeuge auch durch die Wegbeschaffenheit bestimmt. Gängig sind hier Angaben zur Breite (Key `width` mit numerischer Angabe der Breite in Metern) und Steigung.

Als Key für Steigungen dient `incline`. Dessen Values sind positive oder negative prozentuale Steigungen (als numerische Werte, gefolgt von einem Prozentzeichen). Positive Werte stehen dabei für eine Steigung in Richtung des Ways, negative für eine Steigung entgegen der Richtung des Ways. Zur Kennzeichnung von Strecken mit Steigungen existieren auch die älteren, an Nodes verwendeten Tags `highway = incline` sowie `highway = incline_steep`, die laut Tagwatch häufiger zum Einsatz kommen als

`incline = *`. Es existieren jedoch keine aussagekräftigen Definitionen für diese Tags, daher wird für die Zwecke dieser Arbeit lediglich der `incline`-Key verwendet.

Es finden sich zudem verschiedene, sich teils überschneidende Ansätze zur Beschreibung von Oberflächeneigenschaften: die Keys `surface` zur Beschreibung des Oberflächenmaterials und `tracktype` zur groben Kennzeichnung des Ausbauzustands sowie `smoothness`, das die Glattheit der Oberfläche (in Hinblick auf ihre Nutzbarkeit durch bestimmte Fahrzeugtypen) darstellen soll.

`surface` kann zahlreiche unterschiedliche Werte annehmen, von denen eine Auswahl auf [wik: Key:surface] aufgelistet wird. Der Wertebereich von `tracktype` umfasst die fünf Stufen `grade1` bis `grade5`, wobei `grade1` den besten Ausbauzustand kennzeichnet.

`smoothness` erlaubt ein fest vorgegebenes Set von natürlichsprachigen Beschreibungen (von `excellent` bis `impassable`). Da der Key allerdings kaum Akzeptanz findet und deutlich weniger verwendet wird als die beiden vorgenannten (nach [tag: smoothness] und [tag: surface] bzw. [tag: tracktype] ein Verhältnis von ca. 1:100), wird er für die Auswertung in dieser Arbeit nicht berücksichtigt.

Implikationen

Manche der vorgenannten Informationen sind für bestimmte Typen von Verkehrswegen unausgesprochen gültig: In Fußgängerzonen sind LKW im Normalfall nicht zugelassen, Treppen sind nicht für motorisierte Fahrzeuge geeignet. Sind solche Implikationen international anwendbar, werden sie im OSM-Wiki auf der Seite des jeweiligen Tags vermerkt. Darüber hinaus sind die Vorgaben der lokalen Gesetzgebung anzuwenden. Derartige Richtlinien werden etwa auf [wik: OSM tags for routing/Maxspeed] und [wik: OSM tags for routing/Access-Restrictions] dokumentiert.

Implikationen können generell durch explizites Setzen von Tags überschrieben werden. Wird für einen Key also bereits ausdrücklich ein Value vergeben, hat eine Implikation für diesen Key keine Auswirkungen mehr. So könnte eine Fußgängerzone durch ausdrückliches Setzen von `vehicle = delivery` für Fahrzeuge im Lieferverkehr freigegeben werden.

Weitere Tags

Da die Ausdrucksmöglichkeiten der etablierten Tags für zahlreiche Sachverhalte, etwa bedingte Attribute (zeitabhängige Höchstgeschwindigkeiten, auf bestimmte Nutzergruppen beschränkte Limits uvm.) nicht ausreichen, gibt es verschiedene diskutierte Ansätze für eine Erweiterung der Taggingkonzepte. Diese Arbeit beschränkt sich jedoch auf die beschriebenen etablierten Vorgehensweisen.

3.2.3 Barrieren

Hindernisse auf Verkehrswegen können durch Nodes ausgedrückt werden, die einen Tag mit dem Key `barrier` tragen. Barrieren können an beliebiger Stelle in einem Way auftreten und sind ohne weitere Angaben unpassierbar (was einem implizierten `access = no` entspricht). Die Passierbarkeit kann durch Attribute entsprechend denen bei Verkehrswegen (siehe 3.2.2) beschrieben werden.

3.2.4 Abbiegebeschränkungen

Einschränkungen der Abbiegemöglichkeiten, die sich nicht aus den Attributen der einzelnen Ways ergeben, werden als Relations erfasst. Etabliert haben sich hier Relations mit dem Tag `type = restriction`. Als weitere Informationen enthalten diese Relations

- ein Tag mit Key `restriction`. Der Wert beginnt entweder mit dem Präfix `no_` oder dem Präfix `only_`
- einen Way als Member mit der Role `from`
- einen Way als Member mit der Role `to`
- beliebig viele Ways oder beliebig viele Nodes mit der Role `via`

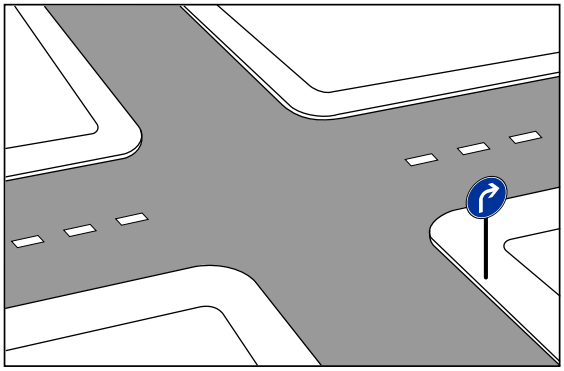
Beginnt der Wert des `restriction`-Tags mit `no_`, so verbietet die Relation, sich vom `from`-Member über `via`-Members zum `to`-Member zu bewegen. Beginnt er mit `only_`, so ist ausschließlich eine solche Fortbewegung erlaubt. Als Beispiel für eine `only_`-Beschränkung siehe Grafik 3.1.

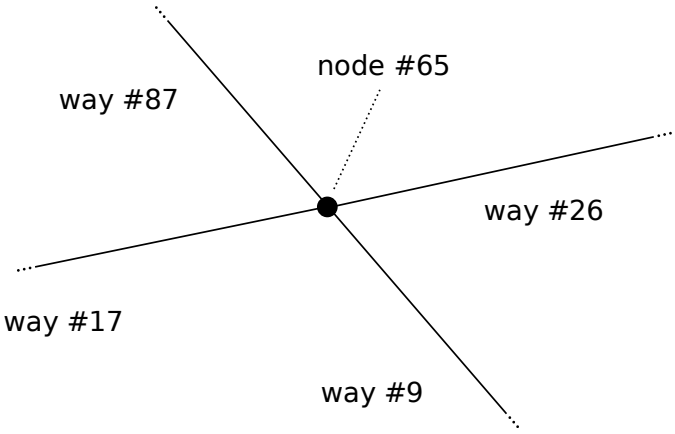
Die Gültigkeit der Abbiegebeschränkung kann zudem durch weitere Tags eingeschränkt werden:

- der Key `except` kann als Wert eine mit Strichpunkten getrennte Liste von Fahrzeugtypen haben, die nicht von der Relation betroffen sind
- die Keys `day_on`, `day_off`, `hour_on` und `hour_off` erlauben die Einschränkung auf bestimmte Uhrzeiten oder Wochentage

Solche Zusatztags werden allerdings eher selten verwendet [tag], so dass sie in dieser Arbeit zunächst nicht berücksichtigt werden.

Abbiegebeschränkungen in dieser Form werden beschrieben in [wik: Relation:restriction] und [RT09: S. 89–91].





relation #51			
members		tags	
role	member	key	value
from via to	way #9 node #65 way #26	type restriction	restriction only_right_turn

Abbildung 3.1: Beispiel für eine als OpenStreetMap-Relation dargestellte Abbiegevorschrift. Die numerischen IDs sind hier beliebig gewählt.

4 Algorithmen

Bei der Konvertierung der OSM-Ausgangsdaten in eine Graphstruktur sind verschiedene Herausforderungen zu bewältigen. Hier wird für die kritischen Arbeitsschritte der für die Implementierung gewählte Algorithmus beschrieben.

4.1 Auswertung von Zugangsrechten in der Fahrzeugklassenhierarchie

Zur Auswertung der Zugangsrechte für Fahrzeugklassen (siehe 3.2.2) auf einem Way sind die folgenden Informationen erforderlich:

- die existierenden Verkehrsmittelklassen
- die Inklusionsbeziehungen zwischen den Verkehrsmittelklassen
- die Tags des auszuwertenden Ways
- die Implikationen von Tags
- die Gruppe der „Basistags“

Bei „Basistags“ handelt es sich um ein nicht ausdrücklich im OpenStreetMap-Wiki verwendetes Konzept. Es soll sicherstellen, dass die Implikationen explizit gesetzter Zusatztags Vorrang vor den Implikationen des Straßentyps (angegeben durch den „Basistag“) erhalten. So sollen etwa die durch eine Einstufung als Kraftfahrstraße (`motorroad = yes`) ausgedrückten Verbote für bestimmte Verkehrsmittel zuverlässig die Informationen des Straßentyps selbst ersetzen, der diese Verkehrsmittel ohne das Zusatztag erlauben würde.

Die eigentliche Auswertung erfolgt in den folgenden Schritten:

1. es wird ein assoziatives Array angelegt, das jeder Verkehrsmittelklasse einen Wert für das Nutzungsrecht zuweist
2. als Initialisierung wird das Nutzungsrecht für alle Verkehrsmittelklassen auf *undefiniert* gesetzt
3. wird für eine Verkehrsmittelklasse ein Nutzungsrecht impliziert, wird dieses gesetzt. Dabei werden zunächst die Implikationen des „Basistags“, dann die Implikationen sonstiger Tags gesetzt. Letztere überscheiben also „Basistag“-Implikationen.

4. wird für eine Verkehrsmittelklasse ein Nutzungsrecht als Tag explizit angegeben, wird dieses Nutzungsrecht gesetzt.

Als Resultat erhält man ein assoziatives Array, in dem für alle Verkehrsmittelklassen entweder ein Nutzungsrecht gesetzt oder dieses als *undefiniert* gekennzeichnet ist. Um daraus das Nutzungsrecht für eine Klasse zu ermitteln, wird zunächst auf ein direkt zugeordnetes Nutzungsrecht geprüft. Ist hier kein Nutzungsrecht definiert, wird schrittweise die nächstgrößere einschließende Klasse herangezogen, bis ein definiertes Nutzungsrecht vorliegt oder die äußerste Klasse erreicht wurde.

Die hier beschriebene Vorgehensweise orientiert sich an einem im OpenStreetMap-Wiki vorgeschlagenen Verfahren [com], verzichtet allerdings auf den tatsächlichen Aufbau einer Baumstruktur. Die Hierarchieinformation wird stattdessen dadurch ausgedrückt, dass für jede Verkehrsmittelklasse eine geordnete Liste der einschließenden Klassen existiert. Diese inhaltlich gleichwertige Darstellungsform erlaubt, die Auswertung leicht auf eine einzelne Klasse und ihre einschließenden Klassen zu beschränken, wenn die übrigen Klassen nicht von Interesse sind. Der Aufwand für die einmalige Durchführung des Verfahrens ist von der Tiefe der Verkehrsmittelhierarchie und der Anzahl der Implikationen abhängig, aber nicht vom maßgeblichen Skalierungsfaktor, dem Umfang der geladenen Kartendaten.

Andere Kriterien für die Nutzbarkeit von Fahrzeugen, wie Einbahnregelungen oder die Auswertung von Fahrzeug- und Verkehrswegattributen (Gewichtsbeschränkungen und dergleichen), werden zusätzlich geprüft und sind nicht Teil dieses Verfahrens. Das Verfahren wird analog auch für Nodes durchgeführt, da es sich um Barrieren handeln kann (3.2.3).

4.2 Transformation der OpenStreetMap-Daten in eine Graphstruktur

Angesichts der Vielfalt der in den OpenStreetMap-Daten enthaltenen Information bietet es sich an, die Transformation in eine Graphstruktur nicht in einem einzelnen Arbeitsschritt vorzunehmen. Beim hier beschriebenen Verfahren wird stattdessen zunächst eine Abstraktion durchgeführt, bevor der eigentliche Graphaufbau erfolgt.

4.2.1 Abstraktionsverfahren

Als Abstraktion wird eine bereits graphähnliche Darstellung gewählt, die sich aus drei Objekttypen zusammensetzt:

SegmentNode Punkt mit geographischer Koordinate analog zu OSM-Node.
SegmentNodes werden nur für diejenigen OSM-Nodes erzeugt, die tatsächlich ins

Wegenetz eingebunden sind. Dies wird sichergestellt, indem ihre Erstellung erst bei Bedarf während der Anlage der Segmente durchgeführt wird.

Segment Gerichtete Verbindung zwischen zwei SegmentNodes.
Die Erzegung geschieht auf der Grundlage der OSM-Ways. Dabei wird auf Basis einer Auswertung der Zugangsrechte (unter anderem mit den in 4.1 beschriebenen Verfahren) auf nicht nutzbare Teile des Wegenetzes ebenso verzichtet wie auf Ways, die nicht zum Wegenetz gehören – etwa Gebäudeumrisse.

Restriction Nicht nutzbare Verbindung zwischen Start- und Zielsegmenten.
Jede Restriction besitzt ein Startsegment (From-Segment), eine ungeordnete Menge von Zielsegmenten (To-Segmente) und eine ungeordnete Menge von Verbindungssegmenten (Via-Segmente). Die Existenz der Restriction macht es unmöglich, sich vom From-Segment zu einem der To-Segmente zu bewegen und dazwischen nur Via-Segmente zu nutzen. Restrictions werden zur Darstellung verschiedener Situationen verwendet:

- Abbiegeverbote (siehe 3.2.4): Die Definition von Restrictions ist unmittelbar von verbietenden Abbiegebeschränkungen inspiriert, hier lässt sich also in der Regel eine direkte Umsetzung durchführen. Werden als `via`-Member in der Abbiegebeschränkung Nodes verwendet. werden diejenigen Segmente zu Via-Segmenten, deren Nodes beide `via`-Member sind. Als From- bzw. To-Segmente werden diejenigen Wegstücke von `from`- bzw. `to`-Member-Ways gewählt, die direkt an `via`-Member anschließen.
- Abbiegeverpflichtungen (siehe ebenfalls 3.2.4): Eine Abbiegebeschränkung, die eine bestimmte Fortbewegung vorschreibt, lässt sich als Verbot aller Alternativen darstellen.
- Barrieren (siehe 3.2.3): Hindernisse auf Nodes lassen sich durch Restrictions darstellen, indem die Fortbewegung von jedem zum Node führenden Segment zu jedem vom Node wegführenden Segment ausgeschlossen wird.

Sämtliche später benötigte Information aus Tags wird bei der Erzeugung der Segmente zu den Segmenten zugeordneten Attributen abstrahiert (z. B. für die grafische Anzeige von Höchstgeschwindigkeiten).

Die Erzeugung der Segmente und SegmentNodes ist dabei effizient durchführbar: Die eigentliche Erzeugung verursacht lediglich linearen Zeitaufwand in Abhängigkeit von der Anzahl der zu erzeugenden Objekte. Hauptproblem für den Zeitaufwand (und Hindernis für eine komplett parallele Verarbeitung) ist die Notwendigkeit, Duplikate bei der Erzeugung der SegmentNodes zu verhindern, wenn ein Node in mehreren Ways enthalten ist. Hier ist eine assoziative Datenstruktur nötig, die den Zusammenhang zwischen den Nodes und den daraus erzeugten SegmentNodes abbildet. Auf dieser Datenstruktur ist eine Zahl von sowohl Einfüge- als auch Lesevorgängen in Größenordnung der SegmentNode-Anzahl erforderlich. Eine geeignete Datenstruktur (namentlich eine ausreichend groß angelegte Hashtabelle) beeinträchtigt das lineare Verhalten also nicht.

Wegen ihrer geringen Zahl spielt die Erzeugung der Restriktionen keine entscheidende Rolle bei der Verarbeitungsgeschwindigkeit. Auch hängt sie von zahlreichen Faktoren ab: Grundlage ist die Menge der Barrier-Nodes bzw. Abbiegebeschränkungen, die Anzahl und Größe der letztlich erzeugten Restriktionen wird jedoch zusätzlich durch die Anzahl der ein- und ausgehenden Segmente der Barrier-Nodes bzw. Relation-Member bestimmt. Auch wenn in den OpenStreetMap-Daten wohl selten mehr als eine einstellige Zahl von Segmenten beteiligt sein werden, gibt es hierfür theoretisch keine Beschränkungen, so dass – wie auch bei den folgenden Verarbeitungsschritten – die reale Performance wenig Bezug zur theoretischen Worst-Case-Situation hat. Es ließen sich allenfalls statistische Aussagen treffen, für die aber wegen der noch unvollständigen Abdeckung mit Barrieren und Abbiegebeschränkungen keine verlässliche Basis existiert.

4.2.2 Graphaufbau

Nach der Abstraktion bleibt die Aufgabe, einen gerichteten Graphen aus den verallgemeinerten Daten herzuleiten. Eine wichtige Entscheidung besteht in der Definition der Knoten und Kanten. Naheliegend wäre es, die SegmentNodes direkt in Graphknoten umzusetzen. Gegen diese Lösung spricht jedoch die Notwendigkeit, Restrictions in die Graphstruktur einzuarbeiten: Ein gemeinsamer SegmentNode muss noch keine Verbindung zwischen Segmenten herstellen, im resultierenden Graphen sollte dies für Graphkanten mit einem gemeinsamen Graphknoten sehr wohl gelten.

Gelöst werden kann dieses Problem, indem ein Graphknoten neben der Positionsinformation durch den zugrunde liegenden SegmentNode auch eine Richtungsinformation in Form eines Segments erhält. Bei dieser Graphknoten-Variante stellt ein Graphknoten den Zustand eines Verkehrsteilnehmers dar, sich

1. auf dem durch das Segment dargestellten Wegstück
2. unmittelbar vor demjenigen Ende, an dem der SegmentNode liegt

zu befinden.

Für einen gültigen Graphknoten muss der SegmentNode dabei notwendigerweise ein Ende des Segments bilden. Wegen 2. wird die volle Länge des Segments, einschließlich aller Attribute, als zu derjenigen Graphkante gehörig gesehen, die die beiden auf dem Segment beruhenden Graphknoten verbindet.

Auf Grundlage dieser Definition kann die Einarbeitung der Restrictions und die tatsächliche Graphkonstruktion erfolgen.

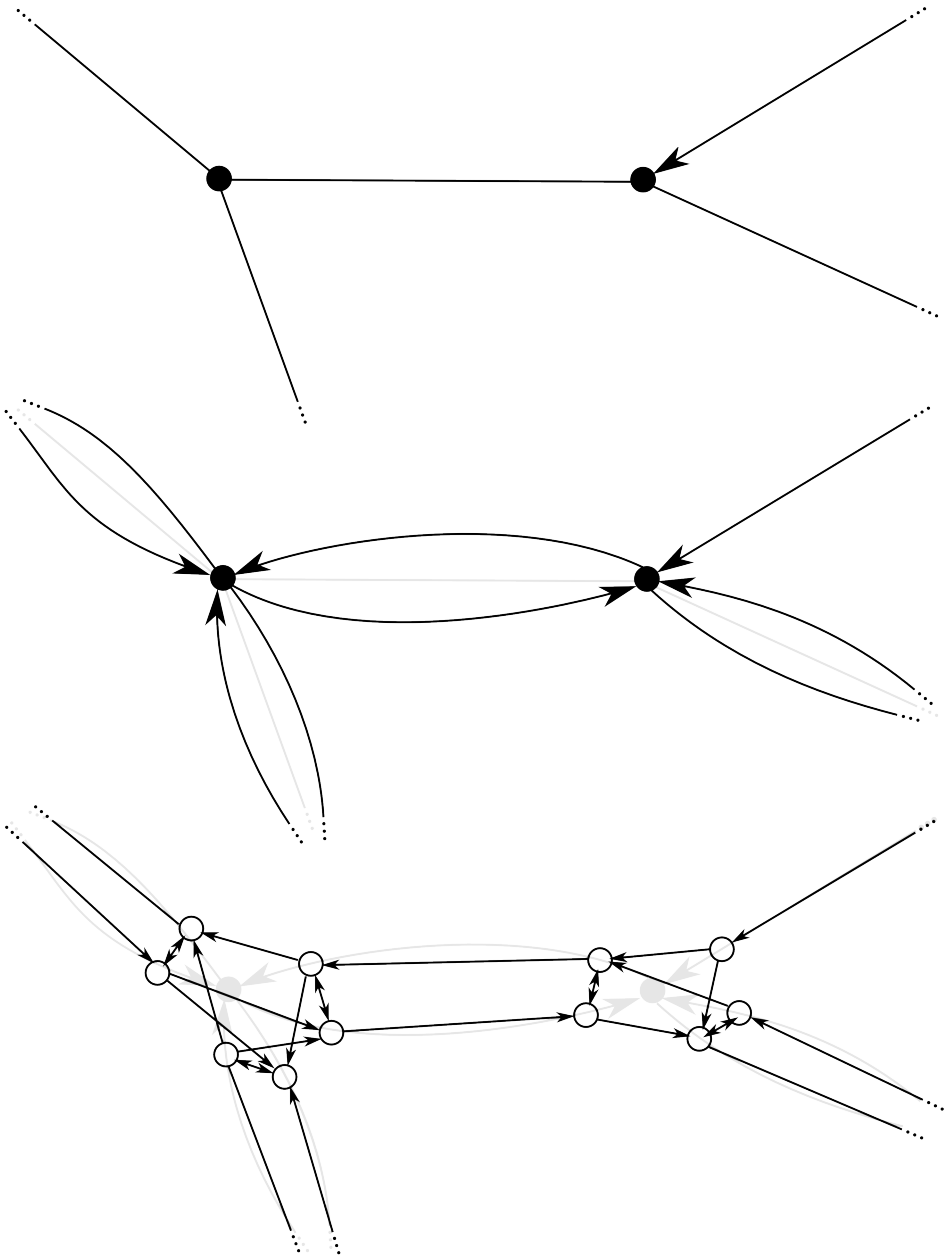


Abbildung 4.1: Beispiel für eine der Vorgehensweise von 4.2 entsprechende dreistufige Transformation: OpenStreetMap-Datenstruktur – Abstraktion – Graph (von oben nach unten)

Restriction-Integration: Gruppenbildung

Restrictions enthalten neben From-Segment und To-Segmenten auch Via-Segmente. Dies hat zur Folge, dass die Information, die einem Graphknoten zugrunde liegt, bei einer Position auf manchen Segmenten (nämlich den Via-Segmenten) alleine nicht ausreichen würde, um die weiteren Fortbewegungsmöglichkeiten eindeutig festzulegen. Zusätzlich ist hier auch die Information über vorher genutzte Segmente notwendig.

Dieses Problem kann umgangen werden, indem alle Teile einer Restriction gemeinsam betrachtet und nur am Rand des betrachteten Bereichs Graphknoten gebildet werden – also bevor oder nachdem die Restriction wirksam ist. Nur zur Bestimmung, ob Graphkanten zwischen den Knoten gebildet werden sollen, wird dieser Bereich dann analysiert. Zu diesem Zweck werden die Segmente und SegmentNodes in Gruppen eingeteilt, die gemeinsam zu analysieren sind. Es werden dabei zwei Arten von Gruppen gebildet:

Kreuzungsgruppen – enthalten Kreuzungs-SegmentNodes und die sie verbindenden Segmente. Als Kreuzungs-SegmentNodes werden solche SegmentNodes betrachtet, die mit mehr oder weniger als zwei anderen SegmentNodes über Segmente direkt verbunden oder Teil des Via-Bereichs einer Restriction sind. Eine Kreuzungsgruppe kann daher einen einzelnen SegmentNode oder mehrere über Restrictions verbundene SegmentNodes enthalten.

Verbindungsgruppen – enthalten zusammen alle Segmente. die nicht Teil der Kreuzungsgruppen sind. Teilen zwei Segmente einen SegmentNode, der nicht in einer Kreuzungsgruppe ist, sind sie in der gleichen Verbindungsgruppe.
Zur Bestimmung der Verbindungsgruppen wird zunächst eine Gruppe für jedes Segment angelegt, die keine weiteren Segmente enthält. Die Gruppen von direkt gegenläufigen Segmenten (ein Segment und seine Gegenrichtung) werden ohne weitere Prüfung vereinigt. Für jedes verbundene Paar von Segmenten wird außerdem geprüft, ob der sie verbindende SegmentNode in einer Kreuzungsgruppe ist. Ist dies nicht der Fall, werden die Gruppen der beiden Segmente vereinigt.

Graphknoten werden an den Übergangsstellen zwischen den Gruppen gebildet. Innerhalb der Gruppen entstehen keine Graphknoten. Bei den Kreuzungsgruppen dient dies der Vermeidung des Ausgangsproblems, dass im Gültigkeitsbereich von Restrictions die Fortbewegungsmöglichkeiten von Informationen zur Vorgeschichte abhängen würden. Bei Verbindungsgruppen werden so unnötige Graphknoten auf einer abzweigungsfreien Strecke vermieden.

Alternativen zur beschriebenen Gruppenbildung wären selbstverständlich denkbar, insbesondere die Erzeugung mehrerer Graphknoten pro Segment-SegmentNode-Paar, um nötige Informationen zur Vorgeschichte in den Knoten zu codieren. Der Gruppenbildung wurde wegen praktischer Vorzüge wie der Zusammenfassung von Verbindungssegmenten und der sich leicht ergebenden Parallelisierbarkeit (bei der gruppeninternen Analyse wird jede Gruppe für sich ausgewertet werden können) der Vorzug gegeben.

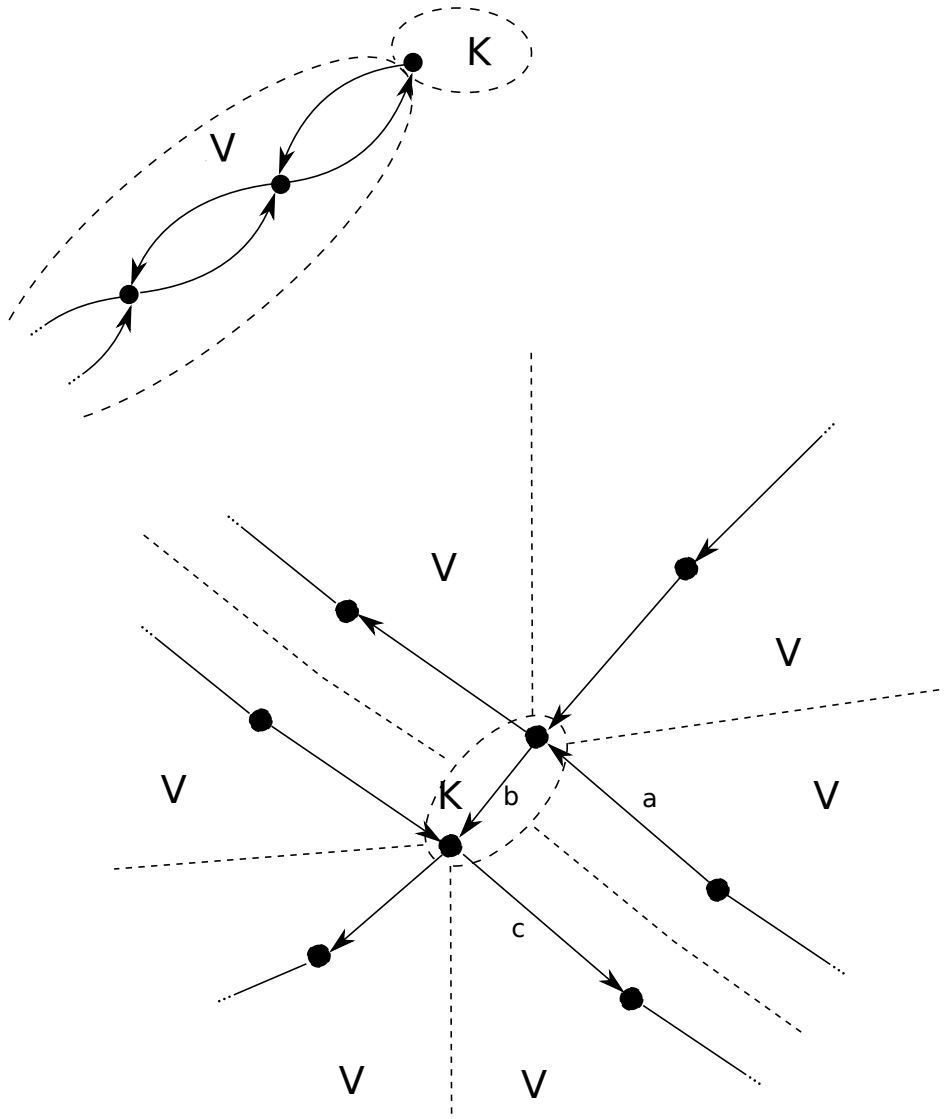


Abbildung 4.2: Charakteristische Beispiele für Kreuzungsgruppen (K) und Verbindungsgruppen (V). In einem Beispiel ist eine Restriction (from: a, to: b, via: c) enthalten, die ein Wendeverbot ausdrückt.

Restriction-Integration: Gruppenübergreifende Graphkonstruktion

Wie zuvor dargestellt, werden Graphknoten an Übergangsstellen zwischen Gruppen gebildet. Es handelt sich aufgrund der Definition der beiden Gruppentypen dabei stets um einen Übergang zwischen einer Kreuzungs- und einer Verbindungsgruppe:

- Direkte Verbindungen zwischen zwei Kreuzungsgruppen sind nicht möglich, da Segmente nur Teil einer Kreuzungsgruppe sind, wenn ihr auch beide SegmentNodes des Segments angehören. Verbindungen zwischen Nodes zweier verschiedener Kreuzungsgruppen sind also nur über Segmente möglich, die ihrerseits nicht Teil einer Kreuzungsgruppe sind.
- Direkte Verbindungen zwischen zwei Verbindungsgruppen sind nicht möglich, da ohne Kreuzungsgruppe aneinander grenzende Verbindungsgruppen im Laufe des Konstruktionsprozesses vereinigt werden.

Die tatsächliche Erstellung der Knoten erfolgt dabei auf Basis der ein- und ausgehendenden Segmente jeder einzelnen Kreuzungsgruppe: Jedes ein- oder ausgehende Segment (bestimmt dadurch, dass genau einer seiner SegmentNodes in der Kreuzungsgruppe ist) dient der Erstellung eines Graphknotens. Als SegmentNode, der für die Definition eines Graphknotens zusätzlich zum Segment erforderlich ist, dient derjenige SegmentNode des Segments, der sich innerhalb der Kreuzungsgruppe befindet.

Mit den ein- und ausgehenden Segmenten einer der Kreuzungsgruppen sind wegen der dargestellten Unmöglichkeit von Übergängen zwischen gleichartigen Gruppen alle Gruppenübergänge abgedeckt.

Restriction-Integration: Gruppeninterne Analyse

Welche Graphknoten durch Graphkanten miteinander zu verbinden sind, ist von den Gegebenheiten innerhalb der gebildeten Gruppen abhängig. Gelangt man innerhalb einer Kreuzungsgruppe über eine Segmentfolge von einem eingehenden zu einem ausgehenden Segment, ohne dabei eine Restriction zu verletzen, so ist von dem auf der Grundlage des eingehenden Segments und seines Ziel-SegmentNodes gebildeten Graphknoten eine (gerichtete) Kante zu demjenigen Graphknoten anzulegen, der durch das ausgehende Segment und seinen Quell-SegmentNode gebildet wurde. Analog kann für Verbindungsgruppen vorgegangen werden, wobei hier keine Restrictions zu beachten sind.

Um festzustellen, ob das ausgehende Segment vom eingehenden Segment aus erreichbar ist, wird eine Breitensuche eingesetzt. Diese kann allerdings nicht auf der Struktur von Segmenten und SegmentNodes erfolgen, weil die Restrictions so nicht zur Geltung kämen. Stattdessen wird ein Graph aus Zuständen eines Verkehrsteilnehmers gebildet.

Diese Zustände sind bestimmt durch

- einen SegmentNode, der die aktuelle Position darstellt
- die Menge der bereits besuchten SegmentNodes
- die Menge der aktiven Restrictions. Aktiv ist eine Restriction dann, wenn zuvor das From-Segment der Restriction passiert und der Via-Bereich der Restriction daraufhin noch nie verlassen wurde.

Damit die Information im Anschluss verfügbar ist, wird außerdem die Segmentfolge zum Erreichen des Zustands gespeichert, sie hat aber keine Auswirkungen auf die Berechnung.

Als Startzustand bei der Analyse einer Kreuzungsgruppe dient ein Zustand mit

- dem Ziel-SegmentNode des eingehenden Segments als aktueller Position
- dem Ziel-SegmentNode des eingehenden Segments als einzigem besuchten SegmentNode
- denjenigen Restrictions als aktiven Restrictions, die das eingehende Segment als From-Segment besitzen.

Akzeptable Endzustände für die Suche, die zur Erstellung einer Graphkante führen, sind solche mit

- dem Quell-SegmentNode des ausgehenden Segments als aktueller Position
- keiner aktiven Restriction, die das ausgehende Segment als To-Segment besitzt.

Von einem Zustand Z aus erreichbar sind diejenigen anderen Zustände, die

- eine aktuelle Position haben, die von der Position aus Z über ein Segment S der Gruppe erreichbar ist, das nicht das verbotene To-Segment einer in Z aktiven Restriction ist
- eine aktuelle Position haben, die nicht Element der besuchten Zustände von Z ist
- den neuen aktuellen SegmentNode sowie sämtliche besuchten SegmentNodes aus Z als besuchte SegmentNodes besitzen
- diejenigen Restrictions als aktive Restrictions enthalten, die
  - S als From-Segment enthalten oder
  - in Z aktiv waren und S als Via-Segment enthalten.

Durch die Sicherstellung, dass kein SegmentNode mehrfach besucht wird, terminiert der Algorithmus immer. Da in der Praxis wenige Kreuzungen mit mehr als 1–3 Knoten zu erwarten sind, ist das Laufzeitverhalten des Algorithmus weniger von Interesse.

5 Implementierung

Zentraler Bestandteil dieser Arbeit ist die tatsächliche Implementierung einer Software, die eine Graphvisualisierung durchführt und – wie zu Beginn motiviert (2.1.3) – zur Qualitätssicherung dienen kann. Nach der Beschreibung von allgemein anwendbaren Verfahren im vorigen Kapitel wird hier daher auf einige Designentscheidungen und Features bei der konkreten Umsetzung eingegangen. Dabei soll dieses Kapitel aber keineswegs die Klassen- und Codedokumentation ersetzen, vielmehr handelt es sich um eine grobe Beschreibung der Software, die unabhängig vom Code verständlich sein sollte.

Die Software stellt ein Plugin für JOSM dar, wird unter der Bezeichnung „Graphview“ veröffentlicht und im Folgenden auch so bezeichnet.

5.1 Technologie

Plattform für die Implementierung von Graphview ist Java 1.5 – diese Version wird von den offiziellen JOSM-Builds verwendet und ist ausdrücklich auch für Plugins empfohlen [how]. Zur Ausführung von Buildskripten wurde Apache Ant 1.7.1 eingesetzt.

Während der Implementierung wurden auch automatisch ausführbare Testfälle mit jUnit 4.3.1 erstellt.

Getestet wird das Plugin lediglich unter Windows XP, Ubuntu Linux und Gentoo Linux, es enthält aber keinen bekanntermaßen auf diese Betriebssysteme beschränkten Code. Zum Zeitpunkt der Fertigstellung des Plugins aktuelle JOSM-Version war 1607.

5.2 Benutzerschnittstelle und Features

Um den Zweck, bei der Fehlersuche und Qualitätssicherung im OpenStreetMap-Projekt verwendet werden zu können, möglichst gut zu erfüllen, ist eine leicht bedienbare Benutzerschnittstelle notwendig. Die Entwicklung als Plugin eines etablierten Editors verlangt zudem eine Anpassung an die Bedienphilosophie des Hauptprogramms. Daher soll hier zunächst ein Überblick über die grafische Oberfläche des JOSM-Editors gegeben werden.

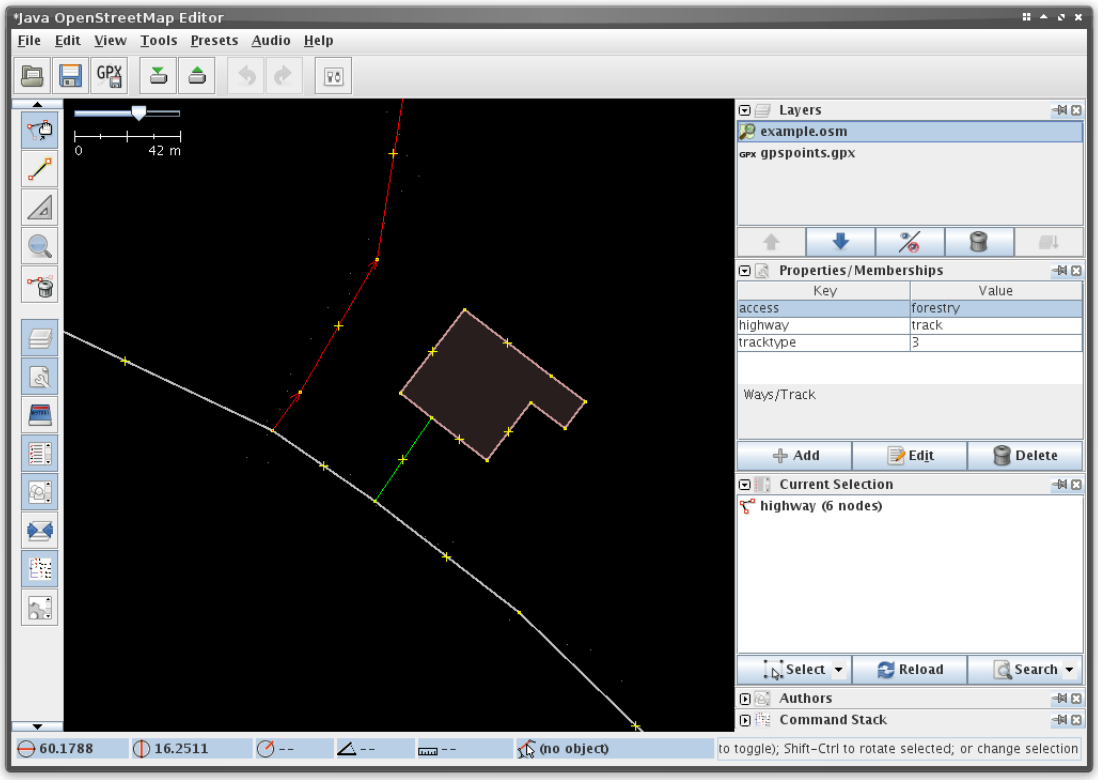


Abbildung 5.1: Standardansicht der JOSM-Benutzeroberfläche

5.2.1 Benutzerschnittstelle des JOSM-Editors

Die Hauptansicht des JOSM (siehe auch Screenshot 5.1) enthält verschiedene Arten von Bedienelementen. Im Einzelnen findet man:

Menü- und Werkzeugleiste – JOSM bietet Menü- und Werkzeugleiste in ihrer etablierten Form am oberen Rand des Fensters. Die Werkzeugleiste ist frei konfigurierbar.

Arbeitsbereich – Der zentrale Teil eines JOSM-Fensters wird vom Arbeitsbereich eingenommen. Hier wird eine zweidimensionale Ansicht der geladenen Kartendaten gezeigt. Es können mehrere unabhängige Ebenen von Daten übereinander liegen – neben zu bearbeitenden OpenStreetMap-Daten kann es sich hier auch um Luftbilder, GPS-Tracks, Overlays mit Warnugen des Validator-Plugins (2.2.2) oder dergleichen handeln. Viele der Zusatzebenen werden von Plugins bereitgestellt.

Zeichenwerkzeuge – Links des Arbeitsbereichs finden sich Werkzeuge, um Daten im Arbeitsbereich einzutragen und zu bearbeiten.

Arbeitsdialoge – Rechts des Arbeitsbereichs werden unterschiedliche Dialoge angezeigt,

sie lassen sich auch abdocken. Durch ihre Platzierung bieten sie sich für häufig genutzte Aufgaben an. Manche stellen einen Zugang zu den Daten der aktuell ausgewählten Elemente bereit, andere globale Informationen (etwa die Bearbeitungshistorie). Über einen dieser Dialoge lassen sich auch die Ebenen des Arbeitsbereichs anordnen, konfigurieren und löschen.

Dialog-Schalter – Unterhalb der Zeichenwerkzeuge links finden sich Schaltflächen, mit denen sich die vorgenannten Arbeitsdialoge ein- und ausblenden lassen.

Statusleiste – Am unteren Rand des Fensters befindet sich eine Statusleiste, die die aktuellen Koordinaten und einige andere Informationen anzeigt sowie Tipps zu möglichen Aktionen gibt.

5.2.2 Benutzerschnittstelle und Featureumfang des Graphview-Plugins

Für die eigentliche Aufgabe des Graphview-Plugins, die Visualisierung der Wegedaten, wird naheliegenderweise eine eigene Ebene im Arbeitsbereich verwendet. Dadurch kann die Graphdarstellung direkt ober- oder unterhalb der zugrundeliegenden Daten angezeigt werden, so dass die Orientierung ebenso wie das Auffinden von Fehlerquellen leicht fällt. Ohne weiteren Aufwand sind durch die Flexibilität des Ebenenansatzes so auch weitere Ansichten denkbar, etwa vor dem Hintergrund von Lufbildern oder gerenderten Karten. Da Ebenen auch ausgeblendet oder gelöscht werden können, kann der Nutzer auch bei installiertem Plugin selbst entscheiden, wann er die Grapheinblendung nutzen will.

Für zusätzliche Features müssen dem Nutzer allerdings noch weitere Optionen zugänglich gemacht werden. Dies geschieht auf zwei Wegen: Zum einen bietet Graphview einen eigenen Arbeitsdialog (Screenshot siehe 5.2), der die wichtigsten Einstellungen leicht zugänglich macht – ein Gebiet wird vermutlich oftmals nach verschiedenen Kriterien untersucht werden (verschiedene Attribute der Verkehrswege, Verkehrswege für verschiedene Fahrzeuge), so dass die entsprechenden Optionen leicht erreichbar sein sollten –, zum anderen existiert ein eigener Tab in den Einstellungen von JOSM, über den auch seltener genutzte Konfigurationsmöglichkeiten verfügbar sind.

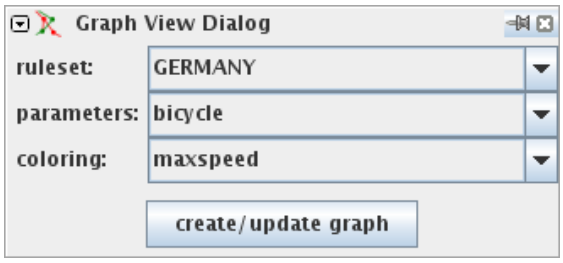


Abbildung 5.2: Arbeitsdialog des Graphview-Plugins mit Auswahlmöglichkeiten zu Verkehrsmittelkonfiguration, Ruleset und Einfärbung des Graphen

Der Arbeitsdialog bietet neben den wichtigen Einstellungen auch eine Schaltfläche an, um die Graphebene neu zu erzeugen – falls sie gelöscht wurde – oder zu aktualisieren. Letzteres ist nötig, wenn nach einer Änderung an den Daten der Graph neu berechnet werden soll.

Rulesets

Manche Aspekte der Auswertung sind abhängig von regionalen Gesetzen und Besonderheiten, diese Regelsätze werden hier als „Rulesets“ bezeichnet. Ideal wäre ein Rückgriff auf die aktuellen Grenzinformationen aus dem OpenStreetMap-Datenbestand. Da diese Daten jedoch in der Regel nicht Teil des vom Editor geladenen Bereichs sind, wird hier ein einfacherer Weg gewählt: Die Auswahl durch den Nutzer.

Allgemein dürfte diese Einstellung seltener geändert werden als die übrigen Optionen – die Verkehrsmittel- oder Farbauswahl. Wegen der zentralen Bedeutung dieser Auswahl für einen korrekten Graphen ist dennoch eine Umschaltung über den Graphview-Arbeitsdialog möglich. Nur im Einstellungsdialog hingegen wird dem Nutzer die Möglichkeit gegeben, statt auf die beschränkte Zahl mitgelieferter Rulesets auf eigene Dateien zurückzugreifen. Das Format dieser Dateien wird im Abschnitt 5.3.2 beschrieben.

Aussehen des Graphen

Die Farbgebung im Graphen wird bei Graphview genutzt, um Attribute der Verkehrswege hervorzuheben (etwa Höchstgeschwindigkeiten oder Gewichtsbeschränkungen); der Benutzer kann hier aus einer vorgegebenen Gruppe wählen. Daneben gibt es die Möglichkeit, Endknoten von Sackgassen farblich zu markieren, um den häufigen Fehler, dass in der Realität verbundene Straßen in den OpenStreetMap-Daten nur nahe beieinander liegende Endpunkte besitzen, leichter entdecken zu können. Auch eine einheitliche Einfärbung, also der Verzicht auf Zusatzinformationen, ist möglich. Damit ein Nutzer leicht mehrere Attribute nacheinander kontrollieren kann, ist die Auswahlmöglichkeit im Arbeitsdialog direkt verfügbar.

Daneben stehen zwei strukturell verschiedene Darstellungsvarianten des Graphen zur Verfügung. Bei der standardmäßig gewählten Form verlaufen die beiden Graphkanten, die sich aus einem in beide Richtungen nutzbaren OSM-Way ergeben, übereinander. Der Unterschied zwischen in beide Richtungen und nur in eine Richtung nutzbaren Verkehrswegen ist nur an den Pfeilspitzen am Ende der Graphkante zu erkennen.

Die alternative Darstellung versetzt jeden Graphknoten im rechten Winkel zum (gerichteten) Segment, auf dem er basiert. Dadurch sind die Möglichkeiten, sich an einer Kreuzung zu bewegen, deutlicher erkennbar – auf Kosten einer höheren Dichte von Linien, die mancherorts die Übersicht erschwert. Nur auf kreuzungsfreien Strecken verlaufen Graphkanten übereinander.

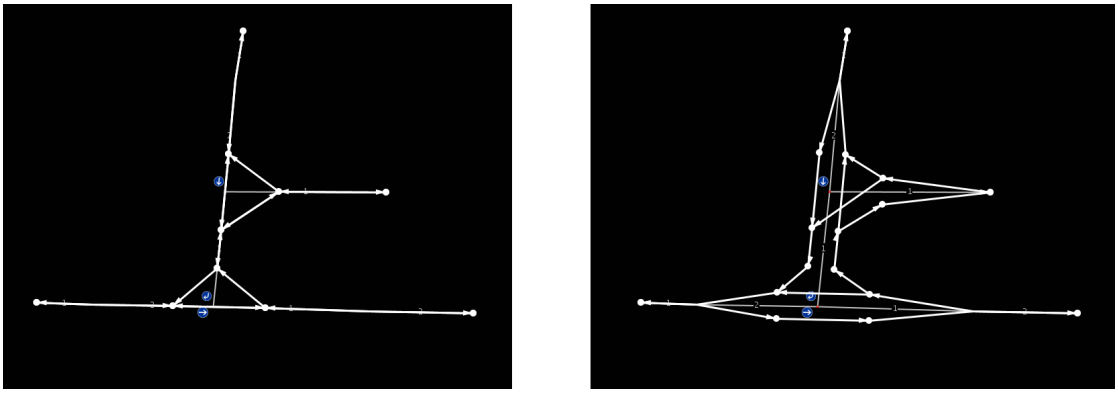


Abbildung 5.3: Vergleich zwischen überlappender (links) und versetzter Darstellung der Richtungen in der Graphdarstellung

Da je nach individuellen Vorlieben des Nutzers und besonders auch nach Datendichte im bearbeiteten Gebiet jede der Darstellungen Vorteile haben kann, wird dem Nutzer diese Wahl überlassen. Da hier nicht mit häufigen Einstellungsänderungen zu rechnen ist, sobald sich der Nutzer an eine Variante gewöhnt hat, ist die Option nur über den Einstellungsdialog zu erreichen.

Verkehrsmittel-Konfigurationen

Maßgeblich hängt die Gestalt des Graphen davon ab, welche Eigenschaften das verwendete Verkehrsmittel hat. Hier sind Angaben des Benutzers notwendig. Zu diesem Zweck existiert eine Eingabemaske, die die möglichen Eigenschaften abfragt und über den Graphview-Tab erreichbar ist. Bei der Angabe des Fahrzeugtyps wird auf Key-Namen des OpenStreetMap-Datenformats zurückgegriffen, um den Wertebereich nicht einzuschränken – die Erfindung neuer Typen, etwa für regionale Besonderheiten, ist schließlich jedem Nutzer möglich. Da der direkte Zugang zu Tags auch Teil des JOSM-Bedienkonzeptes ist, sollten Nutzer hier entsprechende Kenntnisse haben.

Die Zahl der Konfigurationsoptionen macht es allerdings wünschenswert, häufig genutzte Kombinationen nicht stets von neuem eintragen zu müssen. Graphview erlaubt es daher, Verkehrsmittelkonfigurationen unter einem Namen abzuspeichern. Zwischen den existierenden Konfigurationen kann über den Dialog neben der Arbeitsfläche gewechselt werden, das Anlegen, Bearbeiten oder Löschen ist aus Gründen des Platzbedarfs und der Übersicht nur in den Einstellungen zugänglich.

Um die Einstiegshürde für Nutzer zu senken, werden bei einer Erstinstallation des Plugins einige einfache Konfigurationen vorgegeben.

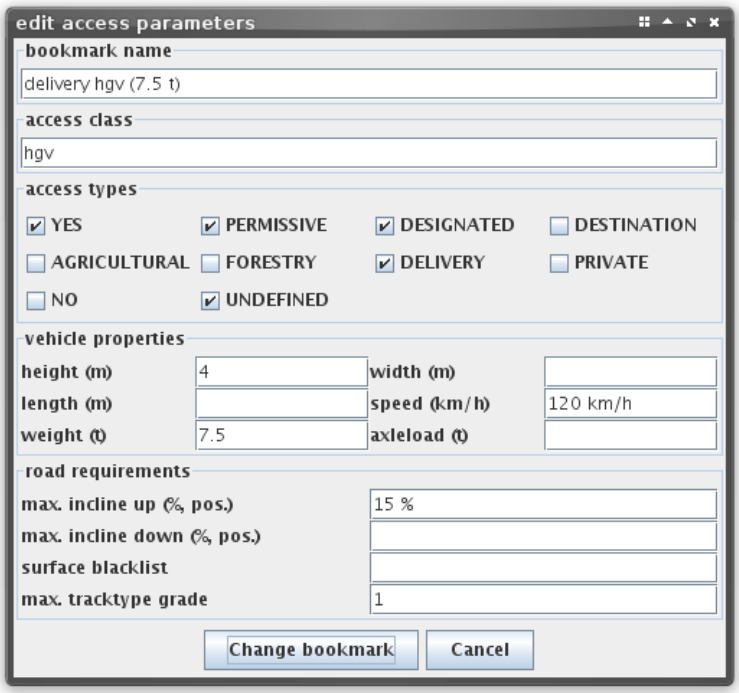


Abbildung 5.4: Dialog zur Bearbeitung einer Verkehrsmittel-Konfiguration

5.3 Dauerhaft gespeicherte Daten

5.3.1 Konfigurationsoptionen

Es lassen sich im Graphview-Plugin unterschiedliche Einstellungen vornehmen. Zur Datenablage wird die in JOSM verfügbare Funktionalität zum Sichern von Konfigurationsoptionen verwendet. Plugins haben hier ebenso wie JOSM selbst die Möglichkeit, Konfigurationsdaten als Schlüssel-Wert-Paare (beides Strings) abzulegen und zu lesen. Diese werden dann von JOSM selbsttätig in einer Textdatei gesichert, so dass sie bei zukünftigen Programmstarts wieder verfügbar sind.

Im Graphview-Plugin erfolgt die Verwaltung der Konfiguration über eine eigene Klasse, die den Abgleich geänderter Einstellungen mit JOSM sicherstellt und die Umwandlung von Objekten in Konfigurationsstrings (und umgekehrt) übernimmt. Da es sich bei der Konfigurationsdatenbank von JOSM um eine softwareweit nur einmal existierende Ressource handelt, ist die Verwaltungsklasse als Singleton [GHJV94] implementiert.

Eine Tabelle mit den von Graphview aktiv angelegten und genutzten Konfigurationseinträgen steht im Anhang A zur Verfügung.

5.3.2 Ruleset-Dateien

Um regionale Unterschiede in verkehrsrechtlichen Vorgaben und OpenStreetMap-Taggingstil erfassen zu können, sind in der Software XML-Dateien vorgesehen, die die Rulesets enthalten (vgl. 5.2.2). Das Rootelement `<accessRuleset>` einer solchen Datei enthält drei Sektionen:

`<classes>` – dient zur Darstellung der definierten Verkehrsmittelklassen und ihrer Inklusionsbeziehungen. Enthält Elemente `<class name="bezeichnung">`, wobei für `bezeichnung` der OpenStreetMap-Verkehrsmittelkey der Klasse einzusetzen ist. Die `class`-Elemente können ineinander verschachtelt sein, Kindelemente sind dabei als Teilklasse der Klasse des Elternelements aufzufassen.

`<basetags>` – listet alle Tags auf, die einen Way als Verkehrsweg markieren (vgl. 4.1). Jedes Tag wird in der Form `<tag k="key" v="value">` repräsentiert, Verschachtelung kommt nicht vor.

`<implications>` – enthält alle Implikationen. Eine einzelne Implikation ist ein Element `<implication>`, das ein `<condition>`- und ein `<implies>`-Element enthält – diese stellen die Bedingung für die Implikation bzw. die implizierten Tags dar. `<condition>` enthält eine Hierarchie aus den folgenden *Bedingungselementen* mit genau einem Wurzelelement:

`<tag k="key" v="value">` –
ist erfüllt, wenn ein Objekt das Tag `key = value` trägt

`<key k="key">` –
ist erfüllt, wenn ein Objekt ein Tag mit dem Key `key` trägt

`<or>` – enthält weitere Bedingungselemente;
ist erfüllt, wenn mindestens ein Kindelement erfüllt ist

`<and>` – enthält weitere Bedingungselemente;
ist erfüllt, wenn alle Kindelemente erfüllt sind

`<not>` – enthält ein weiteres Bedingungselement;
ist erfüllt, wenn dieses nicht erfüllt ist

Das `<implies>`-Element enthält lediglich eine Liste von `<tag>`-Elementen.

Bei der internen Abbildung der Implikationen kommt eine Composite-Struktur [GHJV94] zum Einsatz, die den Bedingungsbaum repräsentiert. Nachdem die Struktur beim Einlesen der Ruleset-Datei erstellt wurde, erfordert die Prüfung der Bedingung für ein bestimmtes Objekt damit nur einen einheitlichen Aufruf einer Methode des Wurzelelements des Bedingungsbaums.

Diese Lösung ist insgesamt als Provisorium gedacht. Letztlich wäre es wünschenswert, dass das OpenStreetMap-Projekt diese Informationen zentral in einem geeigneten Format zur Verfügung stellt.

5.4 Programmarchitektur

Da ein Verfahren zur Gewinnung von Wegegraphen aus OpenStreetMap-Daten für unterschiedliche Zwecke und in unterschiedlicher Software von Nutzen sein kann, sollen die anwendungsspezifischen Klassen sauber vom Rest des Codes getrennt werden. Dieses Entwurfziel zeigt sich in der Modularisierung der Software: Die allgemeine Funktionalität ist gänzlich im `core`-Modul enthalten, die anwendungsspezifischen Programmteile des Graphview-Plugins befinden sich im `plugin`-Modul. Klassen aus dem `core`-Modul verwenden keine Klassen oder Interfaces von außerhalb des `core`-Moduls.

Die Fähigkeiten des `plugin`-Moduls beschränken sich im Wesentlichen auf die Bereitstellung der Benutzerschnittstelle und die Verwaltung der vom Nutzer gesetzten Einstellungen. An der Grapherstellung hat es nur durch die Bereitstellung der Ausgangsdaten und der Parameter für die Konfiguration (etwa Fahrzeugeigenschaften) Anteil. In Grafik 5.5 wird das Zusammenspiel zwischen den Modulen veranschaulicht.

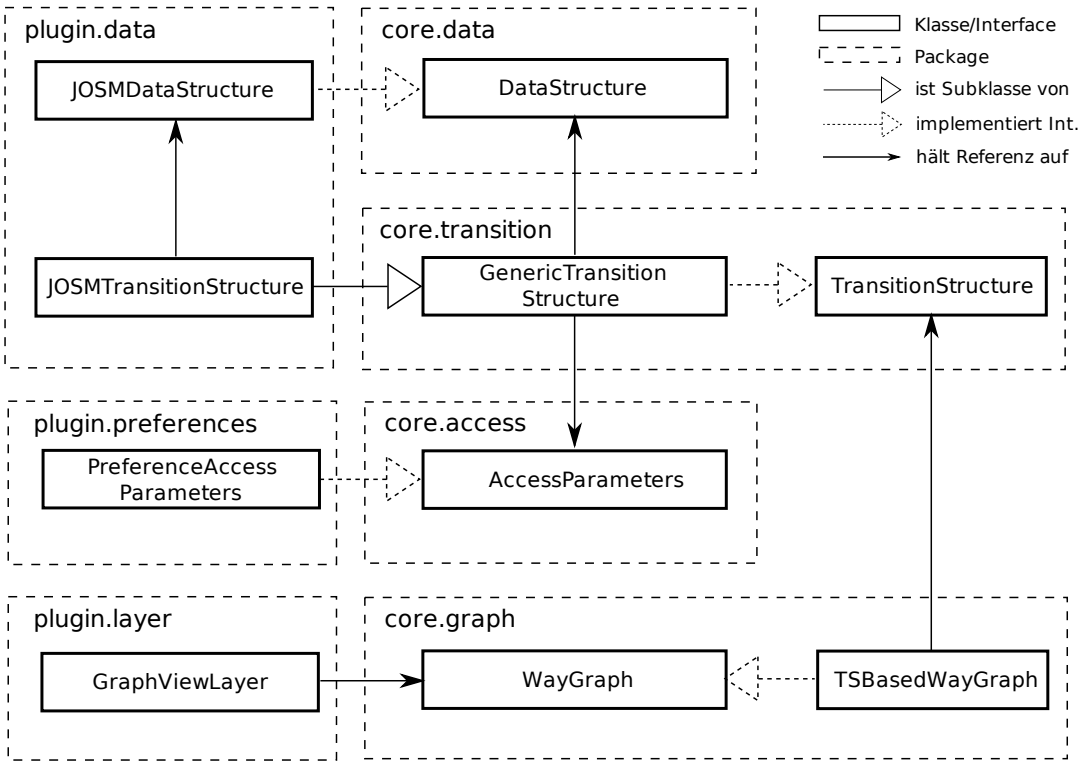


Abbildung 5.5: Vereinfachte Darstellung des Zusammenspiels zentraler Klassen und Interfaces

5.4.1 Modul core

Innerhalb des `core`-Moduls findet eine weitere Modularisierung statt: So ist das Modul `core.visualization` für Farbschemata und Knoten-Positionierung in bereits erstellten Graphen verantwortlich. Die Klassen dieses Moduls werden nicht für alle Anwendungsmöglichkeiten des Transformationsverfahrens benötigt, so dass der übrige `core` von der Visualisierungsfunktionalität unabhängig gestaltet ist.

Die Transformation und die Repräsentation des Resultats werden von den Modulen `core.transition` und `core.graph` übernommen, diese entsprechen den Arbeitsschritten Abstraktion (4.2.1) bzw. Graphaufbau (4.2.2). Da auf den Daten sequentiell die Abstraktion vor dem Graphaufbau erfolgt, hat `core.transition` keine Kenntnisse über das `core.graph`-Modul. In ähnlicher Weise greift das `core.transition`-Modul auf die Ausgangsdaten zu, die in Form von Klassen und Interfaces aus dem `core.data`-Modul vorliegen.

Damit das `transition`-Modul die Abstraktion von den Ausgangsdaten durchführen kann, benötigt es die Module `core.access` und `property` zur Auswertung von Zugangsrechten. `core.properties` kümmert sich dabei um die Repräsentation von Fahrzeug- und Straßeneigenschaften, `access` hingegen enthält die Intelligenz zur tatsächlichen Auswertung (unter anderem eine Implementierung der Auswertung der Verkehrsmittelhierarchie, vgl. 4.1) und zum Zugriff auf Ruleset-Dateien (5.3.2).

Das Modul `core.util` schließlich enthält ausschließlich Utility-Klassen und Interfaces, die an verschiedenen Stellen der Software passiv genutzt werden.

5.4.2 Modul plugin

Das Modul `plugin` enthält weniger Klassen und weniger Untermodule als `core`. Um dem Transformationsprozess Daten bereitzustellen, besitzt es mit `plugin.data` ein Modul, das Interfaces und Oberklassen aus `core.data` und `core.transition` implementiert bzw. erweitert.

Zur Verwaltung von Nutzereinstellungen und deren Synchronisation mit dem JOSM-Konfigurationssystem (vgl. 5.3.1) dienen die Klassen aus `plugin.preferences`. Die grafischen Dialoge zum Zugriff auf die Einstellungen befinden sich in `plugin.dialogs`. Die Darstellung der erzeugten Graphen schließlich ist Aufgabe von `plugin.layer`.

6 Fazit

OpenStreetMap bietet bereits jetzt eine beachtliche Menge an Daten, deren Auswertung jedoch nicht immer einfach ist. Dies liegt zum Teil an der gewollten Flexibilität im Tagging und dem noch unvollständigen und im Aufbau begriffenen Beschreibungsvokabular. Bei der Erstellung dieser Arbeit haben sich jedoch auch einige konkrete Verbesserungsvorschläge ergeben, die den Aufwand für die Datennutzung senken und auch die zügige Anpassung auswertender Software an die Änderungen im Datenformat und -bestand sicherstellen würden.

6.1 Schwachstellen im OpenStreetMap-Projekt

Konsistenz der Dokumentation

Informationen zu einem bestimmten Feature finden sich im OpenStreetMap-Wiki an verschiedenen Orten, üblicherweise mindestens auf der „Map Features“-Seite [wik: Map Features] und auf den individuellen Beschreibungsseiten der Keys bzw. Tags. Dabei existiert jedoch kein Verfahren zu einem automatischen Abgleich zwischen diesen Seiten. Um eine konsistente Dokumentation bereitstellen zu können, ist ein solcher Automatismus jedoch unverzichtbar.

Möglichkeiten zur Umsetzung gäbe es dabei verschiedene, so ließe sich die Anforderung durch eine individuelle Bot-Software erfüllen. Eine andere, bereits im Wiki diskutierte Idee ist die Verwendung einer Wiki-Software, die von Haus aus mit der nötigen Funktionalität ausgestattet ist [wik: Semantic MediaWiki] – es bietet sich das Semantic MediaWiki an, das die Anforderungen erfüllen kann. [smw]

Bereitstellung von Implikationen und Rulesets

Eine große Menge an wichtiger Information ist derzeit nur aus dem Wiki erhältlich. Insbesondere betrifft das Implikationen und Rulesets, wie sie für diese Arbeit in einem eigenen xml-Datenformat gespeichert werden (5.3.2). Teils werden zwar schon standardisierte Formen der Dokumentation im Wiki verwendet – Implikationen etwa werden in einem Parameter der MediaWiki-Templates „KeyDescription“ und „ValueDescription“ erfasst –, die Daten liegen jedoch nicht gesammelt vor, so dass das Auslesen ein nichttriviales, regelmäßiges Parsen des Wikis erfordern würde.

Da dererlei Information von unterschiedlichster Software benötigt wird, wäre die Bereitstellung eines auf dem Wiki basierenden, leicht maschinenlesbaren und regelmäßig aktualisierten Datensatzes sinnvoll. Manche Ansätze zur Sicherstellung der Konsistenz des Wikis (etwa das Semantic MediaWiki) können zugleich auch diesem Zweck dienen.

6.2 Ausblick

Mit den beschriebenen oder vergleichbaren Lösungen lassen sich die Probleme, die – neben der unzureichenden Datenabdeckung in manchen Regionen – das OpenStreetMap-Projekt noch an der vollen Entfaltung seines Potentials hindern, zweifellos bewältigen. In dieser Arbeit wurde ein Werkzeug dazu geschaffen, auch die Datenqualität auf einen Stand zu bringen, der die freie Geodatenbank für mögliche Anwendungen attraktiv macht. Zu erhoffen ist ebenfalls ein Beitrag des Plugins zu einer klareren Dokumentation der Erwartungen der Projektmitglieder an die Auswertung ihrer Daten durch eine Routingsoftware: Die Nutzer des Plugins können die Daten gewissermaßen mit den Augen eines Routers sehen – das lässt Unterschiede zwischen den Annahmen bei der Datenerfassung und der Interpretation durch die Software unmittelbar deutlich werden.

Damit diese Rolle erfüllt werden kann, wird es nötig sein, das Graphview-Plugin auch nach seiner Veröffentlichung zu weiter pflegen, es an neue Entwicklungen in den Datenstrukturen anzupassen und Abweichungen zur Dateninterpretation durch reale Navigationssoftware zu beheben. Die Freigabe des Sourcecodes wird ermöglichen, dieser Aufgabe gemeinsam mit der OpenStreetMap-Community nachzukommen.

A Konfigurationseinträge

Diese Tabelle verzeichnet die von Graphview aktiv gesetzten und genutzten Einträge in der Konfigurationsverwaltung des JOSM-Editors.

Schlüssel	**Wert**
`graphview.activeBookmark`	Name der aktiven Verkehrsmittel-Konfiguration; die Konfigurationen selbst stehen in `graphview.parameterBookmarks`.
`graphview.defaultEdgeColor`	Farbe der Graphkanten, wenn keine Attributwerte hervorgehoben werden sollen; da die meisten Nutzer mit der Vorgabe zufrieden sein dürften, ist die Option nicht über die grafische Oberfläche verfügbar. Farben werden als drei durch Kommata getrennte dezimale Ganzzahlen im Wertebereich 0–255 dargestellt.
`graphview.defaultNodeColor`	Knotenfarbe, analog zu `defaultEdgeColor`
`graphview.parameterBookmarks`	Alle verfügbaren Verkehrsmittel-Konfigurationen samt ihrer Namen. Die einzelnen Konfigurationen werden durch Pipe-Zeichen getrennt, Teile der Konfiguration durch Semikolon: Zu Beginn jeder Konfiguration steht der Name, dann die Verkehrsmittelklasse. Von „types=“ bzw. „properties=“ eingeleitet folgt je eine von geschweiften Klammern eingefasste und kommaseparierte Liste für zulässige Nutzungsrechte bzw. Properties als „Typ=Wert“.
`graphview.rulesetFile`	Pfad der aktuellen externen Ruleset-Datei
`graphview.rulesetFolder`	Pfad des Ordners für externe Ruleset-Dateien
`graphview.rulesetResource`	Name der aktuellen internen Ruleset-Datei
`graphview.separateDirections`	Bestimmt, ob die alternative Graphdarstellung wie unter 5.2.2 beschrieben zum Einsatz kommt. Wahrheitswert.
`graphview.useInternalRulesets`	Entscheidet, ob interne Rulesets verwendet werden. Wahrheitswert.

B Installation und Start des Plugins

Dieser Anhang beschreibt die Schritte, die nötig sind, um das im Rahmen der Arbeit erstellte Graphview-Plugin für JOSM zu installieren und zu testen.

Voraussetzungen

Der Rechner, auf dem Graphview verwendet werden soll, muss über eine lauffähige Java-Installation verfügen, die kompatibel mit Sun Java 1.5 ist. Auf dem Rechner muss außerdem JOSM (getestet wurde Graphview mit Version 1607) vorhanden sein.

Manuelle Installation

Da Graphview zum Zeitpunkt der Abgabe der Arbeit noch nicht im JOSM-Plugin-Repository vorhanden ist, sind zusätzliche Schritte nötig, um eine Installation vorzunehmen. Wenn eine selbstübersetzte Version eingesetzt werden soll, ist diese Vorgehensweise unabhängig von der Verfügbarkeit im Repository erforderlich.

Das Graphview-Plugin liegt als einzelne .jar-Datei vor, die im Plugin-Ordner von JOSM platziert werden muss. Abhängig vom Betriebssystem befindet sich der Plugin-Ordner an unterschiedlichen Orten:

- Windows Vista:
  `C:\Users\<user>\AppData\Roaming\JOSM\plugins`
- Windows (vor Vista):
  `C:\Documents and Settings\<user>\Application Data\JOSM\plugins`
- Windows (vor Vista), deutsch:
  `C:\Dokumente und Einstellungen\<user>\Anwendungsdaten\JOSM\plugins`
- Linux:
  `/home/<user>/.josm/plugins`

Möglicherweise muss der Ordner „plugins“ erst angelegt werden. Sofern JOSM bereits einmal gestartet wurde, sollte aber zumindest der Ordner „JOSM“ (bzw. „.josm“) schon existieren.

Sobald das Plugin im passenden Ordner vorhanden ist, kann die Installation abgeschlossen werden. Hierzu wird JOSM gestartet, der Einstellungsdialog geöffnet (Default-Shortcut F12) und der Plugin-Reiter aufgerufen. Dort sollte ein Eintrag für Graphview vorhanden sein, dessen Checkbox nun aktiviert wird. Der folgende Vorschlag von JOSM, vom Server eine aktuellere Version zu beziehen, wird abgebrochen; der Einstellungsdialog daraufhin mit „OK“ verlassen. Vor der Verwendung ist ein Neustart von JOSM notwendig.

Aktualisierte Hinweise zur manuellen Installation von Plugins (auch für weitere Betriebssysteme) finden sich auf `http://josm.openstreetmap.de/wiki/Plugins` im JOSM-Wiki.

Verwendung

Ist das Plugin vorbereitet, kann es nach einem erneuten Start von JOSM genutzt werden. Falls es bei größeren Karten zu Problemen mit fehlendem Arbeitsspeicher kommt, sollte ein Befehl wie der folgende zum Start von JOSM verwendet werden:
`java -Xmx1024M -jar josm.jar`

Es bietet sich an, in JOSM zunächst eine Datei zu öffnen und den Graphview-Dialog über das Graphview-Icon links des Arbeitsbereichs zu aktivieren. Dort lässt sich eine geeignete Ruleset-Datei auswählen und die Grapherstellung über den Button „create/update graph“ veranlassen.

Weitere Optionen für Graphview finden sich in einem Reiter im Einstellungsdialog. Über den Ebenen-Dialog rechts des Arbeitsbereichs lässt sich die Graphebene gegenüber anderen Ebenen verschieben, ausblenden oder löschen.

Bezugsquellen

Software und Sourcecode werden der gedruckten Bachelorarbeit auf einer Daten-CD beigelegt. Alternativ werden die Inhalte unter `http://tobias-knerr.de/bachelorarbeit/` veröffentlicht.

Aktuelle Versionen von JOSM sind unter `http://josm.openstreetmap.de` zu finden.

C Literaturverzeichnis

[com] *Computing access restrictions.* http://wiki.openstreetmap.org/index.php?title=Computing_access_restrictions&oldid=230713 [Stand 2009-02-21].

[GHJV94] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Design Patterns – elements of reusable object-oriented software.* Addison-Wesley, 1994.

[how] *Developing Plugins.* http://josm.openstreetmap.de/wiki/DevelopingPlugins [Stand 2009-05-30].

[Ray01] Raymond, Eric S.: *The Cathedral and the Bazaar.* O'Reilly Media, 2. Auflage, 2001.

[RT09] Ramm, Frederik und Jochen Topf: *OpenStreetMap.* Lehmanns Media, 2. Auflage, 2009.

[sta] *OpenStreetMap Statistics.* http://www.openstreetmap.org/stats/data_stats.html [Stand 2009-05-30].

[tag] *OpenStreetMap Tagwatch.* http://tagwatch.stoecker.eu/ [Stand 2009-05-30].

[wik] *OpenStreetMap Wiki.* http://wiki.openstreetmap.org [Stand 2009-05-30].

D Referenzierte Software und Webservices

[jos] *JOSM – Java OpenStreetMap Editor*
`http://josm.openstreetmap.de`
[Stand 2009-06-01]

[max] *OSM Maxspeed Map*
`http://maxspeed.osm.lab.rfc822.org`
[Stand 2009-06-01]

[ors] *OpenRouteService*
`http://www.openrouteservice.org`
[Stand 2009-06-01]

[osb] *OpenStreetBugs*
`http://openstreetbugs.appspot.com`
[Stand 2009-06-01]

[rou] *Routing-Plugin JOSM*
`http://wiki.openstreetmap.org/wiki/JOSM/Plugins/Routing`
[Stand 2009-06-01]

[smw] *Semantic MediaWiki*
`http://semantic-mediawiki.org`
[Stand 2009-06-01]

[tra] *Traveling Salesman*
`http://sourceforge.net/projects/travelingsales`
[Stand 2009-06-01]

[val] *Validator-Plugin JOSM*
`http://wiki.openstreetmap.org/wiki/JOSM/Plugins/Validator`
[Stand 2009-06-01]

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle wörtlich oder sinngemäß übernommenen Ausführungen wurden als solche gekennzeichnet. Weiterhin erkläre ich, dass ich diese Arbeit in gleicher oder ähnlicher Form nicht bereits einer anderen Prüfungsbehörde vorgelegt habe.

Passau, den 2. Juni 2009

(Tobias Knerr)