Hochschule Karlsruhe – Technik und Wirtschaft

Diplomarbeit

Algorithmen zur automatisierten Generalisierung durch Zusammenfassung von Linienzügen in OpenStreetMap

für konkrete Spezialfälle

Arne Johannessen

2. März 2018

betreut durch
Prof. Dr. rer. nat. Detlef Günther-Diringer
und
Dipl.-Wi.-Ing. Frederik Ramm

GEOFABRIK
neogeografie // software // beratung

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Themenblatt der Diplomarbeit

für Arne Johannessen

Thema

Algorithmen zur automatisierten Generalisierung durch Zusammenfassung von Linienzügen in OpenStreetMap für konkrete Spezialfälle

Ausgangslage

OpenStreetMap (OSM) ist ein VGI-Projekt *(Volunteered Geographic Information)* zur Herstellung einer Geodatenbank mit weltweiter Abdeckung. Vorgegeben durch die Struktur der Datenbank sind lediglich die primitiven Datentypen Punkt *(node)* und Linie *(way)* sowie Relationen. Alle weiteren Eigenschaften werden durch Key-Value-Pairs festgelegt *(tagging)*. Eine Ontologie existiert nicht; alle Mitwirkenden entscheiden jeweils für sich selbst, wie eine bestimmte Situation der Realität in OSM modelliert und getaggt wird. Mit der Zeit etablieren sich dabei jedoch von selbst gewisse *de facto*–Standards.

Als Folge der fehlenden Ontologie erfassen die Mitwirkenden sehr viele Details, die in anderen Geodatenbanken oft nicht zu finden sind. So könnten zum Beispiel für Teilstücke einer Straße *tags* für Tempolimits, Überholverbote, Fahrspurenanzahl, Oberflächenmaterial sowie -Qualität, Baujahr und mehr eingetragen worden sein. Jede solche Änderung eines *tag* entlang der Straße führt zum Beginn eines neuen *way* in der OSM-Datenstruktur. Diese Fragmentierung sowie der hohe Detaillierungsgrad führen zu großen Datenmengen in der weltweiten OSM-Datenbank.

Angesichts dessen wären automatisiert abgeleitete, generalisierte Datenbanken für kleinere Maßstäbe zur Weiterverwendung wünschenswert, existieren jedoch bisher nicht. Entsprechend sind auch die aus OSM-Daten hergestellten Karten in aller Regel nicht kartographisch generalisiert: Nahe beieinander liegende Objekte überdecken einander scheinbar wahllos, Mindestgrößen und notwendige Formvereinfachungen werden von den automatischen Renderern ignoriert.

Erschwert wird die Generalisierung von OSM-Daten unter anderem durch den hohen Grad der Fragmentierung von Linienzügen. Das Verknüpfen solcher zusammengehörenden, einzelnen *ways* durch Relationen in der Datenbank ist technisch möglich, wird aber von den Mitwirkenden aus unterschiedlichen Gründen nur sehr selten durchgeführt. Eine automatisierte Generalisierung durch Formvereinfachung (zur Reduktion der *node*-Anzahl) bedingt daher, dass die zusammengehörenden *ways* dabei als solche identifiziert werden. Gleiches gilt für die Generalisierung durch Zusammenfassung oder Verdrängung parallel verlaufender Linienzüge, beispielsweise Straßen mit begleitendem Radweg oder mehrgleisigen Bahnstrecken.

Zu bearbeitende Punkte

- Untersuchung bereits erforschter Ansätze zur automatisierten Identifikation zusammengehörender Linienzüge auf Eignung für Geodaten von OpenStreetMap
- Spezifikation der Spezialfälle, anhand derer das Problem im Weiteren zu untersuchen ist und für die konkrete Algorithmen zu entwickeln sind
- Entwicklung praxistauglicher Algorithmen zur Identifikation und zur Generalisierung durch Zusammenfassung kontinuierlicher, parallel verlaufender OSM-Linienzüge und, soweit dazu nötig, zur Verknüpfung der Fragmente *(ways)* dieser kontinuierlichen Linienzüge

Bearbeitungszeit:	4 Monate
Ausgabe der Diplomarbeit:	27. Dezember 2012
Abgabe der Diplomarbeit:	27. April 2013

Erstprüfer Prof. Dr. rer. nat. Detlef Günther-Diringer (betreuender Hochschulprofessor)	Zweitprüfer Dipl.-Wi.-Ing. Frederik Ramm (betreuender Geschäftsführer der Geofabrik)

Vorbemerkung

Wie in der Aufgabenstellung auf der vorangehenden Seite ersichtlich, war die vorliegende Diplomarbeit bereits Ende 2012 ausgegeben worden. Sie sollte ursprünglich im April 2013 fertiggestellt werden. Aufgrund sehr langer krankheitsbedingter deutlicher Einschränkungen der Arbeitsfähigkeit des Verfassers konnte sie – mit Genehmigung des zuständigen Prüfungsausschusses – erst bedeutend später abgeschlossen werden.

Die Arbeit vollständig auf den heutigen Stand zu aktualisieren, hätte eine Neubearbeitung wesentlicher Teile erfordert. Die damit einhergehende zusätzliche Zeitverzögerung hätte die Vergleichbarkeit mit anderen Abschlussarbeiten weiter verschlechtert. Um dies zu vermeiden, entspricht diese Diplomarbeit in den wesentlichen Punkten – mit dem Einverständnis der Betreuer – dem Stand der Wissenschaft im Jahr 2013.

Einige Kartenausschnitte sind neueren Datums; dies ist jeweils anhand der Quellenangabe ersichtlich. Weitere neuere Aspekte sind – soweit möglich – entweder in Fußnoten ergänzt oder abschließend in Abschnitt 7.3 „Andere Ansätze und neuere Forschung“ (Seite 74) berücksichtigt.

Arne Johannessen, im Februar 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Interaktive Kartendienste wie Google Maps haben dazu beigetragen, Geoinformationen allgegenwärtig zu machen. Jeder Mensch kann mit ihrer Hilfe im World Wide Web in Sekundenschnelle hochpräzise Karten von beinahe jedem Ort der Erde einsehen.

Die Qualität der kartographischen Darstellung leidet dabei jedoch in vielen Fällen. So ist es unmöglich, die Masse an Informationen einer für solche Kartendienste genutzten weltweiten Geodatenbank manuell kartographisch zu generalisieren.

Dies gilt um so mehr für das Projekt OpenStreetMap (OSM), das seine Geodaten vornehmlich aus Beiträgen Freiwilliger bezieht. Vielen von ihnen fehlt eine kartographische Ausbildung. Auch deshalb muss eine Generalisierung nach kartographischen Kriterien für OpenStreetMap-Webkarten mit weltweiter Abdeckung automatisiert ablaufen.

Bislang findet eine Generalisierung von Linienzügen in OpenStreetMap-Karten nur in geringstem Umfang statt. Ansätze für automatisierte Verfahren existieren zwar bereits, sie erfordern jedoch meist eine manuelle Nachbearbeitung und sind damit für OpenStreetMap ungeeignet.

Ziel der vorliegenden Arbeit ist die Entwicklung von Algorithmen zur Generalisierung durch Zusammenfassung paralleler OpenStreetMap-Linienzüge (etwa zwei Richtungsfahrbahnen einer Autobahn) auf eine gemeinsame Mittellinie. Diese Vereinfachung der Geodaten soll die Qualität der kartographischen Darstellung verbessern und die eventuelle Weiterverarbeitung erleichtern.

Die Richtungsfahrbahnen einer Autobahn sind nur eines von vielen Beispielen für parallele Linienzüge in der OpenStreetMap-Datenbank. Die Vielfalt der möglichen Situationen verbietet es, im Zeitrahmen einer Diplomarbeit eine Lösung mit allgemeiner Anwendbarkeit anzustreben.

Eine praxistaugliche Lösung für wenigstens eine solche Situation könnte jedoch bereits unmittelbare Verbesserungen bringen. Diese Arbeit beschränkt sich daher auf die Untersuchung eines einzelnen, noch festzulegenden Spezialfalls. Dessen Lösung ließe sich dann möglicherweise auf andere Situationen übertragen.

Die zu entwickelnden Algorithmen sollen zum Nachweis ihrer Funktionsfähigkeit ausführbar implementiert werden. Diese Implementierung könnte zugleich ein möglicher Ansatzpunkt für einen eventuellen späteren praktischen Einsatz des Verfahrens sein. Um dies nicht zu erschweren, soll die so entstehende Software unter einer freizügigen Open-Source-Lizenz verfügbar sein.

Maschinell ausführbare Beschreibungen sind notwendigerweise präzise bis ins letzte Detail, weswegen Software-Quelltext oft nicht leicht zu lesen ist. Deshalb sollen die entwickelten Algorithmen in dieser Arbeit dem leichteren Verständnis zuliebe abstrakt beschrieben werden. Weil dadurch implementierungsspezifische Details entfallen können, wird gleichzeitig die Weiterverwendung der entwickelten Ideen unabhängig von der Software ermöglicht.

Der Verzicht auf Detailgenauigkeit darf an dieser Stelle nicht mit Unbestimmtheit verwechselt werden: „Being abstract is something profoundly different from being vague: by abstraction [...] one creates a new semantic level on which one can again be absolutely precise.“ [Dÿk78, 1]

In dieser Arbeit umreißt Kapitel 2 das Projekt OpenStreetMap und gibt zahlreiche Kartenbeispiele dazu. Diese vermitteln einen Eindruck davon, wie Liniengeneralisierung bisher angewandt wird und wo ihr Fehlen das Kartenbild verschlechtert. Anschließend werden existierende Verfahren zur automatisierten Zusammenfassung von Linienzügen betrachtet und bewertet.

Kapitel 3 legt den Spezialfall fest, anhand dessen das Problem im Weiteren untersucht werden soll. Neben einer illustrierten Erläuterung des Funktionsprinzips enthält Kapitel 4 die formale und abstrakte Beschreibung der entwickelten Algorithmen; Anhang A erklärt die dabei verwendeten Zeichen und Abkürzungen. Kapitel 5 erläutert wesentliche Entscheidungen in Bezug auf die Implementierung in Software und beschreibt unerwartete Erkenntnisse.

Anschließend diskutiert Kapitel 6 die Tauglichkeit der entwickelten Methode anhand einiger Beispiele. Dabei wird auch die Anwendbarkeit auf andere als den zuvor festgelegten Spezialfall untersucht. Schließlich gibt Kapitel 7 eine Gesamtbeurteilung über das Ergebnis dieser Arbeit ab und diskutiert, welche Ansätze zur Weiterentwicklung am lohnensten erscheinen. Kapitel 8 fasst Kontext, Vorgehen und Ergebnisse der Arbeit kurz zusammen.

2 Analyse der Ausgangslage

2.1 OpenStreetMap: Alles für Alle

Ziel des Projekts OpenStreetMap (OSM) ist die Erstellung einer Geodatenbank für eine freie Weltkarte. [cf. Top09, 47] Zehntausende Freiwillige verwenden z. B. in ihrer Freizeit preisgünstige tragbare GPS-Empfänger oder dafür freigegebene Luftbilder, um Geodaten zu erfassen (*volunteered geographic information,* VGI). [cf. NZ12, 148–149, 154] Die so erstellte Geodatenbank kann unter einer freien Lizenz[1] zu beliebigen Zwecken weiterverwendet werden. [cf. SP11, 119–120]

Das von OpenStreetMap verwendete Datenmodell ist sehr einfach gehalten: Es kennt nur Punkte *(nodes),* Linienzüge *(ways)* und Relationen, die alle jeweils über Attribute *(tags)* in Form von *key-value-pairs*[2] verfügen können. Dabei definieren *ways* eine Liste von *nodes* als ihre Stützpunkte, während Relationen Beziehungen zwischen mehreren *ways, nodes* oder anderen Relationen ausdrücken. Flächen werden nur über *tags* modelliert. [cf. RT09, 49–55]

Der Name „OpenStreetMap“ mag irreführend sein, denn es handelt sich dabei nicht um eine Straßenkarte, sondern vielmehr um eine maßstabslose Datenbank mit teils sehr detaillierten Inhalten nahezu beliebiger Themen. [cf. SP11, 115–116] Alle OpenStreetMap-Attribute sind Freitext-Angaben ohne starre Klassifizierungsschemata oder Kartierschlüssel. Syntax und Semantik der einzelnen *tags* werden seit der Gründung von OpenStreetMap fortlaufend von den Beitragenden diskutiert, gestützt auf bisherige Erfahrungen. Die Diskussion findet vornehmlich im Internet statt; Ergebnisse werden öffentlich dokumentiert und in Software wie z. B. Renderern implementiert. Wer neue Inhalte erfasst, orientiert sich oft an den bisher in der Datenbank üblichen oder im Web dokumentierten Konventionen. [cf. RT09, 59–62]

Dieses Vorgehen hat zur Folge, dass die zu OSM Beitragenden genau das erfassen, was sie persönlich für interessant oder wichtig halten. [cf. Top09, 47–48] Jeder der Beitragenden entscheidet somit die Kriterien für die Erfassungsgeneralisierung jeweils für sich selbst. OpenStreetMap-Daten sind dadurch zwar inhomogen, verfügen aber oft über einen sehr hohen Detailreichtum. [cf. Sch12] So könnten zum Beispiel für Teilstücke einer Straße *tags* für Tempolimits, Überholverbote, Fahrspurenanzahl, Oberflächenmaterial sowie -qualität, Baujahr und mehr eingetragen worden sein. [cf. SP11, 115]

Veränderungen an den Attributen im Verlauf eines *ways* führen im OpenStreetMap-Datenmodell zwangsläufig zu einer Fragmentierung in mehrere aufeinander folgende *ways,* jeweils mit unterschiedlichen *tags.* Dies erhöht die Datenmenge und erschwert

1 Seit 12. September 2012 verwendet OSM die Open Database License (ODbL). [cf. osm12a]

2 OpenStreetMap-*tags* sind Kombinationen aus zwei Zeichenketten: einem Attributschlüssel und einem Wert. [cf. RT09, 52] Sie werden meist als *key = value* notiert (z. B. *name = Geofabrik*).

die Weiterverarbeitung. Eine Verknüpfung dieser Fragmente – im aktuellen OpenStreetMap-Datenmodell durch Relationen – kann in manchem Kontext hilfreich sein, ist jedoch auch eine Quelle neuer Probleme. [cf. RT09, 56–57] Es wäre außerdem möglich, einander parallele *ways* in gleicher Weise über Relationen zu verknüpfen, um das Zusammenfassen zu erleichtern.

Für Organisationen mit homogeneren Datenbeständen und Erfassungsweisen wie etwa Vermessungsämter wäre eine solche Lösung naheliegend. So erfasst beispielsweise die deutsche Landesvermessung die Straßenachse bereits in den Basisdaten mitsamt Verknüpfungen zu allen Fahrbahnen und Fragmenten. [cf. adv08, 48–50] Einige Generalisierungsvorgänge werden so deutlich vereinfacht. [cf. UD06, 193]

In OpenStreetMap ist dies jedoch bisher nicht üblich. Tatsächlich scheinen Relationen vergleichsweise wenig Beachtung zu finden.[3] Möglicherweise hängt dies damit zusammen, dass Relationen von OpenStreetMap-Software noch nicht optimal unterstützt werden. [cf. RT09, 83]

Das oben beschriebene Fehlen interner Strukturen unter den OpenStreetMap-Mitwirkenden (der Community) macht es aufwändig, die Erfassung und Kodierung der Geodaten zu verändern. [cf. Sch09, 84–85] Änderungsvorschläge müssen zuerst von der Community akzeptiert und anschließend *großräumig* und *korrekt* in der Datenbank Anwendung finden. Die Erfahrung zeigt, dass dies ein langwieriger Prozess sein kann. Das macht eine automatisierte Ableitung der nötigen Informationen aus den bereits existierenden Geodaten interessant.

2.2 Kartenherstellung mit OpenStreetMap

Wohl am bekanntesten ist OpenStreetMap für die auf der Projekt-Homepage `osm.org` zugängliche Web-Karte (Abbildung 1). Sie besteht aus quadratischen Kacheln *(tiles)* von $256 \times 256\,\text{px}$, die dort in 19 verschiedenen Maßstäben[4] von etwa $1:2000$ bis $1:500$ Mio. angeboten werden. [cf. RT09, 154–155] Eine interaktive Oberfläche erlaubt über die freie Wahl der Zoomstufe eine Maßstabswahl und macht den Kartenausschnitt verschiebbar, so dass jeder im eingesetzten Kartennetz abbildbare Punkt der Erde angezeigt werden kann.

Wegen der sehr großen Anzahl von Kacheln werden diese in größeren Maßstäben *just in time* gezeichnet. [cf. osm12c] Die Kartenbasis ist für alle Maßstäbe die jeweils minutengenau aktualisierte OSM-Datenbank.

Diese schnelle Aktualisierung ist für OpenStreetMap besonders wichtig, um den freiwillig Mitwirkenden nach einem Beitrag zur Datenbank ein schnelles Erfolgserlebnis zu geben, indem der Beitrag in der Karte weltweit für jeden sofort sichtbar wird. Aus diesem Grund muss jede Generalisierung der Karte automatisiert ablaufen. Weiterhin bedeutet die Zahl der Beiträge mit teilweise über 1000 *changesets* pro Stunde, [cf. Nei15] dass der Arbeitsaufwand für eine manuelle Generalisierung prohibitiv hoch wäre. Klassische

3 Neuere Forschung bestätigt dies jedenfalls allgemein und anekdotenhaft. [cf. PM17, 2]

4 In 2013 wurde ein noch größerer, 20. Maßstab ergänzt (bei null beginnend gezählt [cf. Dÿk82] die Zoomstufe 19).

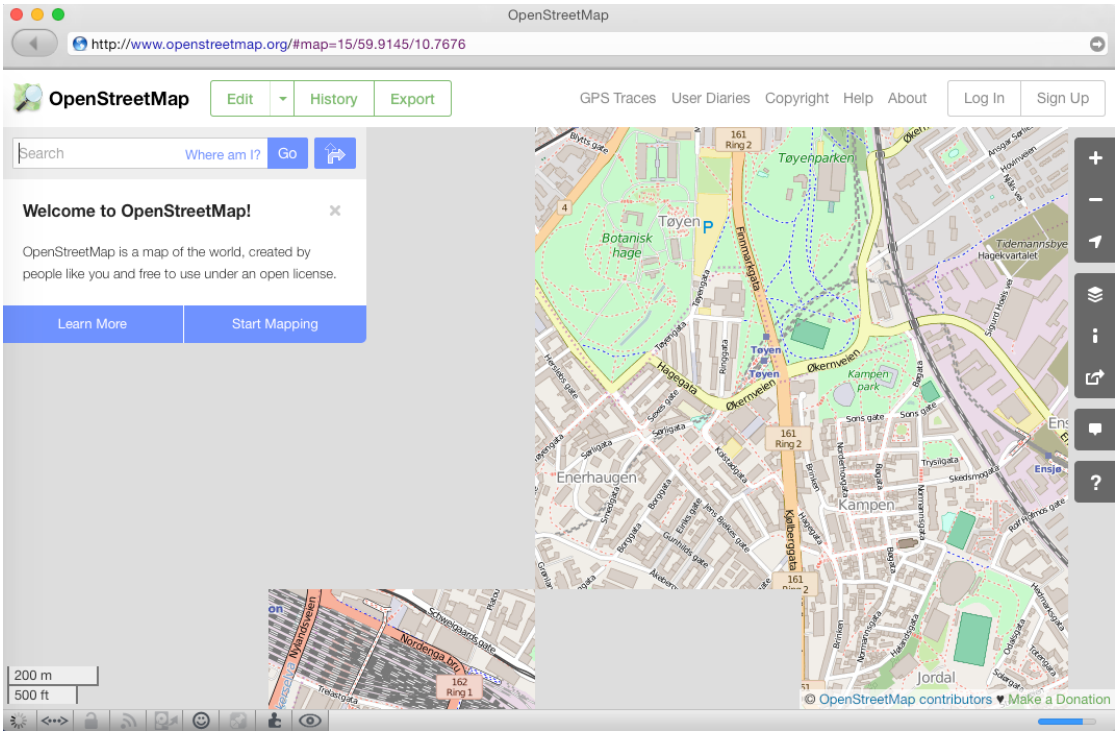

Abbildung 1: Projekt-Homepage von OpenStreetMap während des Ladevorgangs

kartographische Generalisierung von Hand kommt daher nur für nicht nachzuführende und geographisch begrenzte Einzelanfertigungen in Betracht, etwa wenn auf Basis von OpenStreetMap eine traditionelle gedruckte Karte hergestellt werden soll.

Typischerweise besteht die Grundlage für Web-Karten wie derjenigen auf `osm.org` aus einer Geodatenbank, die mit einer Kopie der OpenStreetMap-Datenbank gefüllt und anschließend minütlich aktualisiert wird (Abbildung 2). Diese Geodatenbank enthält die Geometrie und Attribute *(tags)* der Rohdaten. Vom Kartenleser in der interaktiven Oberfläche im Webbrowser erzeugte Anfragen nach Kacheln der Karte werden dann von der Renderer-Software unter Zugriff auf die Geodatenbank abgearbeitet, wobei die Geometrie anhand der vorgegebenen Zeichenregeln dargestellt wird. Gerenderte Kacheln werden in einem Cache abgelegt und zum Browser geschickt.

Eigene Bearbeitungsschritte zur Generalisierung von Daten sind bei diesem Ablauf nicht vorgesehen (im Gegensatz zu den Abläufen im AAA-Modell der deutschen Landesvermessung [cf. ML12, 263]). Lediglich als Nebeneffekt des Einlesens in die Datenbank und des Zeichnens durch den Renderer wird eine begrenzte Modell- bzw. kartographische Generalisierung durchgeführt. [cf. RT09, 194, 198]

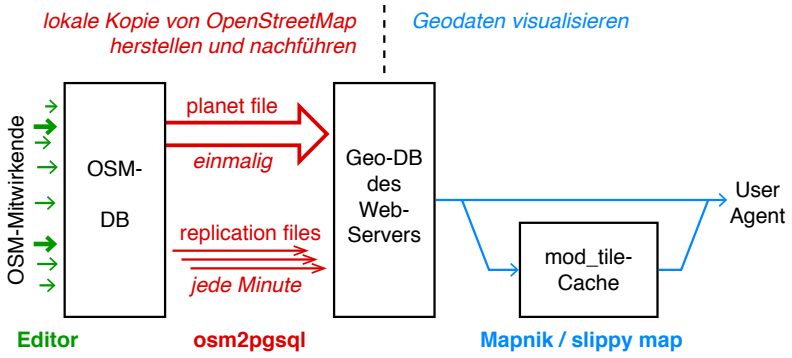

Abbildung 2: typischer OpenStreetMap-Datenfluss (vereinfacht dargestellt)

2.3 Automatisierte Linien-Generalisierung von OpenStreetMap-Daten

Hake et al. benennen sieben elementare Generalisierungsvorgänge: „Vereinfachen – Vergrößern – Verdrängen – Zusammenfassen [...] – Auswählen [...] – Klassifizieren – Bewerten [...]." [HGM02, 168] Von diesen lassen sich Auswahl (auf Basis der Objektklasse) sowie Vergrößerung und Bewertung (durch veränderte Strichbreite) vergleichsweise einfach im Renderer implementieren, da die Geometrie der zugrundeliegenden Geodaten dabei unverändert bleibt.

Andere Arten kartographischer Generalisierung wie etwa Formvereinfachung oder Zusammenfassung würden die Geometrie verändern, sind damit schwieriger zu implementieren und werden bisher nicht oder kaum eingesetzt. Dies demonstrieren im Folgenden Beispiele zu einigen Generalisierungsvorgängen.

2.3.1 Generalisierung durch Objektauswahl

Zur Herstellung der Kacheln der Web-Karte wird für jede Zoomstufe eine Auswahl der jeweils darzustellenden Objektklassen vorgenommen. Beispielsweise werden Anliegerstraßen *(residentials)* unterhalb von Zoom 12 nicht mehr dargestellt (Abbildungen 3 und 4). Dies vereinfacht das Kartenbild für den Betrachter und verringert die zu berechnende Datenmenge für den Renderer.

2.3.2 Generalisierung durch Vergrößerung

Die Wahl der Zeichenregeln beinhaltet für einige Objektklassen auch eine Generalisierung durch Vergrößerung, indem für bestimmte Maßstäbe eine Strichbreite gewählt wird, welche im Abbildungsmaßstab die tatsächliche Größe des Objekts übersteigt. Anliegerstraßen haben für die Zoomstufen 14 bis 19 jeweils eine unterschiedliche Strichbreite von 3 px bis 17 px definiert, so dass die Darstellung der Straße in der Karte Maßstabsänderungen andeutungsweise folgt (Abbildungen 5 und 6). [cf. All15] Die Breite von 3 px auf Zoom 14 entspräche ca. 27 m in der Realität. [cf. RT09, 155] Auf Zoom 13 werden

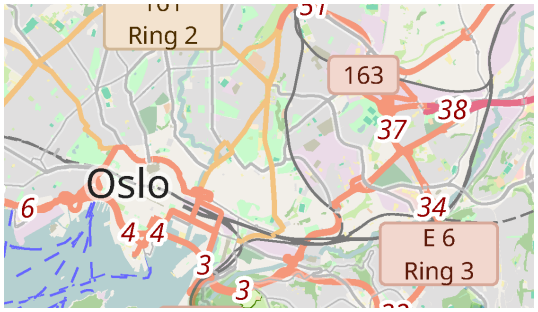

Abbildung 3: Karte von `osm.org`, Zoom 11, *residentials* nicht dargestellt [osm18]

Abbildung 4: Karte von `osm.org`, Zoom 12, *residentials* dargestellt [osm18]

Abbildung 5: Karte von `osm.org`, Zoom 14, *residentials* 3 px breit [osm18]

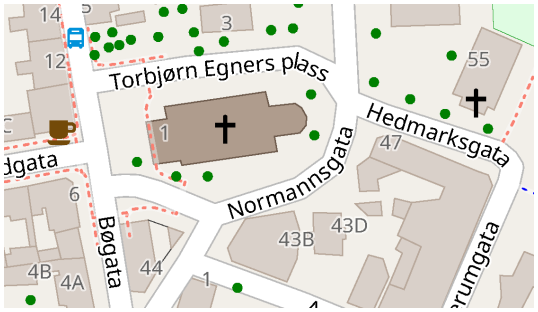

Abbildung 6: Karte von `osm.org`, Zoom 17, *residentials* 12 px breit [osm18]

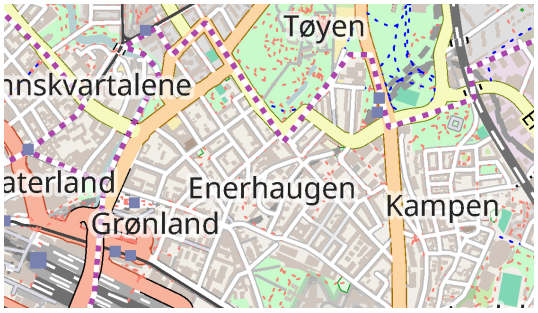

Abbildung 7: Karte von `osm.org`, Zoom 13, *residentials* 3 px breit [osm18]

Anliegerstraßen gar auf dieselbe Breite von 3 px gezeichnet (ca. 57 m) und damit deutlich vergrößert, um die Erkennbarkeit im kleinerem Maßstab zu fördern (Abbildungen 5 und 7).

2.3.3 Generalisierung durch Bewertung

Weiterhin wird durch die Wahl der Zeichenregeln eine Bewertung der tatsächlich dargestellten Objekte vorgenommen, auch dies abhängig vom Maßstab. So werden Anliegerstraßen auf Zoom 12 als schmaler, einfacher Strich gezeichnet, ab Zoom 13 jedoch als weißer Breitstrich mit grauem Rand (Abbildungen 4 und 7).

Ein weit verbreitetes unklares Kartenbild lässt sich allerdings als scheinbare Betonung einiger auf `osm.org` gezeichneten Straßen missverstehen, wie Abbildung 8 beispielhaft zeigt. Die als Sammelstraße klassifizierte Croeselaan (bei Markierung ①) wird mit breiterem Strich gezeichnet als Teile der Hauptstraßen (bei ②). Grund dafür sind nicht tatsächliche Unterschiede im Ausbauzustand[5] oder der Verkehrsbedeutung, sondern der Umstand, dass die Fahrbahnen der niederrangigen Straße durch einen breiten, parkähnlichen Grünstreifen getrennt werden (Abbildung 9). In kleinerem Maßstab überlappen sich die verbreitert gezeichneten Signaturen der beiden Fahrbahnen und fließen zusammen, so dass sich für den Betrachter der irreführende Eindruck einer besonders breiten oder wichtigen Straße ergibt.

Gleiches gilt für die ebenfalls in Abbildung 8 auffallende wechselhafte Strichbreite einer Hauptstraße: Hier existiert bei Markierung ② nur eine einzige Fahrbahn, während sich bei ③ die Signaturen von zwei parallelen Fahrbahnen überlappen.

Abbildung 8: Karte von `osm.org`, Zoom 14, Zusammenfließen von benachbarten Fahrbahnen [osm18]

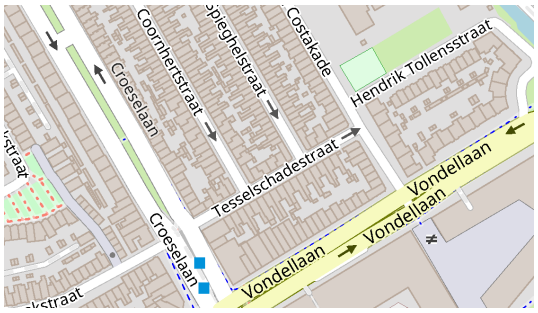

Abbildung 9: Karte von `osm.org`, Zoom 16, Fahrbahnen getrennt erkennbar (① in Abb. 8) [osm18]

Abbildung 10 zeigt ein ähnlich unklares Kartenbild. Hier ist die Hauptstraße längs der Eisenbahnstrecke bei Markierung ① deutlich breiter als bei ②. Auch hier liegt keine gezielte Bewertung vor; vielmehr wird die Straße von der Bahnstrecke teilweise verdeckt, so dass die Karte den falschen Eindruck einer unterschiedlichen Strichbreite verschafft. Die eigentlich notwendige Verdrängung findet nicht statt.

Interessanterweise kehrt sich in größerem Maßstab das Verhältnis der Strichbreiten um: In Abbildung 11 ist die Hauptstraße bei Markierung ① nun deutlich *schmaler* als bei ②. Dies liegt an getrennt erfassten Richtungsfahrbahnen bei ②, wie sie schon in Abbildung 9 gezeigt wurden, während bei ① mangels baulicher Trennung der beiden Fahrtrichtungen

5 Der kürzlich begonnene in der Abbildung bei ② zu erkennende Neubau einer Busfahrbahn verschmälert die Hauptstraße nicht.

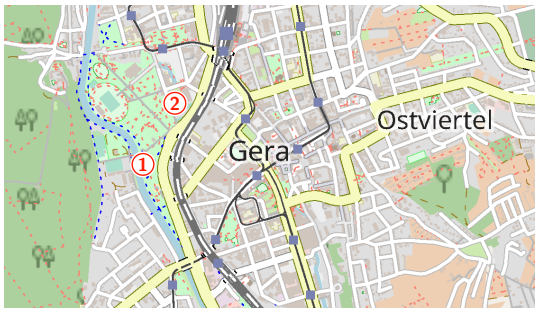

Abbildung 10: Karte von `osm.org`, Zoom 13, Straße teilweise durch Bahnstrecke verdeckt [osm18]

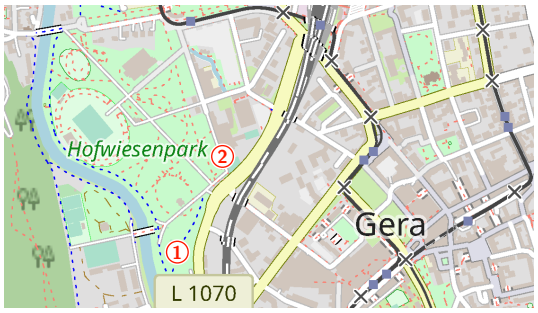

Abbildung 11: Karte von `osm.org`, Zoom 14, Straße neben Bahnstrecke [osm18]

nur ein einziger Linienzug erfasst ist. Wiederum handelt es sich nicht um eine gezielte Bewertung; der Anschein unterschiedlicher Strichbreiten ist unbeabsichtigt.

2.3.4 Generalisierung durch Formvereinfachung

Eine kartographische Formvereinfachung wird für die Web-Karte nicht vorgenommen. Zufriedenstellende, voll automatisiert arbeitende Algorithmen dafür stehen nicht zur Verfügung und ein Eingreifen von Hand ist wie zuvor in Abschnitt 2.2 erläutert nicht vorgesehen. [cf. Kla11, 25] In kleineren Maßstäben ergibt dies beispielsweise für Straßen mit vielen kleineren Kurven eine zittrige Darstellung, so etwa in Abbildung 12 bei Markierung ①. Der Leser kann mit etwas Phantasie zwar erkennen, dass die Straße nicht absolut gerade verläuft, jedoch macht dies die Darstellung undeutlich und ist auch in diesem Maßstab nicht von Bedeutung.

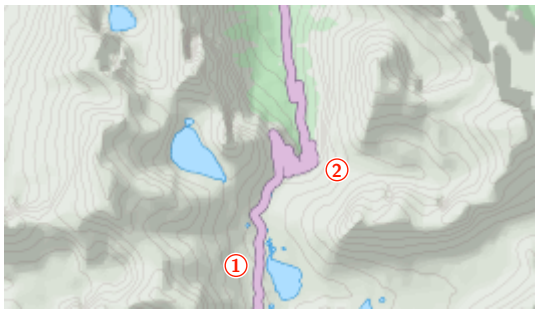

Abbildung 12: Trollstigen, Web-Karte mit OpenStreetMap-Daten, 1 : 135 000 [All17]

Abbildung 13: Trollstigen, manuell kartographisch generalisiert, 1 : 135 000 [nma17]

An Serpentinen fällt dieser Mangel an Generalisierung besonders auf (Markierung ②). Im Extremfall kann sich hier in kleinem Maßstab durch das Zusammenfließen der Strecken zwischen den Kehren der falsche Eindruck einer großen, ebenen Straßenfläche ergeben (Abbildung 12). Eine sorgfältige Formvereinfachung müsste dagegen den Charakter

der Straße durch Darstellung nur einiger ausgewählter Kehren mit vereinfachtem Verlauf erhalten (Abbildung 13).

„Die Formvereinfachung ist insgesamt eine Reduzierung der Anzahl von Stützpunkten, wenngleich u. U. lokal Stützpunkte hinzugefügt werden, um typische Formen übertreibend zu betonen.“ [BK01, Formv.] Während Letzteres nach Kenntnis des Verfassers gegenwärtig nicht automatisiert möglich ist, kann eine bloße Reduzierung der Zahl der Stützpunkte zur Linienglättung automatisiert erfolgen. Ein wichtiger Nebeneffekt ist dabei die verringerte Menge an Geodaten und damit deren vereinfachte Verarbeitung. Der vielfach benutzte Algorithmus von Ramer, Douglas und Peucker zur Linienglättung zielt gerade auf Effizienz ab („efficient representation“ [Ram72, 244]). Eine Formvereinfachung nach kartographischen Gesichtspunkten ergibt sich mit ihm in der Regel nicht.

Eines der Probleme bei der Automatisierung der Formvereinfachung besteht in der Notwendigkeit, alle vereinfachten Linien aufeinander abzustimmen, damit die charakteristischen Formen im Kartenbild insgesamt erhalten bleiben. [cf. BK01, ibid.] Dies wird unter anderem dort deutlich, wo parallele Linienzüge mit Ramer-Douglas-Peucker vereinfacht werden.

Beispielsweise zeigt Abbildung 14 Richtungsfahrbahnen von Autobahnen, die nach Vereinfachung mit Ramer-Douglas-Peucker durch die ungleichmäßige Verteilung ihrer Stützpunkte stark variierende Abstände zueinander haben und sich sogar teilweise überkreuzen. Dieses Problem ließe sich wenigstens für den Fall paralleler Linien leicht umgehen, wenn diese vor der Formvereinfachung zusammengefasst werden könnten.

Als weiteres Beispiel sind breite Flüsse zu nennen, für die in OpenStreetMap üblicherweise neben einer Mittellinie auch beide Uferlinien erfasst sind, um die flächenhafte Ausdehnung abzubilden. Je nach den gewählten Parametern kann eine Formvereinfachung hier dazu führen, dass die Mittellinie das Flussbett verlässt. [cf. Kla11, 57]

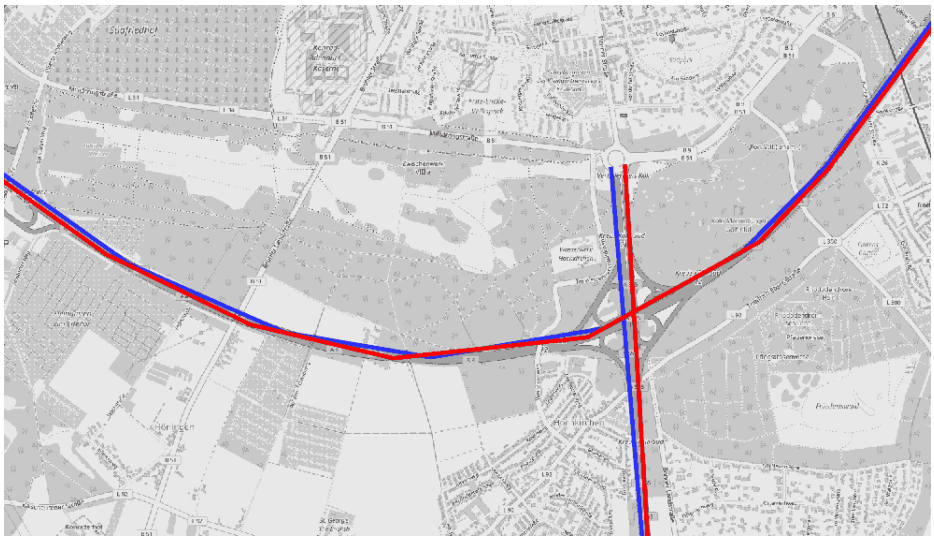

Abbildung 14: Überkreuzen paralleler Linienzüge nach Vereinfachung mit Ramer-Douglas-Peucker [Hintergrund osm18, entsättigt]

2.3.5 Generalisierung durch Verdrängung

Typisch für Web-Karten ist das völlige Fehlen von Verdrängungen zur Einhaltung kartographischer Minimalabstände und -dimensionen. Statt dessen werden mit kleiner werdendem Maßstab Objekte dichter und dichter aneinander, schließlich gar übereinander gezeichnet, worunter das Kartenbild teils erheblich leidet (Abbildung 15). Web-Karten mit OpenStreetMap-Daten sind hier keine Ausnahme.

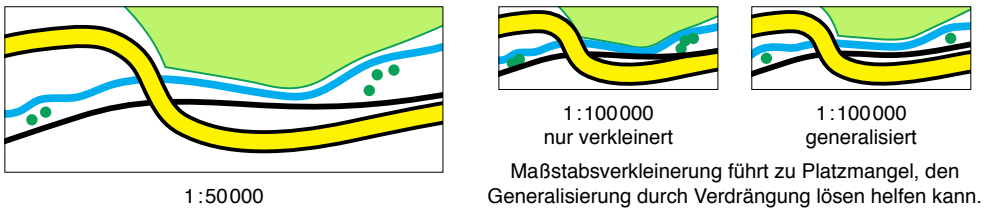

Abbildung 15: typisches Generalisierungsergebnis in Web-Karten [cf. sgk02, 49]

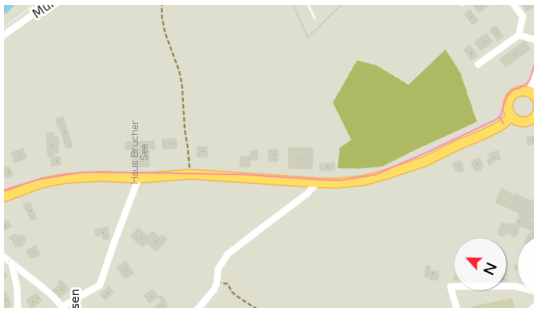

Abbildung 16: straßenbegleitender Radweg (rot) schneidet Straßenfläche [mm16, rotiert]

Abbildung 17: straßenbegleitender Radweg (blau strichliert) von Straße verdeckt [osm18, rotiert]

Abbildungen 16 und 17 zeigen beispielhaft, wie ein straßenbegleitender Radweg mal „über“, mal „unter“ der Straße zu liegen kommt. Keiner dieser beiden Ansätze ist kartographisch gelungen. Bevor eine Generalisierung durch Verdrängung stattfinden kann, muss zunächst der Konflikt zwischen Straße und Radweg erkannt werden. Es ist denkbar, dass dies durch Untersuchen auf Parallelität geschehen oder erleichtert werden könnte.

2.3.6 Generalisierung durch Zusammenfassung

Die Generalisierung durch Zusammenfassung von Linienzügen kommt vor allem bei Parallelität in Betracht. Dabei treten immer wieder in verschiedenen Situationen Darstellungsprobleme auf.

Beispielsweise variiert in OpenStreetMap auffallend oft der Abstand von zwei Richtungsfahrbahnen einer Straße. Abbildung 18 zeigt, wie innerhalb weniger hundert Meter die Linearsignaturen zweier Richtungsfahrbahnen teilweise so weit voneinander entfernt

liegen, dass dazwischen eine Lücke entsteht (Markierung ①), aber teilweise auch so dicht beieinander, dass die Lücke geschlossen wird und die Randstriche der beiden Signaturen ineinander übergehen, was den Eindruck eines Mittelstriches wechselnder Strichstärke bewirkt (②). An einer Stelle sind die beiden Richtungsfahrbahnen gar so dicht beieinander gezeichnet, dass der Rand- bzw. Mittelstrich verschwindet und beide Fahrbahnen ineinander übergehen (③).

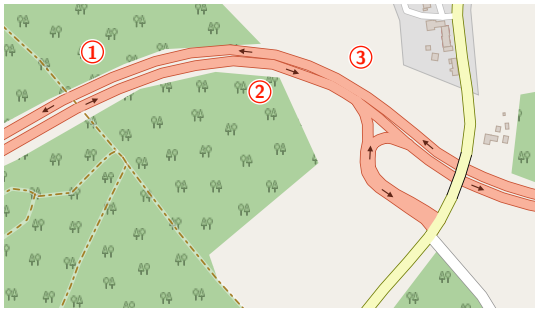

Abbildung 18: variierende Abstände von Richtungsfahrbahnen [osm18, bearbeitet]

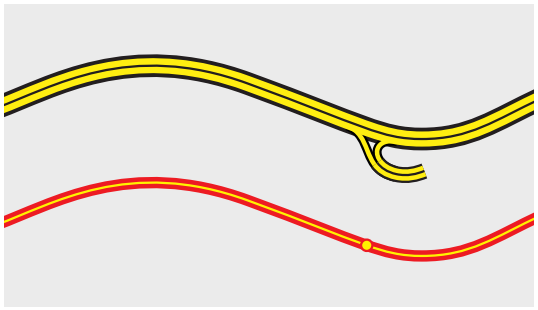

Abbildung 19: kombinierte Signaturen für autobahnähnliche Straßen mit Anschlussstelle

In Maßstabsbereichen, in denen die Grundrisstreue keine Rolle spielt, könnte möglicherweise eine stattdessen entlang der mittig zwischen den beiden Fahrbahnen verlaufenden Straßenachse gezeichnete kombinierte Signatur ein besseres Kartenbild erreichen (Abbildung 19). Da jedoch in OpenStreetMap die gesonderte Erfassung der Straßenachse unüblich ist, müssten hierzu zunächst die beiden Richtungsfahrbahnen als zusammengehörig (parallel) erkannt werden.

In Deutschland erfasst die OpenStreetMap-Community autobahnähnliche Straßen als *highway=trunk* anhand ihres Ausbauzustands. Allerdings bildet die OpenStreetMap-Straßenklassifizierung historisch bedingt das britische System ab, wo *trunk roads* ausdrücklich keinen bestimmten Ausbauzustand haben, sondern das Kernnetz der wichtigsten Straßen erster Ordnung bilden. Unter Umständen wird *highway=trunk* somit auch für schmalere Landstraßen verwendet.

Wird den Gepflogenheiten in Deutschland entsprechend für *highway=trunk* eine auf den Ausbau mit baulich getrennten Fahrbahnen hindeutende Signatur gewählt, so birgt dies im internationalen Einsatz Potenzial für Verwirrung. Abbildung 20 zeigt eine solche Situation aus Norwegen: Die Kartensignatur erweckt den Eindruck eines autobahnähnlichen Ausbaus, tatsächlich handelt es sich jedoch um eine einfache Landstraße mit Grundstückszufahrten und engen Kurvenradien (Abbildung 21).

Dass aus dem OpenStreetMap-Datenmodell samt Attributen der tatsächliche Ausbauzustand oft nicht unmittelbar hervorgeht, macht die Möglichkeit einer algorithmischen Ableitung desselben aus der Geometrie der Geodaten interessant. Möchte man einheitliche Zeichenregeln verwenden, so wäre auch hier die Möglichkeit einer Erkennung von zwei Richtungsfahrbahnen als parallel und zusammengehörig hilfreich.

Abbildung 20: Landstraße (Vallaheiane) als *highway = trunk* mit autobahnähnlicher Signatur [osm17b]

Abbildung 21: typische norwegische Fernstraße (*riksveg* E 39, Vallaheiane), als *highway = trunk* erfasst

Die Probleme durch fehlende Generalisierung durch Zusammenfassung sind nicht auf das Straßennetz beschränkt. So wird in OpenStreetMap mit *tracks* = ∗ die Anzahl der Eisenbahngleise gekennzeichnet, für die ein *way* steht. Die Karten in Abbildung 22 stellen diesen *tag* farblich dar.

Bei Betrachtung in kleinem Maßstab sieht darin ① zunächst nach einer viergleisigen Strecke aus, in großem Maßstab plötzlich nach wenigstens *zwei* viergleisigen Strecken nebeneinander. Tatsächlich liegen insgesamt vier einzelne Gleise, die jeweils durch einen *way* mit dem Attribut *tracks* = *4* erfasst sind.[6] In kleinem Maßstab fließen sie zusammen, in großem Maßstab sind sie jedoch einzeln erkennbar und vermitteln dem Leser das falsche Bild von insgesamt 16 Gleisen. Möglicherweise hat ein OpenStreetMap-Beitragender *tracks* = ∗ missverstanden als die Gesamtanzahl der Gleise, [cf. F12] wofür jedoch das Attribut *passenger_lines* = ∗ zu verwenden ist.

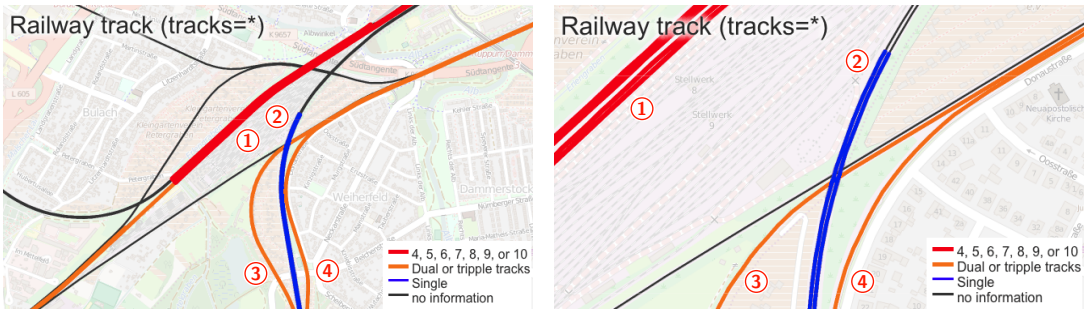

Abbildung 22: fehlerhafte Repräsentation der Gesamtzahl paralleler Gleise mit dem *tracks*-Attribut (links Zoom 15, rechts Zoom 17; Karlsruhe-Weiherfeld) [cf. ito12]

Analog kann in kleinem Maßstab der Eindruck entstehen, dass von der eingleisigen Strecke ② zwei doppelgleisige Strecken ③ und ④ derart abzweigen, dass am unteren Kartenrand insgesamt fünf Gleise nebeneinander liegen. In größerem Maßstab wird erkennbar, dass es sich gerade umgekehrt verhält: Die Strecke ② ist korrekt mit zwei

6 Die *tracks*-Attribute wurden zwischenzeitlich durch Beitragende von diesen *ways* entfernt.

parallelen Gleisen erfasst, deren Signaturen jedoch im kleineren Maßstab zusammenfließen; die Abzweige ③ und ④ bestehen allerdings aus jeweils nur einem einzelnen Gleis, das fälschlich das Attribut *tracks = 2* trägt.

Insgesamt sind die Darstellungen von *tracks* = ∗ in Abbildung 22 wenig gelungen.[7] Das unglücklich benannte Attribut *passenger_lines* reicht dazu aus, *tracks* = ∗ für manche Anwendungen zu ersetzen, wird jedoch nur auf speziellen Karten angezeigt, so dass es nicht annähernd flächendeckend erfasst ist (Abbildung 23).[8]

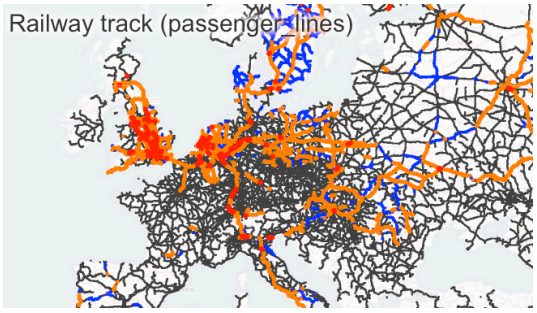

Abbildung 23: Eisenbahnstrecken in Europa, für die *passenger_lines* = ∗ erfasst ist [ito16]

2.4 Zielsetzung der Arbeit

Das automatisierte Zusammenfassen paralleler Linienzüge kann zur Vermeidung einiger der oben diskutierten Probleme bei der Generalisierung und Darstellung von Geodaten beitragen.

Bereits das Zusammenfassen selbst ist ein Vorgang der kartographischen Generalisierung. Es kann das Kartenbild vereinfachen und verbessern.

Beim Zusammenfassen könnten Attribute der beteiligten *ways* derart aggregiert werden, dass zuvor durch Geometrie ausgedrückte Zusammenhänge in den generalisierten Daten erhalten bleiben. Beispielsweise könnte beim Zusammenfassen paralleler Gleise die in *passenger_lines* vermerkte Gleisanzahl der zuvor getrennten *ways* für das generalisierte Ergebnis addiert werden (vgl. Abschnitt 2.3.6). Die so vereinfachten Geodaten können dann entsprechend ihrer Attribute visualisiert werden. Die Gefahr von missverständlichen oder unschönen Kartenbildern, die wie oben gezeigt etwa durch Überlagerung entstehen können, wird so verringert.

7 Hier zeigt sich eine Schwäche von OpenStreetMap: Zwar ist mit *relations* eine logische Verknüpfung paralleler Gleise in der OpenStreetMap-Datenbank technisch möglich, jedoch ist dies nicht zuverlässig und simpel genug, dass es von einer Mehrheit der OpenStreetMap-Beitragenden angewandt wird.

8 Bis Fertigstellung der vorliegenden Arbeit haben OpenStreetMap-Beitragende das Attribut *passenger_lines* für die meisten europäischen Strecken ergänzt.

Ebenfalls verringert wird die Datenmenge an sich. Das Arbeiten mit Geodaten für große Gebiete, aber auch die Weiternutzung für andere Zwecke wird so unter Umständen einfacher.

Zum Beispiel sind automatisierte Formvereinfachungen einfacher durchzuführen, wenn keine Rücksicht auf eventuell parallele Linienzüge mehr genommen werden muss (siehe Abbildung 14).

Ziel *dieser* Arbeit ist die Entwicklung von Algorithmen zur Generalisierung durch Zusammenfassung parallel verlaufender Linienzüge in OpenStreetMap. Weitere Generalisierungsvorgänge sind nicht Teil dieser Arbeit, insbesondere weder eine Formvereinfachung noch eine Verdrängung.

2.5 Diskussion existierender Ansätze zur automatisierten Linien-Generalisierung

2.5.1 Zusammenfassen durch Pufferung

Das Projekt „OSM-3D“ verwendet OpenStreetMap-Daten automatisiert zum Aufbau eines 3D-Modells für die ganze Welt. Die *ways* aus OpenStreetMap werden für die 3D-Szene durch Pufferung mit attributabhängigem Radius in Polygone gewandelt, um ein Gesamtbild mit realistischen Breiten von Straßen, Wegen und Bahnlinien zu erzeugen. Um dabei durch Überlappungen von eng parallelen *ways* entstehende unnötig große Datenmengen zu vermeiden, werden diese Polygone anschließend vereinigt. [cf. OHSZ10, 2–3]

Durch diesen Ansatz werden unerwünschte Parallelen beseitigt. Allerdings ist die für diese Arbeit erwünschte Rückführung der durch die Pufferung entstandenden Flächen zu einer Linie nicht ohne Weiteres möglich. Auch werden Parallelen nicht als solche erkannt, sondern entfallen restlos durch die Polygonvereinigung. Daher erscheint dieser Ansatz isoliert betrachtet für die dieser Arbeit zugrundeliegende Fragestellung nicht geeignet.

2.5.2 Bildung von Skelettlinien

Eine Möglichkeit, Flächen auf innere Linien zu reduzieren, bietet die Skelettierung. Für das *straight skeleton* werden Polygone so lange sukzessive geometrisch verkleinert, bis nur noch ein aus geraden Linien zusammengesetztes topologisches Skelett übrig bleibt (Abbildung 24). Hierfür existieren unterschiedliche Algorithmen. [cf. wp13c]

Haunert und Sester nutzen ein *straight skeleton*, um aus Flächen in der Automatisierten Liegenschaftskarte (ALK) Linien zur Nutzung in einer Multiple Representation Database (MRDB) abzuleiten. Der Algorithmus von Eppstein und Erickson [EE99] musste für die ALK-Daten angepasst und erweitert werden, um das resultierende Straßennetz auf die gewünschten Mittellinien der Flächen zu reduzieren. Dennoch kommt es zu teils erheblichen Problemen an Kreuzungen, die nicht in allen Fällen gelöst werden können. [cf. HS04, 4–6]

Migurski setzt eine Skelettierung ein, um für eine topographische Karte auf OpenStreetMap-Basis Straßennamen und -nummern graphisch ansprechend darzustellen. Im

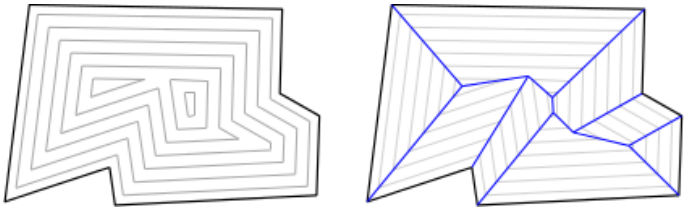

Abbildung 24: sukzessive Verkleinerung einer Fläche (links) und deren Skelettlinien (blau; rechts) [Hub04]

Falle von zwei zueinander parallelen Fahrbahnen derselben Straße sollen Nummer und Name so gezeichnet werden, dass sie für beide Fahrbahnen gelten und unschöne Doppelungen des Namens vermieden werden. Hierzu benutzt Migurski die Software Skeletron. ([ME13], cf. [Mig12])

Skeletron benutzte ursprünglich ein *straight skeleton* nach dem erwähnten Ansatz von Haunert und Sester, was aber letztlich nicht besonders gut funktioniert hat („ultimately didn't work very well" [ME13, Readme]). Inzwischen wird die von Ladak und Martinez für diesen Zweck beschriebene Thiessen-Polygon-Methode eingesetzt.

Hierbei werden ausgehend von einer sehr detaillierten Straßenfläche zunächst deren Kanten verdichtet auf mehrere Knoten pro Meter, bevor ausgehend von diesen Knoten mittels Delaunay-Triangulierung ein Voronoi-Diagramm berechnet wird, dessen Kanten die Skelettlinien der Straßenfläche sind. Auch hier ergeben sich allerdings erhebliche Probleme an allen Kreuzungen, die trotz Einbeziehung von Metadaten nur teilweise automatisch gefiltert werden. Solche Fälle können immerhin automatisch erkannt und dann manuell beseitigt werden. Der Aufwand für die Implementierung wird mit sieben Personenmonaten beschrieben. [cf. LM96]

Um aus den OpenStreetMap-*ways* Flächen zu erhalten, mit denen das Anwenden der Thiessen-Polygon-Methode erst möglich wird, puffert Skeletron zuvor die OpenStreetMap-*ways* wie in Abschnitt 2.5.1 beschrieben. [cf. ME13]

2.5.3 Konflikterkennung

Van Kreveld und Peschier beschreiben eine Möglichkeit, bei der automatischen Ableitung von Straßenkarten Konflikte nahe beieinander liegender Straßen zu erkennen und durch Auswahl nur einer der beteiligten Straßen aufzulösen. Neben anderen Bedingungen werden so auch graphische Mindestabstände eingehalten. [cf. vKP98]

Es liegt auf der Hand, dass sich parallele Straßen meist durch ein Unterschreiten dieser Mindestabstände auf ihrer gesamten Länge auszeichnen. Das beschriebene Verfahren untersucht allerdings lediglich den minimalen Abstand zweier Straßen. Das Untersuchen der gesamten Länge der Straße wäre erheblich teurer. [cf. vKP98, 4.] Dieser Ansatz erscheint daher für diese Arbeit wenig erfolgversprechend.

Thom erwähnt ausdrücklich, dass die Erkennung eines Konflikts von parallelen Straßenabschnitten für die automatische Generalisierung hilfreich wäre („... recognizing stret-

ches of parallel road sections would help their automatic generalization“ [Tho06a, 676]), allerdings ohne dabei eine mögliche Lösung anzudeuten.

2.5.4 Visuelle Wahrnehmung

Thomson beschreibt, wie das menschliche Gehirn optisch wahrgenommene Elemente selbst dann „spontan organisiert“, wenn deren Semantik völlig unbekannt ist. Er bezeichnet dieses Konzept als „perceptual grouping“ und zählt unter den dazu herangezogenen geometrischen Eigenschaften neben Nähe, Ähnlichkeit und Symmetrie auch Parallelität und Kontinuität auf. [cf. Tho06b, 683–684]

Zusammen mit Richardson beschreibt er eine Möglichkeit, die Eigenschaft der Kontinuität von Linienzügen (genannt *stroke*) zu nutzen, um deren Wichtigkeit einzustufen und damit ein Straßennetz zu generalisieren („Principle of Good Continuation“, Abbildung 25). [cf. TR99] Edwardes und Mackaness nutzen denselben Ansatz zur Analyse des Netzes, um damit ihre auf Graphentheorie basierende Generalisierung zu unterstützen. [cf. EM00] Wie ein „perceptual grouping“ nach der Eigenschaft der Parallelität aussehen könnte, wurde nach Kenntnis des Verfassers bisher nicht konkret untersucht.

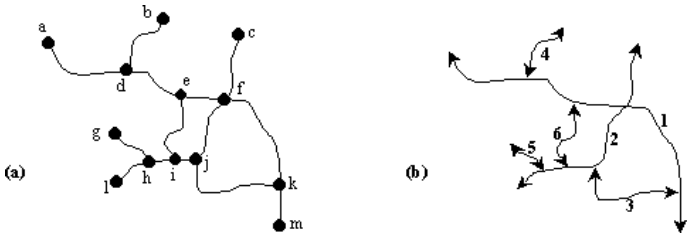

Abbildung 25: ein Netzwerk (a) und seine *strokes* (b) [© TR99, *figure* 2]

Chaudhry und Mackaness schlagen das Konzept von *strokes* zur Generalisierung von Straßennetzen durch Auswahl wichtiger Straßen für die digitale OS MasterMap des Ordnance Survey vor. Sie betonen jedoch, dass trotz ermutigender Resultate noch weitere Probleme gelöst werden müssen. Auf die Eigenschaft der Parallelität gehen sie nicht ein. [cf. CM05]

Thom beschreibt vier Schritte zum Zusammenfassen getrennter Richtungsfahrbahnen von zweibahnigen Straßen und Verkehrsinseln auf eine gemeinsame Mittellinie für die OS MasterMap. Zunächst werden unter Nutzung von *strokes* und Attributen durchgehende Linienzüge für die Gesamtlänge der getrennten Richtungsfahrbahnen erzeugt, die dann durch Graphenanalyse ihrer jeweiligen Enden einander zugeordnet werden können. Anschließend kann durch Skelettierung die Mittellinie der zugeordneten Linienzüge berechnet und im letzten Schritt noch Korrekturen an deren Enden vorgenommen werden. [cf. Tho05, 3–10]

Diese Schritte liefern in vielen Fällen ein gutes Ergebnis, was Thom der Attribuierung der Ausgangsdaten zuschreibt. Entscheidend ist insbesondere das „one way“–Attribut sowie die Klassifikation nach Bauart der Straße (einbahnig, zweibahnig, Verkehrsinsel, Kreisverkehr etc.). [cf. Tho05, 14]

2.5.5 Graphenanalyse

Frühe Entwicklungen der automatisierten Generalisierung versuchten oft, unter Vernachlässigung der Topologie des Ergebnisses den Fokus des Kartographen auf der Visualisierung nachzuahmen. [cf. TR95, 1871] Ansätze der Graphentheorie können jedoch nicht nur wie zuvor beschrieben andere Methoden unterstützen, sondern auch direkt zur Generalisierung eingesetzt werden.

Heinzle et al. zeigen Methoden, um typische Muster in Straßennetzen zu erkennen und daraus Informationen abzuleiten. Es liegt auf der Hand, dass parallele Fahrbahnen einer zweibahnigen Straße ein typisches Muster sein können, jedoch werden diese nur beiläufig erwähnt. [cf. HAS05, 3.2.]

Jiang und Claramunt verwenden Zentralitätsmaße, um die wesentliche Struktur eines Straßennetzes zu identifizeren und bei der Generalisierung zu erhalten. Dieser Ansatz entspricht dem Prinzip, dass schlechter ans Netz angebundene Straßen weniger wichtig sind als jene, die gut angebunden sind („less connected streets are less important than those well connected from a structural point of view“ [JC04, 161]). [cf. JC04] Für parallele Kanten im Graph ist allerdings zu erwarten, dass sie jeweils ähnlich gut an den Rest des Netzes angebunden sind, weswegen dieses Kriterium für diese Arbeit nicht als nützlich erscheint.

Probleme an Kreuzungen wurden bereits von mehreren Autoren genannt. Mackaness und Mackechnie demonstrieren eine Methode, um Straßenkreuzungen durch eine Kombination von räumlicher und graphentheoretischer Analyse zu identifizieren und zu vereinfachen. [cf. MM99]

Der Gedanke, dass etwa im Falle einer zweibahnigen innerstädtischen Straße durch Generalisierung komplexer Kreuzungen auf einen einzelnen Punkt die vormals nur geometrisch parallelen Kanten zu zwei topologisch parallelen Kanten (Mehrfachkanten) in einem ungerichteten Graphen des Straßennetzes werden, welche dann leicht zusammenzufassen sind, liegt im Kontext dieser Arbeit nahe (Abbildung 26 verdeutlicht dies anhand von als parallel geltenden Brücken). Die Arbeit von Mackaness und Mackechnie berücksichtigt diesen Aspekt allerdings nicht.

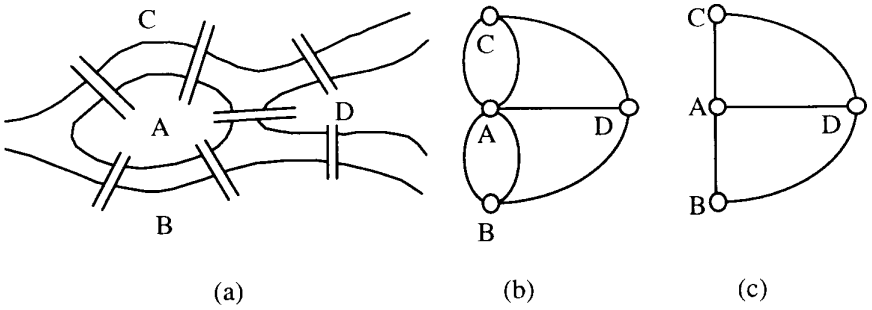

Abbildung 26: das „Königsberger Brückenproblem“ (a), seine graphentheoretische Form (b) und eine Reduktion des Graphen (c) [© MM99, 188]

3 Spezifikation der zu untersuchenden Fälle

3.1 Vergleich verschiedener Problemfälle der automatisierten Linien-Generalisierung

An dieser Stelle werden zunächst unterschiedliche Spezialfälle beschrieben, in denen die automatisierte Identifikation und Zusammenfassung parallel verlaufender Linienzüge hilfreich wäre. Der darauf folgende Abschnitt begründet die Auswahl der entsprechend der Aufgabenstellung im Rahmen dieser Arbeit im Weiteren zu behandelnden Spezialfälle.

3.1.1 Mehrgleisige Eisenbahnstrecken

Das Problem der Auswertung der Gleisanzahl mehrgleisiger Bahnstrecken ist bereits in Abschnitt 2.3.6 beschrieben. Neben der Erkennung solcher Gleise als parallel kann die automatisierte geometrische Ermittlung der Bahnachse hilfreich für eine ansprechende Visualisierung sein. Lagepläne im Eisenbahnwesen zeigen sie zusätzlich zu den Gleisachsen, in OpenStreetMap wird sie jedoch nicht erfasst.

3.1.2 Richtungsfahrbahnen im Straßenraum mit baulicher Trennung

Auch für zweibahnige Straßen, welche in OpenStreetMap als zwei parallele Linienzüge modelliert sind, wird im Gegensatz zum deutschen amtlichen Vermessungswesen in OpenStreetMap keine Mittellinie als Achse der Straße erfasst (siehe Abschnitt 2.1). Die sich daraus für OpenStreetMap ergebenden Probleme wurden bereits in Abschnitten 2.3.3 und 2.3.6 beschrieben. Konkrete Beispiele für solche Straßen sind Autobahnen, aber auch zweibahnige innerstädtische Straßen oder Überlandstraßen.

3.1.3 Parallele Wege für unterschiedliche Arten von Verkehr

Ein oft zu beobachtendes Muster sind zusammengehörende, jedoch baulich getrennte und damit jeweils als eigene Linienzüge modellierte Verkehrswege für unterschiedliche Fahrzeugtypen, Geschwindigkeitsbereiche oder Zwecke des Verkehrs. Im Straßenraum zählen dazu:

- straßenbegleitende Fuß- und Radwege,
- Nebenfahrbahnen *(frontage roads)* für Anliegerverkehr, von dem die Hauptfahrbahn freigehalten werden soll,

- langgezogene Rampen an teilplanfreien Anschlussstellen insbesondere der Bauformen Diamant und *SPUI*,[1]
- Verteilerfahrbahnen an Doppelanschlussstellen und Autobahnkreuzen,
- Sonderfahrbahnen für Busse, Fahrgemeinschaften oder Mautzahler,
- straßenbündige oder -parallele Bahnkörper, und schließlich
- die Kombination mehrerer der genannten Punkte, etwa als Teil einer komplexen innerstädtischen Straße mit parallelen getrennten Radwegen, Gehwegen, Richtungsfahrbahnen, Nebenfahrbahnen und besonderem Stadtbahn-Gleiskörper.

Die sich in solchen Fällen ergebenden Probleme sind vergleichbar zu Abschnitt 3.1.2, jedoch komplexer, weil die beteiligten Verkehrsarten jeweils unterschiedliche Voraussetzungen haben. So sind z. B. straßenbegleitende Radwege aufgrund der geringeren gefahrenen Geschwindigkeiten oft kurviger als die Fahrbahn für Kraftfahrzeuge, sollten aber dennoch als parallel zu ihr gelten können. Auch erfordern die Unterschiede in Kombination mit Platzmangel oft individuelle bauliche Lösungen, was die Automatisierung der Verarbeitung der Geodaten nicht vereinfacht.

3.1.4 Grenzen

In Abschnitt 2.3 nicht erwähnt wurden unsichtbare Grenzen, deren Verlauf dem physischer Objekte folgt oder zu ihnen parallel ist. Beispielhaft zu nennen wären Postleitzahlgebiete, deren Grenze einem Straßenzug folgt, oder administrative Grenzen entlang eines Wasserlaufs. In diesen Fällen kann in der Kartographie die Grenze vom verlaufsgebenden Objekt abgesetzt werden, damit beide Signaturen klar erkennbar sind. Auch eine Unterbrechung der Signatur ist möglich durch den dann eintretenden Stellvertretereffekt. [cf. sgk02, 82–83] Abbildung 27 zeigt dies beispielhaft. In OpenStreetMap-Karten ist eine solche Generalisierung bisher noch nicht üblich.

Ein weiteres Beispiel sind zwei aneinander angrenzende Gebiete mit Schifffahrtsbeschränkungen. Nach den Zeichenregeln für Seekarten ist in solchen Fällen die übliche T-Signatur [cf. iho13, B-439.2] nur für das gefährlichere der beiden Gebiete zu zeichnen („For coincident limits, the limit symbol (line) portraying the area which is considered to be potentially the most dangerous to navigation [...] has priority.“ [iho13, B-439.6 a.]). Eine derartige Abwägung dürfte zwar schon aus rechtlichen Gründen nicht automatisationsfähig sein. Eine Verbesserung der gegenwärtig unbefriedigenden Darstellung in der auf OpenStreetMap-Daten basierenden Karte „OpenSeaMap“ (Abbildung 28) wäre allerdings möglich und anzustreben.

Insgesamt erscheinen Grenzen jedoch im Kontext dieser Arbeit als ein eher schwieriges Feld.

1 *single-point urban interchange* (Diamant mit nur einer Kreuzung) [cf. wp13a]

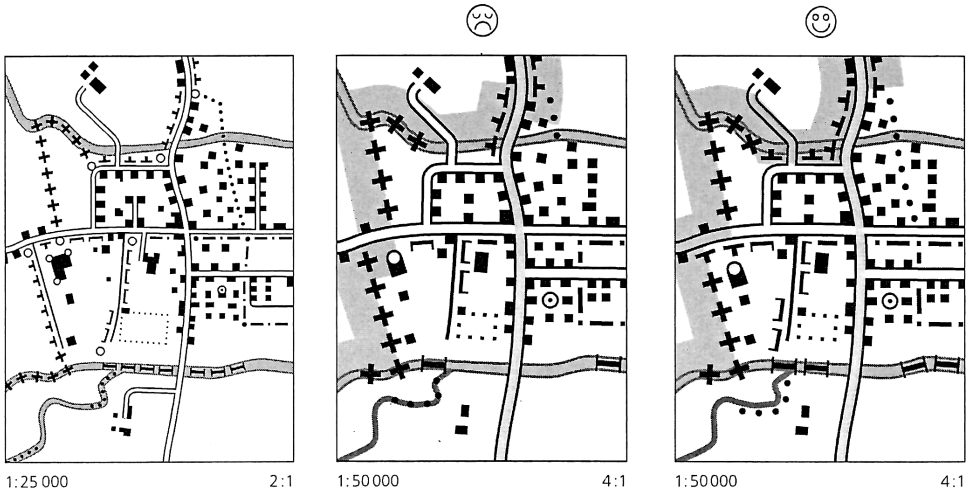

Abbildung 27: Generalisierung des Grenzverlaufs entlang physischer Objekte [© sgk02, 83, entsättigt und verkleinert]

Abbildung 28: zusammenfallender Verlauf zweier Grenzen mit einander überlagernden Signaturen (Hohwachter Bucht) [osm17a]

3.1.5 Grundrisstreu erfasste linienhafte Objekte

Mit zunehmendem Grad der Detaillierung in OpenStreetMap wird versucht, eigentlich linienhafte Objekte als grundrisstreue Fläche zu erfassen. Für Flüsse ist dies bereits üblich, [cf. RT09, 72] für Straßen und Rollbahnen in Diskussion. Dadurch entstehen sehr lange und schmale Flächen, deren Ränder größtenteils parallel zueinander sind. Bei der Darstellung in kleinem Maßstab wäre dann ein Qualitätsumschlag in eine Linearsignatur angebracht.

Wenigstens bei Flüssen ist es allerdings etabliert, zusätzlich zur Fläche auch eine Mittellinie in OpenStreetMap zu erfassen. Daher stellen diese Fälle in der Praxis kein Problem dar und sind für diese Arbeit nicht weiter interessant.

3.1.6 Vegetationsgrenzen entlang von Verkehrswegen

Ein Sonderfall der in Abschnitt 3.1.5 beschriebenen langen und schmalen Flächen sind Waldschneisen. Im Zuge der immer detaillierteren Erfassung nicht nur des Wegenetzes, sondern auch der Vegetation kommt es vor, dass zusätzlich zu einem durch den Wald führenden Weg auch die sich durch den Weg ergebende Schneise im Wald erfasst wird, indem die Waldfläche in OpenStreetMap in zwei Flächen links und rechts des Wegs aufgeteilt wird (Abbildung 29).

Einerseits ermöglicht dies eine präzisere Abbildung der Wirklichkeit, indem die Breite der Schneise – welche womöglich die Breite des Wegs übersteigt – modelliert werden kann. Andererseits werden große Wälder ohnehin gerne in mehrere kleinere Flächen aufgeteilt, um das Arbeiten mit den Geodaten zu vereinfachen. Schneisen bieten sich dabei als natürliche Stelle zum Aufteilen an.

Abbildung 29: Waldbegrenzung in dieser OpenStreetMap-Karte dargestellt durch grüne Linie (Oslo, Zoom 16) [All17]

Abbildung 30: in kleinerem Maßstab Waldschneisen durch grüne Linien unangemessen betont (Oslo, Zoom 12) [All17]

Je nach den verwendeten Zeichenregeln werden solche Schneisen in OpenStreetMap-Karten jedoch stärker betont als wünschenswert (Abbildung 30). Hier wäre eine Erkennung der parallelen Ränder der Schneise hilfreich, um durch Zusammenfassen der Waldfläche das Kartenbild zu verbessern und gleichzeitig zu verhindern, dass durch diese Art der Modellierung entstehende kleinere, von Schneisen umringte Waldparzellen aufgrund von Mindestgrößen automatisch wegfallen.

3.2 Auswahl der in dieser Arbeit zu behandelnden Spezialfälle

Viele der bisherigen Arbeiten zur automatisierten Zusammenfassung haben sich mit Straßen beschäftigt (Abschnitt 2.5). In mehreren dieser Arbeiten betonten die Autoren die Wichtigkeit von Ausgangsdaten mit hoher Qualität. Im Falle von OpenStreetMap ist zu erwarten, dass die Datenqualität für Elemente des Straßennetzes höher ist als die von manch anderen Elementen der OpenStreetMap-Datenbank.

So gibt es Anhaltspunkte für eine Korrelation zwischen Datenqualität und dem Umstand, ob diese Daten in der Standard-Visualisierung dargestellt werden. [cf. NZZ12, 18] Dies ist auch anschaulich klar, denn leicht sichtbare Fehler fallen schneller auf und werden somit auch schneller korrigiert als andere Fehler. Die Standard-Visualisierung der Karte auf `osm.org` berücksichtigt deshalb viele Details des Straßennetzes. Unter anderem werden Name und Nummer von Straßen dargestellt, nicht jedoch beispielsweise von Eisenbahnstrecken.[2]

Hinzu kommt die hohe praktische Relevanz des Straßennetzes, die bereits im Namen „OpenStreetMap“ Ausdruck findet, für die OpenStreetMap-Beitragenden genau wie auch für die Allgemeinbevölkerung. OpenStreetMap-Daten werden nicht mehr nur zur visuellen Darstellung, sondern auch zur Navigation verwendet. Auch dies geht mit einer erhöhten Datenqualität einher. [cf. NZZ12, 17–18] Verbesserte Algorithmen könnten hier besonders vielen Nutzern zugutekommen.

Aus diesen Gründen wird von den zuvor in Abschnitt 3.1 beschriebenen Anwendungsfällen für eine automatisierte Linienzusammenfassung der Fall von **baulich getrennten Richtungsfahrbahnen im Straßenraum** für diese Arbeit ausgewählt (Abschnitt 3.1.2). Solche Straßen sind nahezu allgegenwärtig und die mit deren Darstellung verbundenen Probleme verbreitet.

Die in Abschnitt 3.1.3 beschriebenen Fälle paralleler Wege für unterschiedliche Verkehrsarten bieten sich aufgrund ihrer Komplexität nicht an, solange nicht der einfachere Fall von Richtungsfahrbahnen zufriedenstellend gelöst ist. Es ist jedoch denkbar, dass eine Lösung für baulich getrennte Richtungsfahrbahnen auch auf einige solcher Fälle übertragbar ist. Auch auf mehrgleisige Eisenbahnstrecken könnte das Ergebnis dieser Arbeit übertragbar sein.

2 Die Darstellung von Namen für Eisenbahnstrecken wurde 2017 eingeführt.

4 Algorithmen zur Generalisierung

4.1 Vorüberlegungen

Aus der Menge der zuvor erwähnten bereits existierenden Ansätze hebt sich zunächst die Skelettierung heraus, weil es für sie bereits mehrere funktionale Implementierungen der Zusammenfassung gibt (Abschnitt 2.5.2). Diese haben jedoch laut der Autoren alle mit erheblichen Problemen in Kreuzungsbereichen zu kämpfen, die nur teilweise automatisiert gelöst werden können. Dieser Ansatz erscheint wenig vielversprechend.

Letzteres gilt auch für den in Abschnitt 2.5.3 diskutierten Ansatz zur Konflikterkennung. Zwar wäre eine Kombination mit anderen Ansätzen denkbar. So könnte möglicherweise eine Pufferung der Linienzüge mit anschließender Verschneidung der entstehenden Flächen Informationen über die Linienzüge liefern: Dort, wo sich Schnittflächen bilden, bestehen Konflikte; Konflikte zwischen zwei Linien über große Teile ihrer Länge hinweg wären ein Indiz für Parallelität. Dies löst jedoch nicht das Problem der Zusammenfassung beider Linien (vgl. Abschnitte 2.5.1 und 2.5.2).

Interessanter erscheint der bereits in Abschnitt 2.5.5 erwähnte Gedanke, dass Straßen mit parallelen Richtungsfahrbahnen oft komplex modellierte Kreuzungen haben. Können diese zu jeweils einzelnen Knoten generalisiert werden, dann sind die parallelen Richtungsfahrbahnen im Graphen des Straßennetzes zwei gegenläufig gerichtete Kanten. Anschließend wäre die geometrische Zusammenfassung dieser Kanten möglicherweise einfach.

Der von Thom beschriebene Ansatz zur Generalisierung der OS MasterMap klingt vielversprechend (Abschnitt 2.5.4). Er kombiniert Ideen aus der visuellen Wahrnehmung *(strokes)* und der Graphenanalyse, um Richtungsfahrbahnen zusammenzufassen. Den Erfolg seiner Methode schreibt er jedoch unter anderem der gründlichen Klassifikation seiner Ausgangsdaten nach Bauart der Straße zu (einbahnig/zweibahnig usw.). [cf. Tho05, 14] Diese existiert in dieser Form nicht in OpenStreetMap, so dass seine Methode nicht direkt übertragbar erscheint.

Zwar kann für Autobahnen davon ausgegangen werden, dass sie zweibahnig ausgebaut sind. Seltene Ausnahmen wie etwa die Bundesautobahn 62 zwischen Landstuhl und Pirmasens sollten, wenn sie denn tatsächlich als *highway = motorway* eingetragen sind, das Attribut *oneway = no* tragen und sich so identifizieren lassen. Im übrigen Straßennetz sind solche Aussagen auf Basis der OpenStreetMap-Daten jedoch nicht möglich.

Es ist ohnehin fraglich, wie stark die von Thom eingesetzte Erkennung von *strokes* als möglichst *lange* Linienzüge die Generalisierung im Kontext von OpenStreetMap tatsächlich vereinfachen würde. Zwar sind die einzelnen *ways* in OpenStreetMap wegen ihrer ungleichmäßigen Aufteilung oft nicht gut zur Weiterverarbeitung geeignet. Allerdings wären zur Prüfung auf Parallelität und zur Zusammenfassung auf eine gemeinsame Mit-

tellinie auch möglichst *kurze* Linienfragmente gut geeignet, sofern die Fragmentierung beider Parallelen gleichmäßig ist.

Im folgenden Abschnitt 4.2 wird in allgemeinen Begriffen beschrieben, wie parallele Linienzüge auf Basis einer solchen gleichmäßigen Fragmentierung zusammengefasst werden können. Im Anschluss daran folgt die formale Beschreibung dieser Algorithmen.

4.2 Grundprinzip

Um das Zusammenfassen paralleler Linienzüge vorzubereiten, werden alle *nodes* des einen Linienzugs jeweils einem gegenüberliegenden *node* auf dem parallelen Linienzug zugeordnet. Die Verbindung der Mittelpunkte zwischen den so einander zugeordneten *nodes* ergibt direkt den zusammengefassten Linienzug als Ergebnis der Generalisierung (Abbildung 31).

Dieses Vorgehen vermeidet, dass die in Abschnitt 2.1 besprochene häufige ungleichmäßige Fragmentierung von OpenStreetMap-Linienzügen in mehrere *ways* einen Einfluss auf den Generalisierungsvorgang hat. Aufgrund der nicht immer gleichen Anzahl und Verteilung der *nodes* kommt es vor, dass ein *node* des einen Linienzugs mehreren *nodes* des anderen Linienzugs zugeordnet wird, was jedoch unproblematisch ist.

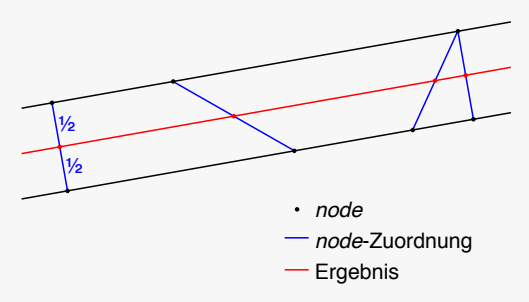

Abbildung 31: Linienzusammenfassung durch Mittelpunktbildung nach Zuordnung gegenüberliegender *nodes*

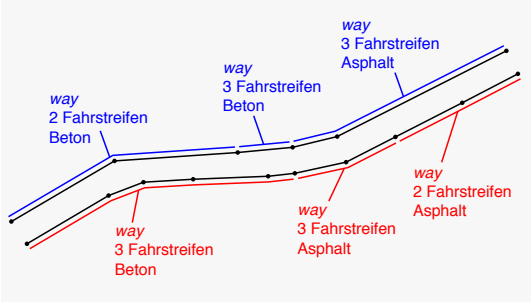

Abbildung 32: ungleichmäßige Fragmentierung paralleler Linienzüge (Beispiel)

Um zwei gegenüberliegende *nodes* einander zuordnen zu können, müssen zunächst die Linien, deren Teil sie sind, als zueinander parallel erkannt werden. Auch hierbei ist die erwähnte ungleichmäßige Fragmentierung in bestimmten Fällen problematisch. Beispielsweise müssten die einzelnen *ways* in Abbildung 32, aus denen die beiden dargestellten Parallelen bestehen, zunächst zu einem längeren Linienzug verknüpft werden, um einen Vergleich zu ermöglichen. Dies entspräche dem in Abschnitt 2.5.4 beschriebenen Ansatz von Thom, der damit zufriedenstellende Resultate erzielte, dabei jedoch die hohe Qualität seiner Ausgangsdaten betonte, welche bei wie für OpenStreetMap von Freiwilligen erfassten Geodaten nicht vorausgesetzt werden kann.

Um diese Problematik zu umgehen, verwendet das in dieser Arbeit vorgestellte Verfahren möglichst *kurze* Liniensegmente anstelle möglichst *langer* Linienzüge. Die Ver-

bindung zweier benachbarter *nodes* in einem *way* als kürzestmögliche lineare Einheit in den OpenStreetMap-Ausgangsdaten (früher als *segment* bezeichnet [cf. RT09, 57]) ist allerdings für einen Vergleich nicht viel besser geeignet als der vollständige *way*, wie aus Abbildung 32 ersichtlich ist. Daher werden die *segments* vor der Analyse auf Parallelität solange immer weiter unterteilt, bis schließlich ein einfacher geometrischer Vergleich möglich ist (Abbildung 33).

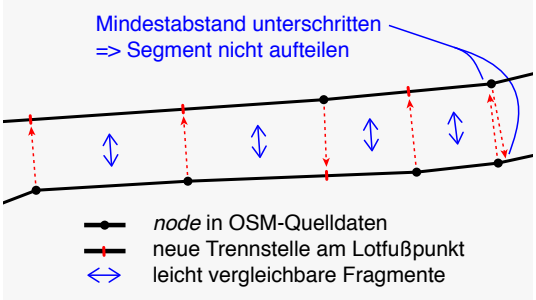

Abbildung 33: Herstellung einer gleichmäßigen Fragmentierung paralleler Linienzüge

4.3 Operationen

Der zuvor beschriebene Lösungsansatz lässt sich vereinfacht als Sequenz von vier Operationen ausdrücken:

1. Segmente unterteilen
2. Analysieren
3. Punktezuordnung
4. Parallelen zusammenfassen

Die folgenden Abschnitte beschreiben jede dieser Operationen im Detail. Verwendete mathematische Symbole werden in Anhang A erklärt.

4.3.1 Segmente unterteilen

In OpenStreetMap-Daten sind seit Oktober 2007 Segmente (Verbindungen von exakt zwei *nodes*) nicht mehr als eigene Objekte enthalten [cf. RT09, 57]. Anhand der aus der Datenquelle eingelesenen Menge aller *ways* wird daher zunächst die Menge aller Segmente ermittelt (Abbildung 34).

Um Verbindungen zwischen zwei *nodes* deterministisch beschreiben zu können, werden diese hier als Kanten in einem gerichteten Graphen betrachtet. Die beiden *nodes* werden im Folgenden als Start und Ende bezeichnet; es gilt:

$$\{\textsc{Start}(s), \textsc{Ende}(s)\} = \textsc{Nodes}(s) \ \forall \ s$$

Geometrisch betrachtet entsprechen *nodes* Punkten, die durch vom Koordinatenursprung ausgehende Vektoren in der euklidischen Ebene definiert sind. Segmente als ge-

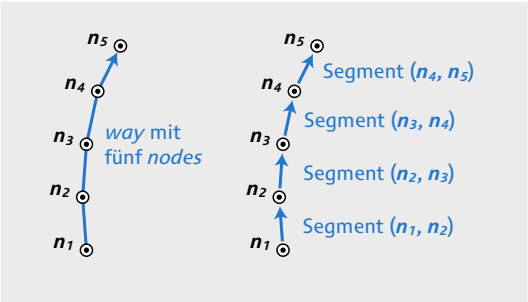

Abbildung 34: vier Segmente in einem *way* (Beispiel)

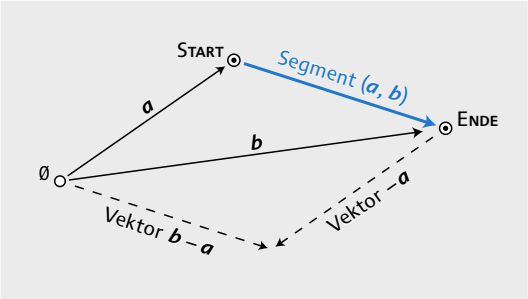

Abbildung 35: einfache Vektorgeometrie an einem Segment demonstriert

richtete Verbindungen zweier solcher Punkte lassen sich dann ebenfalls als Vektoren auffassen und als geordnetes Paar notieren.

So könnten beispielsweise die Ortsvektoren a und b das Segment (a, b) definieren, welches geometrisch zum freien Vektor $b - a$ kongruent wäre (Abbildung 35).

SEGMENTE(W) $\equiv$
Ergebnis sei die Menge aller Segmente (n, n') für Paare von Punkten jedes Linienzugs $w \in W$ derart, dass n Vorgänger von n' im Verlauf von w ist.

Die Segmente werden anschließend durch SPLITTEN derart aufgeteilt, dass ein geometrischer Vergleich leicht möglich wird (Abbildung 33). Die so entstehenden Fragmente sind keine Segmente im Sinne der früheren gleichnamigen OpenStreetMap-Objekte, da der Punkt, an dem sie zerteilt wurden, keinen *node* in OpenStreetMap darstellt. Im Folgenden werden sie dennoch vereinfachend Segmenten gleichgestellt, da sie ansonsten dieselben Eigenschaften besitzen.

Aufgeteilt werden Segmente jeweils am FUSSPUNKT eines Lots, das von einem *node* eines anderen Segments gefällt wird. Der gewählte Algorithmus für das SPLITTEN arbeitet geometrisch rekursiv: Alle erzeugten Fragmente werden wieder zur Menge der Segmente hinzugefügt und somit immer weiter aufgeteilt, bis kein FUSSPUNKT mehr gefunden wird.

Relation *wurzel* : Segment → Segment
Anfangswert $wurzel(s) \leftarrow s \; \forall \; s$
SPLITTEN(S) ≡ ▷ Menge aller Segmente S
  $S' \leftarrow S$
  für alle s **mit** $s \in S'$
    für alle n **mit** $n \in \{\text{START}(s), \text{ENDE}(s)\}$
      $T \leftarrow \text{NAHESEGMENTE}(s, S')$
      für alle t **mit** $t \in T \wedge \exists \; f : f = \text{FUSSPUNKT}(n, t)$
        $t_1 \leftarrow (\text{START}(t), f)$
        $t_2 \leftarrow (f, \text{ENDE}(t))$
        $S' \leftarrow \{t_1, t_2\} \cup S' \setminus \{t\}$ ▷ t ersetzen durch Fragmente t_1 und t_2
        $wurzel(t_1) \leftarrow wurzel(t)$
        $wurzel(t_2) \leftarrow wurzel(t)$
  Ergebnis S'

Es ist nicht notwendig, jedes Segment mit allen anderen Segmenten zu vergleichen. Segmente, die zu weit entfernt und somit nicht NAHESEGMENTE sind, können von vornherein als potenzielle Parallelen ausgeschlossen werden. Zur Entscheidung, welche Segmente als „nah“ gelten und damit für Parallelität in Frage kommen, wird für jedes Segment eine HÜLLE gebildet, welche etwas größer als das jeweilige Segment ist, so dass sich die Hüllen von nahe beieinander liegenden Segmenten überschneiden.

NAHESEGMENTE(s, S) ≡ ▷ Menge aller Segmente S
  Ergebnis $\{t \in S \mid \text{HÜLLE}(s) \cap \text{HÜLLE}(t) \neq \varnothing\}$

Als HÜLLE könnte ein Puffer dienen. Aus Gründen der Effizienz bietet es sich jedoch an, die HÜLLE rechteckig im Koordinatennetz anzulegen. Für diesen Fall hängt die Größe der Hülle auch von der Orientierung des Segments ab. Im Ergebnis wird dann später zu sehen sein, dass diagonal zum Koordinatennetz ausgerichtete Segmente bei größeren Abständen zueinander als parallel erkannt werden als solche Segmente, die orthogonal zum Koordinatennetz ausgerichtet sind.

Um die Korrektheit der Algorithmen sicherzustellen, ist aus diesen Gründen in der späteren ANALYSE auf Parallelität eine erneute Prüfung des tatsächlichen Abstands zweier zu vergleichender Segmente notwendig. Diese kann dann jedoch auf NAHESEGMENTE beschränkt werden und ist dementsprechend billig (vergleiche Abschnitt 4.3.2).

Weiterhin hängen diese Erkennungsabstände vom gewählten Koordinatennetz ab. Würde beispielsweise mit geographischen Koordinaten gearbeitet, so variiert das Verhältnis der Erkennungsabstände in Ost/West- und in Nord/Süd-Richtung mit der geographischen Breite. Für geometrische Operationen in der euklidischen Ebene bieten sich jedoch nur winkeltreue Abbildungen mit geringen Maßstabsfehlern im Anwendungsgebiet an. [cf. Sny87, 20]

Hülle(s) ≡

Ergebnis sei die nach allen Kardinalrichtungen um $\eta/2$ vergrößerte rechteckige Hülle um das Segment s, wobei η der festzulegende Höchstabstand sei, für den zwei Segmente als „nahe“ gelten dürfen.

Das Aufteilen in Fragmente erfolgt jeweils am Fusspunkt (Abbildung 36).

Fusspunkt(n, t) ≡

Ergebnis sei der Lotfußpunkt f des Punkts n auf der Geraden t. Liegt f nicht zwischen den Punkten Start(t) und Ende(t) oder liegt f näher an Start(t) oder Ende(t) als die festzulegende Mindestlänge μ eines Fragments, dann gibt es kein Ergebnis.

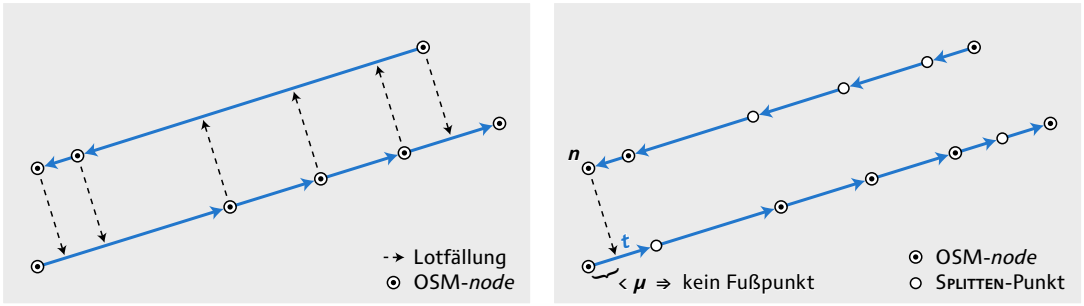

Abbildung 36: Bestimmung der Punkte, an denen Segmente aufzuteilen sind (links vor, rechts nach dem Splitten)

4.3.2 Analysieren

Das Ergebnis der Operation „Segmente unterteilen“ eignet sich zur Analyse auf Parallelität. Hierfür werden Geometrie und Attribute näher verglichen mit dem Ziel, für jedes Segment beidseitig entweder genau ein oder kein anderes Segment als „parallel“ zu kennzeichnen. Die Unterscheidung der Seiten links und rechts kann durch Bestimmung der Winkel zu den *nodes* des jeweils anderen Segments erfolgen (Abbildung 37).

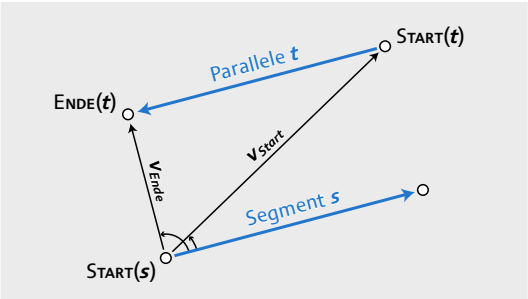

Abbildung 37: Erkennen einer nahen Parallele auf der linken Seite eines Segments

Relation *parallelLinks* : Segment → Segment
Anfangswert $parallelLinks(s) \leftarrow \varnothing \ \forall \ s$
Relation *parallelRechts* : Segment → Segment
Anfangswert $parallelRechts(s) \leftarrow \varnothing \ \forall \ s$
$\textsc{Analyse}(S) \equiv$ ▷ Menge aller *unterteilten* Segmente S
für alle s **mit** $s \in S$
$T \leftarrow \{t \mid t \in \textsc{NaheSegmente}(s, S) \land \textsc{Parallel}(s, t)\}$
$v_{Start} \leftarrow \textsc{Start}(t) - \textsc{Start}(s)$
$v_{Ende} \leftarrow \textsc{Ende}(t) - \textsc{Start}(s)$
$L \leftarrow \{t \in T \mid \sphericalangle(s, v_{Start}) < 0 \land \sphericalangle(s, v_{Ende}) < 0\}$
$R \leftarrow \{t \in T \mid \sphericalangle(s, v_{Start}) > 0 \land \sphericalangle(s, v_{Ende}) > 0\}$
▷ Mengen L, R aller nahen Parallelen links- bzw. rechtsseitig
▷ Winkel seien definiert im Intervall $(-\pi, \pi)$ und im Uhrzeigersinn
für alle l **mit** $l \in L$
falls $0 < \textsc{Distanz}(s, l) \leqslant \textsc{Distanz}(s, l') \ \forall \ l' \in L$
$parallelLinks(wurzel(s)) \leftarrow l$
für alle r **mit** $r \in R$
falls $0 < \textsc{Distanz}(s, r) \leqslant \textsc{Distanz}(s, r') \ \forall \ r' \in R$
$parallelRechts(wurzel(s)) \leftarrow r$
Ergebnis *parallelLinks*, *parallelRechts*

Dabei kommen nur solche Segmente für die Kennzeichnung in Frage, die im Kontext des jeweiligen Anwendungsfalls als Parallel gelten (vergleiche Kapitel 3).

$\textsc{Parallel}(s_1, s_2) \equiv$
Die Segmente s_1 und s_2 seien parallel genau dann, wenn sie:
- ähnlich ausgerichtet sind (z. B. $\alpha := \|\sphericalangle(s_1, s_2)\| < 15°$),
- nahe beieinander liegen (z. B. $\eta := \textsc{Distanz}(s_1, s_2) < 40\,\text{m}$),
- nicht so sehr zueinander versetzt sind, dass sie mehr hintereinander als nebeneinander liegen,
- sich nicht überkreuzen,
- Teile von *ways* mit dem gleichen Wert für *highway* = ∗ sind und
- Teile von *ways* mit dem gleichen Wert für *ref* = ∗ sind.

Dass die miteinander verglichenen Segmente nahe beieinander liegen, wird in der Analyse bereits durch die Beschränkung auf NaheSegmente sichergestellt. Welche Segmente als „nah“ gelten, ist jedoch von der Definition der Hülle abhängig, für die Überlegungen zur Effizienz eine Rolle spielen (vergleiche Abschnitt 4.3.1).

Aus der Menge aller als Parallel geltenden Segmente auf einer Seite wird das am nächsten gelegene als Parallele ausgewählt.[1] Auf welche Weise dazu die Distanz zwischen zwei Segmenten bestimmt wird, kann vom speziellen Anwendungsfall abhängen und soll

1 Die Zieldatentypen der Relationen *parallelLinks* und *parallelRechts* können wahlweise auch *Mengen* von Segmenten sein, um *alle* als parallel erkannten Segmente zu sammeln. Die so entstehenden Duplikate würden später automatisch wieder entfallen.

hier nur beispielhaft angegeben werden. So ist eine mögliche Metrik für die Distanz zweier Segmente der Abstand ihrer Mittelpunkte.

$$\textsc{Distanz}(s_1, s_2) \equiv ||\textsc{Mittelpunkt}(s_1) - \textsc{Mittelpunkt}(s_2)||$$

Für den Mittelpunkt eines Segments gilt dabei mit einfacher Vektorgeometrie:

$\textsc{Mittelpunkt}(a) \equiv$
Es seien n_1 und n_2 die beiden Punkte, die a definieren; dann gelte:
Ergebnis $(n_1 + n_2) \cdot \frac{1}{2}$

4.3.3 Punktezuordnung

Das Zusammenfassen einander paralleler Segmente soll durch Zusammenfassen ihrer Start- und Endpunkte erfolgen. Hierzu müssen zunächst die *nodes* der zusammenzufassenden Segmente einander zugeordnet werden. Dies geschieht auf Basis der durch die Analyse ermittelten Kennzeichnungen der Segmente als parallel.

Der Algorithmus NodesZuordnen sucht für jeden *node* beidseitig den jeweils am nächsten gelegenen *node* der als parallel gekennzeichneten Segmente (Abbildung 38).

$\textsc{NodesZuordnen}(S) \equiv$
$\mathcal{Z} \leftarrow \{\}$
für alle s **mit** $s \in S$
für alle t **mit** $t \in \{parallelLinks(s), parallelRechts(s)\}$
für alle n_1 **mit** $n_1 \in \{\textsc{Start}(s), \textsc{Ende}(s)\}$

$$n_2 \leftarrow \begin{cases} \textsc{Start}(t) & \text{für } ||n_1 - \textsc{Start}(t)|| < ||n_1 - \textsc{Ende}(t)|| \\ \textsc{Ende}(t) & \text{sonst} \end{cases}$$

$\mathcal{Z} \leftarrow \mathcal{Z} \cup \{\{n_1, n_2\}\}$
Ergebnis $\mathcal{Z}$

Weil die Orientierung der Zuordnungen keine Rolle spielt, wird hier mit $\{n_1, n_2\}$ ein *ungeordnetes* Paar beschrieben. Dies vermeidet Duplikate.

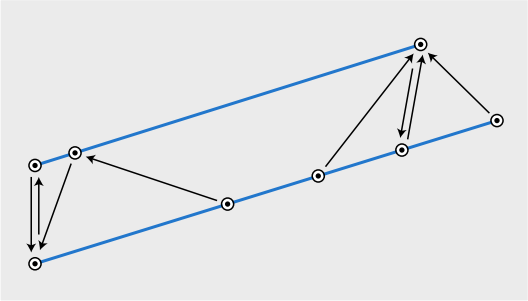

Abbildung 38: Zuordnung gegenüberliegender *nodes* paralleler Linienzüge

4.3.4 Parallelen zusammenfassen

Anhand der Punktezuordnungen können die parallelen Segmente nun zusammengefasst werden. Hierzu wird ein zufällig ausgewähltes Segment als Beginn eines Linienzugs betrachtet. Während diesem Linienzug von Segment zu Segment gefolgt wird, können leicht die Mittelpunkte der Punktezuordnungen als Stützpunkte der Mittellinie zwischen den Parallelen verwendet werden (Abbildung 39).

Relation *zusammengefasst* : Segment → boolean
Anfangswert *zusammengefasst*$(s) \leftarrow$ nein $\forall\ s$
ZUSAMMENFASSEN$(S, \mathcal{Z}) \equiv$ ▷ Segmente S, Punktezuordnungen $\mathcal{Z}$
  $E \leftarrow \{\}$ ▷ Generalisierungsergebnis E
  für alle s **mit** $s \in S \wedge \neg\, zusammengefasst(s)$
    für alle Z **mit** $Z \in \mathcal{Z} \wedge$ START$(s) \in Z$
      $t \leftarrow$ mit dem jeweils anderem *node* von Z verknüpftes Segment,
        ↪ welches auf der gleichen Seite von Z liegt wie s
      wiederhole ▷ versuchen, die nächste Punktzuordnung Z' zu finden
        $s' \leftarrow$ FORTSETZUNG(s, Z, S)
        $t' \leftarrow$ FORTSETZUNG(t, Z, S)
        $Z'_1 \leftarrow \{$NODES$(s') \cap Z,$ NODES$(t') \setminus Z\}$
        $Z'_2 \leftarrow \{$NODES$(t') \cap Z,$ NODES$(s') \setminus Z\}$
        $Z'_3 \leftarrow Z'_1 \cup Z'_2 \setminus Z$
        falls $\exists\ Z' : \{Z'\} = \{Z'_1, Z'_2, Z'_3\} \cap \mathcal{Z}$ ▷ Mittellinie bilden
          $E \leftarrow E \cup \{($MITTELPUNKT$(Z),$ MITTELPUNKT$(Z'))\}$
          $zusammengefasst(s) \leftarrow$ ja
          $zusammengefasst(t) \leftarrow$ ja
          $s \leftarrow s',\ t \leftarrow t',\ Z \leftarrow Z'$ ▷ weiterschalten
      bis $\nexists\ Z'$
Ergebnis $E \cup \{u \in S \mid \neg\, zusammengefasst(u)\}$

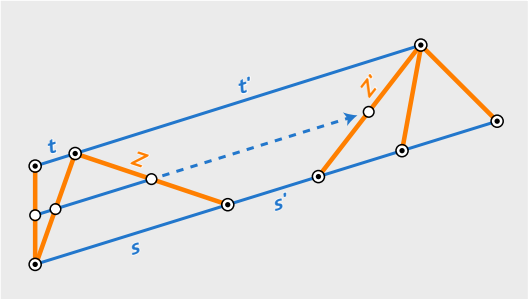

Abbildung 39: Bilden der Mittellinie als Generalisierungsergebnis auf Basis der Punktezuordnungen

Dabei lässt sich dasjenige Segment, welches jeweils als FORTSETZUNG eines Linienzugs verwendet wird, wie folgt ermitteln:

$$\begin{aligned}
&\textsc{Fortsetzung}(s, Z, S) \equiv && \triangleright \text{ Segment } s\text{, Punktezuordnung } Z \\
&\quad S' \leftarrow \{s' \in S \mid s' \neq s \wedge \textsc{Nodes}(s) \cap Z \subset \textsc{Nodes}(s')\} \\
&\quad \textbf{Ergebnis} \begin{cases} s' \in S' & \text{für } |S'| = 1 \\ \varnothing & \text{sonst} \end{cases} && \triangleright \text{ nur eindeutige Fortsetzungen}
\end{aligned}$$

Zur Fortführung der Mittellinie werden jeweils drei mögliche Paare von Punkten betrachtet (Z'_1, Z'_2 und Z'_3 in Abbildung 40). Nur dann, wenn genau eines von ihnen gleichzeitig einer nach der ANALYSE identifizierten Punktezuordnung entspricht, wird diese zur Fortführung der Mittellinie verwendet.

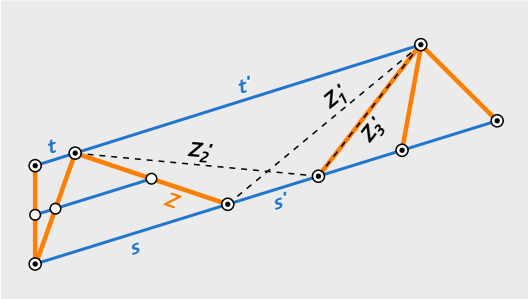

Abbildung 40: Auffinden der nächsten Punktezuordnung zur Fortführung der Mittellinie (im Beispiel Z'_3)

Die obenstehende Definition des Algorithmus zum ZUSAMMENFASSEN ist insofern vereinfacht, als dass sie nicht zur Bildung eines durchgehenden Linienzugs führt, sondern lediglich alle Segmente der Mittellinie sammelt. Dies kann unschwer bei der Implementierung in Software entsprechend berücksichtigt und angepasst werden.

Ebenso berücksichtigt die Definition weder Kreuzungen noch den Übergang eines einzelnen Linienzugs in zwei parallele Linienzüge. In diesen Fällen wird die Generalisierung unterbrochen.

Auch der Fall von mehr als zwei Parallelen bleibt der Verständlichkeit der Definition zuliebe unberücksichtigt. Dies muss bei der Implementierung bedacht werden, da der Begriff „als zusammengefasst markiert“ sonst nicht eindeutig ist. Der hier beschriebene Algorithmus fasst n parallele Linienzüge zu $n-1$ parallelen Linienzügen zusammen, die jeweils mittig zwischen zwei Parallelen in den Ausgangsdaten liegen. Eine Erweiterung zur Zusammenfassung einer beliebigen Anzahl von Parallelen auf genau einen Linienzug ist über die Punktezuordnungen möglich, was jedoch in dieser Arbeit nicht weiter betrachtet wird.

4.4 Gesamtüberblick

Setzt man die beschriebenen Algorithmen wie vorgesehen zusammen, so ergibt sich:

GENERALISIERUNG(W) ≡
  $S \leftarrow$ SEGMENTE(W)
  $S' \leftarrow$ SPLITTEN(S) ▷ Abschnitt 4.3.1
  ANALYSE(S') ▷ Abschnitt 4.3.2
  $\mathcal{Z} \leftarrow$ NODESZUORDNEN(S) ▷ Abschnitt 4.3.3
  $E \leftarrow$ ZUSAMMENFASSEN($S, \mathcal{Z}$) ▷ Abschnitt 4.3.4
  Ergebnis E

Je nach Anwendungsfall sind Nachbearbeitungen und weitere Generalisierungsvorgänge sinnvoll. In Frage kommen insbesondere die Verknüpfung von aneinander anschließenden Linienzügen sowie eine Formvereinfachung. Entsprechend der Aufgabenstellung wird darauf hier nicht weiter eingegangen.

Der Rechenaufwand der gesamten GENERALISIERUNG lässt sich nicht exakt angeben, da er von der Geometrie der Eingangsdaten abhängt. So findet das SPLITTEN nur dann statt, wenn NAHESEGMENTE passende FUSSPUNKTe haben, an denen sie aufgeteilt werden können. Eine grobe Abschätzung des Rechenaufwands ist jedoch möglich:

In erster Näherung kann davon ausgegangen werden, dass für jeden Punkt genau ein FUSSPUNKT auf einem anderen (möglicherweise parallelen) Segment gefunden wird, an dem dieses jeweils geteilt wird, dass weitere FUSSPUNKTe jedoch nicht gefunden werden. Die Größe der Menge von Segmenten würde damit durch das SPLITTEN verdoppelt:

$$|S'| \approx 2 \cdot |S|$$

Tatsächlich werden für *real world*–Daten viele FUSSPUNKTe, an denen Segmente aufgeteilt werden, bei erneuter Lotfällung zum Auffinden weiterer FUSSPUNKTe und damit zu zusätzlichen Teilungen führen, insbesondere innerstädtisch und in der Nähe von Straßenkreuzungen. Jedoch werden gleichzeitig auch viele Punkte gar keine FUSSPUNKTe haben und somit auch nicht zu Teilungen führen, insbesondere in ländlichen Gebieten, wo es insgesamt weniger Straßen und seltener Richtungsfahrbahnen gibt. Da ländliche Gebiete insgesamt in der Fläche überwiegen, kann angenommen werden, dass $|S'|$ jedenfalls im durchschnittlichen Fall nicht wesentlich größer als $2 \cdot |S|$ ist.

Diese Überlegung bestätigt sich anhand der Betrachtung einer einfachen Autobahn mit zwei Richtungsfahrbahnen: Hierbei liegen in den Eingangsdaten oftmals Stützpunkte einander gegenüber. In diesen Fällen werden meist keine FUSSPUNKTe auf der Gegenfahrbahn gefunden, weil deren Definition eine Mindestlänge μ für geteilte Segmente fordert. Demnach wäre hier $2 \cdot |S|$ als obere Schranke für $|S'|$ im durchschnittlichen Fall zutreffend.

Es seien nun $n := |S|$ die Anzahl der Segmente in den Eingangsdaten, ν die durchschnittliche Anzahl der jeweils „nahen“ Segmente und ψ die durchschnittliche Anzahl

der Punktezuordnungen je Punkt. Für die Abschätzung des gesamten Rechenaufwands ergibt sich mit Zählung der geschachtelten Iterationen: [cf. wp13b]

$$\begin{aligned} \textsc{Splitten} &\in \mathcal{O}(2n \cdot 2 \cdot \nu) = \mathcal{O}(n \cdot \nu) \\ \textsc{Analyse} &\in \mathcal{O}(2n \cdot 2\nu^2) = \mathcal{O}(n \cdot \nu^2) \\ \textsc{NodesZuordnen} &\in \mathcal{O}(n \cdot 2 \cdot 2) = \mathcal{O}(n) \\ \textsc{Zusammenfassen} &\in \mathcal{O}(n \cdot \psi) \end{aligned}$$

Jedoch sind ν und ψ im Verhältnis zu n vernachlässigbar, weil sie regelmäßig sehr klein sind. So werden durch die Beschränkung der Hülle auf die unmittelbare Umgebung des jeweiligen Segments nur wenige Segmente als „nah“ betrachtet; es ist abhängig von der Struktur der Eingangsdaten eine Zahl im unteren zweistelligen Bereich zu erwarten. Weiterhin sind mehr als zwei Punktezuordnungen im Wesentlichen nur im (seltenen) Fall von mehr als zwei erkannten Paralellen zu erwarten. Mit

$$\{\nu, \psi\} \subset \mathrm{o}(n)$$

zeigen somit im durchschnittlichen Fall alle Teile der Generalisierung und folglich auch die Generalisierung insgesamt ein lineares Wachstum des Rechenaufwands:

$$\textsc{Generalisierung} \in \mathcal{O}(n)$$

Im *schlechtesten* Fall liegen alle Segmente so nah beieinander und haben so ungünstige Winkel, dass *alle* Segmente solange aufgeteilt werden, bis *jede* weitere Teilung zur Unterschreitung der Mindestlänge μ führen muss. An diesem Punkt bricht das Splitten ab, weil die Einhaltung der Mindestlänge definitionsgemäß Bedingung für die Existenz eines Fusspunkts ist. Die Größe der Menge S' und damit der weitere Rechenaufwand hängen dann von der geometrischen Länge der Segmente und der Wahl von μ ab. Eine solche Situation ist allerdings in *real world*–Daten für ein Straßennetz wenig wahrscheinlich. Sie könnte nur dann überhaupt auftreten, wenn der Höchstabstand η für NaheSegmente zu groß gewählt wurde (vgl. Abschnitt 4.3.1).

Selbst in diesem schlechtesten Fall terminieren die Algorithmen. Für das Splitten wurde dies soeben begründet; für alle anderen imperativ beschriebenen Algorithmen ist es trivial zu erkennen, da diese nur über endliche unveränderliche Mengen iterieren.

Die Algorithmen arbeiten jedoch nicht deterministisch: Abhängig von den Eingabedaten ist anzunehmen, dass je nach Reihenfolge der Iteration das Splitten und damit auch die Analyse unterschiedliche Ergebnisse liefern. Es ist jedoch nicht zu erwarten, dass diese Varianzen großen Einfluss auf das Ergebnis haben.

Auf eine formale Festlegung der Kriterien für die Korrektheit der Generalisierung wurde verzichtet. Um nachzuweisen, dass die entwickelten Algorithmen funktionieren, wurden diese stattdessen ausführbar in Software implementiert. Dies beschreibt Kapitel 5. Im Anschluss daran werden in Kapitel 6 das Generalisierungsergebnis und die Praxistauglichkeit der Algorithmen für unterschiedliche Fälle diskutiert.

5 Implementierung

5.1 Entwicklungsumgebung

Zur Umsetzung der entwickelten Algorithmen in Software wurde die Plattform Java verwendet. Die Wahl von Java erfolgte neben der Vertrautheit des Verfassers mit dem zugehörigen *framework* auch aufgrund zu erwartender Effizienzvorteile von kompiliertem Code gegenüber Skriptsprachen wie etwa Perl.

Java als imperative Sprache erlaubt es nicht, Algorithmen mit dem gleichen Grad an Abstraktion zu beschreiben wie zuvor in Kapitel 4.3 geschehen. Dort konnten zugunsten einer vereinfachten Beschreibung praktische Erwägungen wie der Bedarf an Rechenzeit und Speicherplatz teilweise hintenanstehen. Bei der Implementierung in Java sollten hingegen sorgfältig solche Datenstrukturen gewählt werden, die eine effiziente Ausführung erlauben, und die Algorithmen soweit nötig entsprechend angepasst werden. Das Ergebnis wird in Abschnitt 5.3 beschrieben.

Während der Entwicklung wurde versucht, so viel existierenden Code in Form von *frameworks* und anderen Programmbibliotheken wiederzuverwenden wie möglich. Diese Bestrebung verursachte Probleme, wie auch später in Abschnitt 5.4.2 dargelegt wird. Bei der Entwicklung kamen zuletzt die folgenden Plattformen und Bibliotheken zum Einsatz:

- Mac OS X 10.11.6
- Java™ Standard Edition JDK 8 Update 162
  `http://www.oracle.com/technetwork/java/javase/downloads/`
- Apache Ant 1.10.2 `https://ant.apache.org/`
- args4j 2.33 `http://args4j.kohsuke.org/`
- GeoTools 18.0 `http://www.geotools.org/`
- TestNG 6.8 `http://testng.org/`
- GDAL 2.2.2 `http://www.gdal.org/`

Ursprünglich wurden ältere Softwareversionen verwendet. Die nötigen Anpassungen an die hier genannten aktuellen Versionen waren gering, Kompatibilität mit den älteren Versionen ist jedoch gegenwärtig aufgrund von Änderungen in GeoTools nicht mehr vollständig gegeben. Der Code ist dabei noch immer konform zur Syntax von Java 6. [cf. GJSB05]

Der folgende Abschnitt erläutert einige Überlegungen, die bei der Auswahl der Bibliotheken relevant waren.

5.2 Systemarchitektur

Für ein gut funktionierendes Gesamtpaket sind vor der Umsetzung von vorgegebenen Algorithmen in ausführbarem Code einige praktische Aspekte zu bedenken.

Da die entwickelten Algorithmen ein für das jeweilige Anwendungsgebiet optimiertes kartesisches Koordinatennetz verlangen (vgl. Abschnitt 4.3.1), sind die aus der OpenStreetMap-Datenbank stammenden Eingangsdaten nach Länge und Breite vor der Verarbeitung in einem geeigneten Kartennetzentwurf abzubilden. Hierzu bietet sich unter anderem die UTM-Abbildung an *(Universal Transverse Mercator),* welche einen querachsigen Schnittzylinder winkeltreu abbildet und zur Minimierung des Maßstabsfehlers in jeweils 6° breite Zonen unterteilt. [cf. Sny87, 57–58] Zur Vereinfachung wird die Implementierung zunächst auf eine einzelne UTM-Zone beschränkt.

Weiterhin stellt sich die Frage nach der Benutzerschnittstelle der Software. Die gewählte Plattform Java bietet verschiedene Möglichkeiten, graphische Benutzeroberflächen (*graphical user interface,* GUI) zu gestalten. Denkbar wäre beispielsweise eine Integration in den weit verbreiteten OpenStreetMap-Editor JOSM als *plug-in.* Das Entwickeln und Debuggen einer GUI-Anwendung wäre jedoch zeitaufwändig. Ohnehin würde es ein modularer Aufbau der Anwendung erlauben, zu einem späteren Zeitpunkt eine optionale GUI zu ergänzen. Aus diesen Gründen soll im Rahmen dieser Arbeit auf eine GUI zugunsten einer einfachen textbasierten Kommandozeilen-Schnittstelle (*command-line interface,* CLI) verzichtet werden. Zum Parsen der CLI-Parameter bieten sich einfache Bibliotheken wie etwa args4j an.

Zur Verarbeitung beliebiger Geodaten müssen diese aus Datenspeichern eingelesen und wieder ausgegeben werden können. Um Entwicklungszeit zu sparen, soll hierfür nach Möglichkeit ein existierendes *framework* genutzt werden.

Die in Abschnitt 4.3.1 gegebene Definition für NaheSegmente kann leicht mit Hilfe eines R-Baums als räumlicher Index umgesetzt werden. Die Hülle ist dabei das minimal umgebende Rechteck eines Blatts im R-Baum. Nachdem die OpenStreetMap-Eingangsdaten vor der Generalisierung vollständig bekannt und somit statisch sind, bietet sich der Einsatz eines gepackten R-Baums an [cf. RSV02, 255-256]. Ein solcher Baum wird von der JTS Topology Suite (JTS) angeboten, welche vom *framework* GeoTools als Implementierung des Geometriemodells verwendet wird.

GeoTools bietet außerdem Möglichkeiten zur Ein- und Ausgabe (*input/output,* I/O) von Geodaten in zahlreichen Formaten einschließlich der Transformation von Koordinatensystemen. Zwar wird das native OpenStreetMap-XML-Format ebensowenig unterstützt wie das neuere Protocol Buffer Binary Format (PBF). Das verbreitete Format ESRI Shapefile wird jedoch unterstützt. Die Geofabrik stellt aktuelle OSM-Auszüge öffentlich als Shapefile bereit. Alternativ lassen sich Shapefiles leicht mit gängiger Software wie etwa GDAL aus anderen Formaten erzeugen. GDAL ermöglicht dabei auch den Zuschnitt auf ein abgegrenztes Untersuchungsgebiet, was sich für Testzwecke anbietet.

Zusammengesetzt ergibt sich aus den besprochenen Aspekten die in Abbildung 41 dargestellte Architektur: Dem Anwender steht eine Kommandozeilen-Schnittstelle (CLI) zur Verfügung, welche die Bibliothek args4j zum Parsen der Parameter sowie selbst

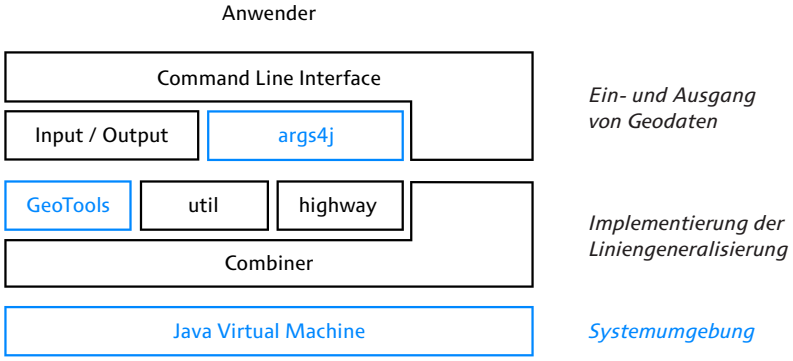

Abbildung 41: Komponenten der entwickelten Software (schwarz: eigene Entwicklung, blau: benutzte Bibliothek)

entwickelte Routinen zur Ein- und Ausgabe der Geodaten verwendet (welche ihrerseits auf GeoTools zurückgreifen).

Die CLI steuert damit die eigentliche Liniengeneralisierung (hier als *Combiner* bezeichnet), welche ihrerseits neben dem *framework* GeoTools noch einige selbst entwickelte Hilfsmodule verwendet, die nicht vom *Combiner* abhängig sind und deshalb als separates Softwarepaket dargestellt werden („util“). Schließlich sitzt zwischen *Combiner* und CLI ein mit „highway“ bezeichnetes Paket, welches versucht, die Logik des in Abschnitt 3.2 ausgewählten Spezialfalls an einem einzigen Ort zu bündeln, um die Anpassung auf andere Spezialfälle zu erleichtern.

In den folgenden Abschnitten wird näher auf die Implementierung der Liniengeneralisierung insbesondere im *Combiner* eingegangen.

5.3 Datenstrukturen

Aus Gründen der Übersichtlichkeit werden in den folgenden Abschnitten nur wesentliche Aspekte und wichtige Entscheidungen behandelt. Für Detailfragen sei auf die Dokumentation der Programmierschnittstelle verwiesen, welche in englischer Sprache als Teil der Softwareentwicklung im Javadoc-Format angelegt wurde und sowohl im Java-Quellcode als auch im HTML-Format zugänglich ist (`http://arne.johannessen.de/thesis`).

Anhang B löst Bezeichner aus Abschnitt 4.3 zu Bezeichnern im Quellcode auf.

5.3.1 Grundlegendes Geometrie-Modell von GeoTools

Zur Umsetzung der in Abschnitt 4.3 definierten geometrischen Algorithmen werden zunächst Strukturen zur Repräsentation der grundlegenden Datentypen benötigt. Im Kontext dieses Projekts sind dies:

1. Punkt (OpenStreetMap-*node*)
2. Segment (Verbindung von zwei Punkten als Teil eines Linienzugs)
3. Linienzug (OpenStreetMap-*way*)

Das *framework* GeoTools benutzt dazu die JTS Topology Suite, die wie zuvor beschrieben ohnehin wegen des R-Baums für NaheSegmente zum Einsatz kommen soll. JTS bietet mit der `Geometry`-Hierarchie Datenstrukturen nach ISO 19125-1 an, die grundsätzlich auch für die Implementierung der Algorithmen aus Abschnitt 4.3 geeignet wären. [cf. jts15a]

Die Algorithmen Splitten und Analyse ordnen den Segmenten allerdings zusätzliche Attribute zu (*wurzel* bzw. *parallelLinks* / *parallelRechts*). Dies unterstützt JTS zwar in Form von *user data objects*. Weil jedoch die Arbeit damit eher umständlich ist, wäre dies nur sinnvoll, wenn die Nutzung der JTS-Datenstrukturen andere Vorteile bietet.

Dies ist jedoch nicht der Fall. Im Gegenteil passen die JTS-Datenstrukturen nicht gut auf die definierten Algorithmen: Diese beschränken sich zu einem großen Teil auf Segmente aus exakt zwei Punkten, während JTS von längeren Linienzügen ausgeht. Die Algorithmen machen sich außerdem zunutze, dass sich Segmente direkt als Vektoren interpretieren lassen (vergleiche Abschnitt 4.3.1). Routinen zur Vektor-Mathematik bietet JTS zwar grundsätzlich an, aber wichtige Teile davon sind nicht spezifiziert. Beispielhaft sei die Klasse `jts.math.Vector2D` genannt, deren Methode `angle` offenbar Winkel berechnet, jedoch in keiner Weise dokumentiert ist. [cf. jts15c] Die Definition der Analyse auf Parallelität verlangt jedoch einen bestimmten Wertebereich für Winkel von Vektoren. Obwohl sich die derzeitige Implementierung von `angle` leicht anhand des Quellcodes von JTS ermitteln ließe, ist die Methode nicht zuverlässig verwendbar, weil ohne Dokumentation bereits unklar bleibt, ob die derzeitige Implementierung überhaupt korrekt ist.

Hinzu kommt, dass auch die von JTS angebotenen Operationen zur Manipulation geometrischer Daten – abgesehen vom räumlichen Index – hier wenig hilfreich sind. Ein direktes Verwenden der bereitgestellten JTS-Datenstrukturen erscheint folglich insgesamt betrachtet nicht sinnvoll zu sein. Stattdessen wurde ein eigenes Datenmodell entwickelt, das im Folgenden beschrieben wird.

JTS bietet die Möglichkeit, das *framework* zusammen mit eigenen Datenstrukturen zu benutzen, wenn diese bestimmte Schnittstellen anbieten. [cf. jts03, 11] Dies käme hier durchaus in Betracht, damit die eigenen Datenstrukturen zukünftig von Dritten unmittelbar mit JTS genutzt werden können. Beispielsweise könnten so zusammengefasste Linienzüge als Ergebnisse der hier entwickelten Software direkt mit JTS einer Formvereinfachung unterzogen werden. Weil dies jedoch nicht Bestandteil dieser Arbeit ist und ein Implementieren jener Schnittstellen keinen direkten Nutzen hat, wurde darauf zunächst verzichtet. Es ließe sich für eine zukünftige Version leicht ergänzen.

5.3.2 Eigenes grundlegendes Geometrie-Modell

Es erwies sich also als notwendig, auf dieser Basis ein neues Geometriemodell zu entwickeln. Ein zusammengesetztes Diagramm des Ergebnisses ist Anhang C zu entnehmen. Vereinfacht auf die grundlegenden Geometrie-Datentypen ergibt sich das in Abbildung 42 gezeigte Datenmodell. Die Segmente sind – wie in Abschnitt 4.3.1 beschrieben – zunächst nicht in den Eingangsdaten vorhanden; sie daraus durch Segmentierung herzustellen, ist jedoch einfach.

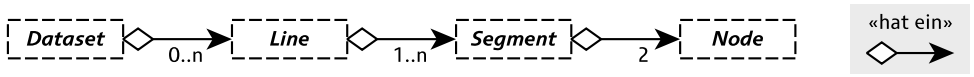

Abbildung 42: Datenmodell für die Geometrie

Die hier gezeigten Datentypen wurden als Interfaces (formale Schnittstellendefinitionen) umgesetzt, um ihre eventuelle Wiederverwendung zu erleichtern. [cf. GHJV95, 18] Abbildung 43 zeigt die bei der Implementierung dieses Datenmodells entstandene Klassenstruktur.

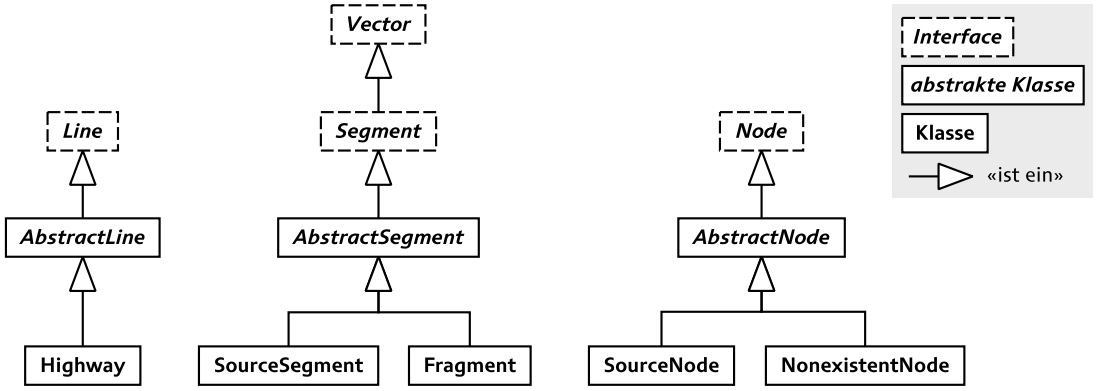

Abbildung 43: Strukturdiagramm der für das Splitten relevanten Klassen

Die Anzahl der jeweils das Interface implementierenden Klassen ist in diesem Projekt klein, so dass es sich anbietet, die jeweiligen Gemeinsamkeiten in abstrakten Klassen zusammenzufassen, um Redundanzen zu vermeiden. Diese Klassen gehören daher zu den umfangreicheren im Projekt.

Wie in Abschnitt 4.3.1 erläutert, existiert derjenige *node*, an dem Segmente beim Splitten zerteilt werden, nicht in den Eingangsdaten. Dementsprechend wird unterschieden zwischen `SourceNode` (aus den OpenStreetMap-Quelldaten kommend) und `NonexistentNode` (beim Splitten entstehend). Beide implementieren jedoch die gemeinsame Schnittstelle für den Typ `Node`. Zu den Gemeinsamkeiten beider Arten von `Node`s gehören in erster Linie deren Koordinaten, welche folglich die Klasse `AbstractNode` verwaltet und an die beiden konkreten Implementierungen vererbt.

In gleicher Weise fasst `AbstractSegment` Gemeinsamkeiten solcher Segmente, die direkt aus den Eingangsdaten stammen (`SourceSegment`), und solcher Segmente, die erst durch Splitten eines anderen Segments entstehen (`Fragment`), zusammen. In Abschnitt 4.3 wurden solche Segmente zur Vereinfachung einander gleichgestellt. Für derartige Fälle schlagen Gamma et al. den Einsatz des Kompositum-Entwurfsmusters vor: „Use the Composite pattern when [...] you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.“ [GHJV95, 164] Im Kompositum-Muster werden

Objekte zu Baumstrukturen aus individuellen Objekten (Blättern) und Komposita von Objekten zusammengefügt.

Im Kontext des Kompositum-Musters ist jedoch die Besonderheit zu berücksichtigen, dass hier einige Blätter aus den Eingangsdaten stammen (`SourceSegments`), andere nicht (`Fragments`). Gleiches gilt auch für Komposita, so dass sich in einer nur auf Vererbung bestehenden Klassenstruktur sofort eine Wucherung aus größtenteils redundanten Klassen ergäbe. Für solche Fälle schlagen Gamma et al. vor, das Brücken-Muster *(Bridge pattern)* einzusetzen, mit dem die Redundanz durch Delegation vermieden werden kann. [cf. GHJV95, 153]

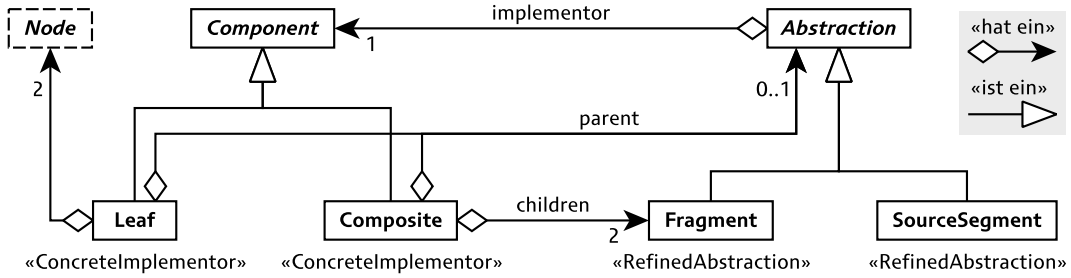

Abbildung 44: Strukturdiagramm einer Kombination aus *Composite pattern* und *Bridge pattern* [cf. GHJV95, 153, 164]

Eine Kombination aus Kompositum und Brücke wäre zwar möglich, würde jedoch zu einer recht unübersichtlichen Struktur führen. Gamma et al. betonen, dass Delegation unweigerlich Komplexität mit sich bringt und nur dann sinnvoll ist, wenn andere Vorteile überwiegen („Delegation is a good design choice only when it simplifies more than it complicates.“ [GHJV95, 21]). Dies wäre hier offensichtlich nicht der Fall (Abbildung 44).

Ihre Empfehlung, Delegation möglichst nur in Form von standardisierten Mustern zu verwenden, [cf. ibid.] scheint hier auch wegen einer weiteren Besonderheit nicht anwendbar zu sein: Alle Objekte im `AbstractSegment`-Kompositum sind veränderbar (*mutable*), damit bei der Umwandlung von Blättern (die jeweils ein Segment repräsentieren) in Komposita (die zwei oder mehr Segmente repräsentieren) beim Splitten keine großen Kosten entstehen. Bloch betont die erheblichen Vorteile von *immutability* (*nicht* veränderbarer Objekte), weist aber auch darauf hin, dass bestimmte komplexe Operationen dann sehr teuer werden können („The performance problem is magnified if you perform a multistep operation that generates a new object at every step, eventually discarding all objects except the final result.“ [Blo01, 67]). Im vorliegenden Kompositum würde *immutability* gerade erfordern, dass bei jedem Splitten der Baum neu aufgebaut wird.

Unterschiedliche Klassen für Blätter und Komposita sind deshalb hier nicht angebracht; vielmehr bietet es sich in Anlehnung an standardisierte Muster an, dass eine gemeinsame Klasse mit einem Attribut *children* sich entweder wie ein Blatt oder wie ein Kompositum verhält, je nachdem, ob es *children* in Form von Fragmenten gibt oder nicht.

Aus dieser Erkenntnis ergibt sich eine vergleichsweise einfache Klassenstruktur (Abbildung 45): `AbstractSegment` implementiert die Gemeinsamkeiten aller Segmente. Dazu

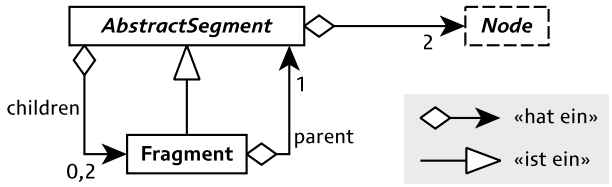

Abbildung 45: Diagramm der anstelle des *Composite pattern* gewählten vereinfachten Klassenstruktur für die beim SPLITTEN entstehenden Fragmente

gehört, dass jedes Segment in genau zwei Fragmente geteilt sein kann, die ihrerseits ebenfalls Segmente sind. Jedes solche Fragment „weiß“ darüber hinaus, aus welchem Segment es abgetrennt wurde (*parent*).

Die Unterschiede zwischen `SourceSegment` und `Fragment` sind zunächst klein. Die in Abschnitt 4.3.1 gegebene Definition von NAHESEGMENTE wurde über den schon erwähnten gepackten R-Baum in JTS implementiert. Dessen Implementierung hat den Nachteil, dass der so erzeugte R-Baum nach dem Packen nicht mehr verändert werden darf. [cf. jts15b] Weil hier ohnehin `SourceSegment` als eigene Klasse existiert, bietet es sich an, die NAHESEGMENTE-Logik nur in `SourceSegment` zu implementieren statt in der Superklasse `AbstractSegment`. Das Packen des R-Baums braucht dann nur genau einmal nach dem SEGMENTIEREN durchgeführt zu werden. Danach werden die `SourceSegments` nicht mehr verändert. Da Abschnitt 4.3.1 ohnehin eine erneute Abstandsprüfung in der späteren ANALYSE vorsieht, stellt diese Abweichung von der Definition der Algorithmen nicht die Korrektheit der Implementierung in Frage, obwohl sie mehr Segmente als „nah“ betrachtet als die formale Beschreibung für NAHESEGMENTE.

5.3.3 Analyse und Anwendbarkeit auf andere Spezialfälle

Die Algorithmen zur ANALYSE erfordern die Ergänzung des Datenmodells aus Abbildung 42 mit den zusätzlichen Attributen *parallelLinks* und *parallelRechts*, in denen das Ergebnis der ANALYSE abgelegt wird. Wie in Abschnitt 4.3 definiert sind dies Attribute des Wurzel-Segments, also der Klasse `SourceSegment`.

Obwohl nach der Auswahl eines bestimmten Spezialfalls der Liniengeneralisierung in Abschnitt 3.2 andere Fälle nicht weiter berücksichtigt wurden, ist eine auch andere Anwendungsfälle zulassende Flexibilität doch Teil der angestrebten Praxistauglichkeit der zu entwickelnden Software. Angesichts der Arbeitsweise der Algorithmen ist es naheliegend, dass sich dies insbesondere in der Analyse auf Parallelität niederschlagen sollte. Die Definition der ANALYSE beschränkt sich auf geometrische Aspekte; für die konkrete Evaluierung zweiter Segmente spielen im Wesentlichen die Definitionen für PARALLEL und DISTANZ eine Rolle. Es wäre daher von Vorteil, wenn diese beiden Definitionen in der Software leicht austauschbar wären.

Dieses Verhalten scheint auf den ersten Blick auf die Beschreibung des Schablonenmethoden-Entwurfsmusters *(Template Method pattern)* zu passen, womit einzelne Schritte verändert werden können, ohne die gesamte Struktur eines Algorithmus zu verändern. Tatsächlich beschränkt sich dieses Muster jedoch darauf, diese einzelnen Schritte von

Unterklassen implementieren zu lassen, was auch in der einleitenden Beschreibung von Gamma et al. schon deutlich wird: „Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.“ [GHJV95, 325] Eine Anwendung an dieser Stelle würde also gerade unter Berücksichtigung der oben beschriebenen Komposition von Segmenten in `SourceSegment`s und `Fragment`s zu einer Wucherung von weiteren Unterklassen führen und wäre daher ungeeignet.

Stattdessen empfehlen Gamma et al. das Besucher-Muster *(Visitor pattern)* einerseits gerade für solche Kompositionen, andererseits gerade dann, wenn wie hier die eigentlichen Datenstrukturen stabil sind, aber doch Flexibilität zur Definition neuer Operationen bieten sollen. [cf. GHJV95, 333, 344] Dabei erlauben die Objekte der bereits etablierten Datentypen, ihnen einen sog. *visitor* als ein unabhängiges Objekt mit definierter Schnittstelle zu übergeben, welches die jeweiligen Operationen implementiert. [cf. GHJV95, 331–335]

In der vorliegenden Software akzeptieren die Segmente den Besuch eines Objekts, das die Schnittstelle `Analyser` implementiert. Dieses Objekt soll die Implementierungen der Algorithmen Parallel und Distanz enthalten. Im Sinne der besprochenen Systemarchitektur braucht somit das Wissen über das Behandeln des in Abschnitt 3.2 gewählten Spezialfalls „baulich getrennte Richtungsfahrbahnen im Straßenraum“ nicht Teil des *Combiners* zu sein, sondern kann vom Client auf andere Themen angepasst werden.

5.3.4 Punktezuordnung

Für Abschnitt 4.3.3 ergibt sich das in Abbildung 46 gezeigte Datenmodell. Die Klasse `NodeMatch` implementiert die Punktezuordnungen. Dies sind entsprechend der formalen Definition Mengen aus exakt zwei gegenüberliegenden *nodes* einander paralleler Segmente. Die Definition als mathematische Menge zur Vermeidung von Duplikaten wird später beim Zusammenfassen die Interpretation des Begriffs „nächste Zuordnung Z'“ erleichtern (vergleiche Abschnitt 4.3.4). Um Objekte der Klasse `NodeMatch` klar als Menge zu kennzeichnen, wurde sie zusätzlich als `Set<Node>` (Menge von `Node`s) im Java Collections Framework deklariert. Dies erleichtert auch die Implementierung der Mengeneigenschaften.

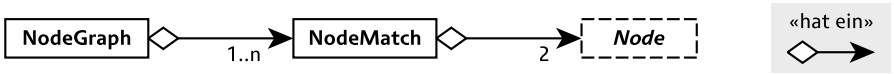

Abbildung 46: Datenmodell für die Punktezuordnung

Da die Analyse für jedes Segment immer jeweils links- und rechtsseitig nach Parallelen sucht, werden beim NodesZuordnen viele Zuordnungen doppelt gefunden, nämlich einmal in jeder Richtung. Erneut vereinfachen die Eigenschaften der mathematischen Menge, hier der Menge aller Zuordnungen $\mathcal{Z}$, die Beschreibung des Algorithmus in Abschnitt 4.3.3. Weil `NodeMatch`-Objekte wie beschrieben die Mengeneigenschaften implementieren, können sie leicht direkt im Java Collections Framework verwendet werden, hier als `Set<NodeMatch>` (Menge von `NodeMatch`-Objekten). Da der Datentyp

`NodeMatch` gleichzeitig die Schnittstelle `Set<Node>` implementiert, hat die Menge aller Zuordnungen implizit den Typ `Set<Set<Node>>`, ist also eine Menge *von Mengen* von *nodes*, was genau der formalen Anforderung in Abschnitt 4.3.3 entspricht.

Punktezuordnungen und Segmente ergeben gemeinsam einen gemischten Graphen des zu generalisierenden Netzes. Als parallel erkannte Segmente sind über deren *nodes* derart einander zugeordnet, dass sich eine gemeinsame Mittellinie als Verknüpfung der Mittelpunkte derjenigen Kanten im Graphen, welche die Punktezuordnungen darstellen, leicht finden lässt. Zur Definition des Graphen genügt die Menge aller Punktezuordnungen, wenn über die einzelnen Punkte die mit ihnen verknüpften Segmente ermittelbar sind.

5.3.5 Liniengeneralisierung

Das Ergebnis der Liniengeneralisierung nach Abschnitt 4.3.4 ist ein Graph aus Linienzügen, von denen einige durch Zusammenfassen entstehen, andere jedoch mangels Parallelen direkt aus den Eingangsdaten übernommen werden. Der beschriebene Algorithmus zum Zusammenfassen ergibt einen möglichst langen Linienzug. Um die spätere Weiterverarbeitung des Generalisierungsergebnisses zu erleichtern, werden hier auch Segmente ohne Parallele zu möglichst langen Linienzügen verkettet.

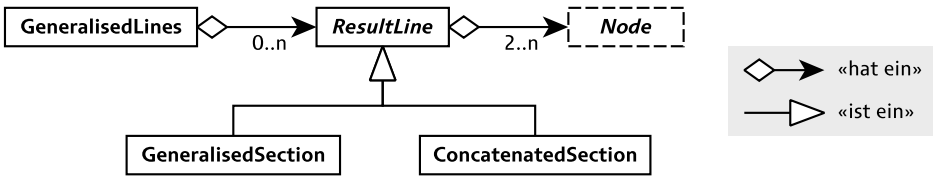

Abbildung 47: Datenmodell für das Zusammenfassen

Entsprechend gibt es zwei unterschiedliche Arten von Linienzügen in den Ergebnisdaten. Dies wird auch vom Datenmodell so abgebildet (Abbildung 47). Die Klassen `GeneralisedSection` und `ConcatenatedSection` enthalten jeweils im Konstruktor Code, der allein anhand der Angabe eines geeigneten Startpunkts einen Linienzug für das Ergebnis bildet. Die bearbeiteten Segmente werden dabei entsprechend markiert, um doppelte Bearbeitungen zu vermeiden.

Die Klasse `GeneralisedSection` enthält den Algorithmus zum Zusammenfassen. `ConcatenatedSection` ist ähnlich aufgebaut, aber wesentlich einfacher, da Punktezuordnungen keine Rolle spielen, sondern nur die jeweils anschließenden Segmente zu berücksichtigen sind. Beide Klassen sind vom Typ `ResultLine`, um dem Client des *Combiners* einen einheitlichen Typ für die Ergebnisdaten liefern zu können.

Die Klasse `GeneralisedLines` ist eine einfache Fassade für die Generalisierung, um deren Einsatz zu erleichtern. [cf. GHJV95, 186]

5.4 Schwierigkeiten bei der Umsetzung

Im Laufe der Implementierung ergaben sich nicht vorhergesehene Probleme und Erkenntnisse. In den folgenden Abschnitten werden diese kurz erläutert.

5.4.1 Eigenschaften der Sprache

Die Entwicklung von der Idee für einen Algorithmus hin zu einer für komplexe Geodaten praxistauglichen Implementierung kann als iterativer Prozess verstanden werden: Unter dem Eindruck der Ergebnisse früher Versionen werden Algorithmus und Implementierung nach und nach verfeinert. Die Wahl von Java als streng typisierte Sprache erschwerte jedoch dieses Vorgehen. Die strenge Typisierung hat den Fokus dieser Arbeit (unter Berücksichtigung des vorgegebenen Umfangs notwendigerweise) ein Stück weit verschoben von Algorithmusentwicklung und -beurteilung zu den Details objektorientierter Programmierung. Dies war so vor Beginn nicht absehbar.

Möglicherweise wäre eine dynamisch typisierte Skriptsprache wie Perl für dieses explorative Vorgehen geeigneter gewesen. Den vermuteten Effizienznachteilen solcher Sprachen (siehe Abschnitt 5.1) steht die einfachere Umsetzung von Änderungen gegenüber.

Weiterhin war das Aufbauen eigener Datenstrukturen für die Arbeit mit Segmenten lehrreich, aber zeitaufwändiger als erwartet. Erwartungsgemäß war dabei jedoch das Java Collections Framework äußerst hilfreich.

Ebenfalls als hilfreich stellte sich die Idee heraus, nicht nur im Algorithmus (Abschnitt 4.3.1), sondern auch in der Typenhierarchie Segmente als Vektoren zu behandeln. Die Implementierung des *Combiners* setzt dies um, indem über Interface-Vererbung alle Segmente auch Vektoren sind (siehe Abbildung 43). Es ist denkbar, dass eine Sprache mit Operatorüberladung wie Perl dies noch eleganter hätte lösen können.

5.4.2 Ein- und Ausgabe von Geodaten

Im Zuge der Entwicklung stellte sich schnell heraus, dass die direkte Verwendung von OpenStreetMap-Daten etwas schwieriger war als erwartet. Die Eingabe wurde daher zunächst über die unter `download.geofabrik.de` angebotenen OpenStreetMap-Shapefiles realisiert. Auch die Ausgabe erfolgt zunächst als Shapefile, damit das Generalisierungsergbnis leicht mit gängiger Software betrachtet werden kann.

Die Shapefiles der Geofabrik enthalten aus technischen Gründen veränderte Attribute anstelle der Original-OpenStreetMap-*tags*. Diesem Problem wurde begegnet, indem ein Adapter (*object adapter* [cf. GHJV95, 139–141]) die aus den Shapefiles gelesenen Attribute wieder zurück in das OpenStreetMap-Format wandelt, ohne dass dies für den *Combiner* sichtbar wird.

Die Nutzung des *frameworks* GeoTools zur Ein- und Ausgabe von Geodaten war unerwartet problematisch. Zwar bietet GeoTools umfangreiche Unterstützung für verschiedene Datenformate, jedoch sind die entsprechenden GeoTools-Klassen umständlich zu benutzen, ihr Interface ist nicht stabil und sie arbeiten sehr langsam. Es ist gelungen – wenn auch mit erheblichem Zeitaufwand –, eine Lösung hierfür zu entwickeln.

Um die Datenausgabe auf eine akzeptable Geschwindigkeit zu beschleunigen, werden diese nun im GeoJSON-Format ausgegeben (implementiert als reine Textausgabe, was sehr billig ist). Das GeoJSON wird anschließend mit GDAL in ein Shapefile konvertiert. Insgesamt ist diese Lösung um ein Vielfaches schneller als die von GeoTools zur Verfügung gestellte Shapefile-Ausgabe.[1]

Außer dem eigentlichen Generalisierungsergebnis kann der *Combiner* auch zahlreiche zusätzliche Geodatensätze ausgeben, die das Verständnis von der Arbeitsweise des Algorithmus bei Anwendung auf *real world*–Daten verbessern und auch bei der Fehlersuche hilfreich sind. Diese Ausgaben sind in keiner Weise optimiert. Sie können mit dem Schalter `--debug` kontrolliert werden.

5.4.3 Flexibilität und Effizienz

Die Definition des Spezialfalls, auf den die Arbeit beschränkt werden soll, in einem eigenen Paket `highway` vom Rest des Combiners zu trennen und sie damit leicht austauschbar zu machen, war grundsätzlich eine gute Idee. Eine solche Trennung konsequent umzusetzen, wäre allerdings über die Aufgabenstellung hinaus gegangen.

Im Unterschied zur formalen Definition in Abschnitt 4.3.2 sind *parallelLinks* und *parallelRechts* nicht als homogene binäre Relation auf Segmenten implementiert, sondern als Relation zwischen Segmenten und *Mengen* von Segmenten. Mit anderen Worten: Es wird nicht jeweils nur ein einziges Segment je Seite als „parallel“ markiert, sondern mehrere. Auf die Korrektheit der Algorithmen hat dies keinen Einfluss. Zwar entstehen beim NodesZuordnen zunächst mehr Zuordnungen. Dies sind jedoch Duplikate, die aufgrund der Mengeneigenschaft entfallen (vergleiche Abschnitt 5.3.4). Es stellt sich hier also die Frage, welche Implementierungsvariante mit dem Java Collections Framework am effizientesten umzusetzen ist. Ein *profiling* wurde noch nicht durchgeführt.

Eine besonders effiziente Implementierung stand nicht im Vordergrund. Die gewählten Datenstrukturen haben einen höheren Speicherbedarf als eigentlich nötig. Beispielsweise werden in den Implementierungen der Algorithmen Fusspunkt (verwendet beim Splitten von Segmenten) und Analyse zahlreiche neue Vektoren-Objekte erzeugt, die jeweils nur für eine einzige geometrische Berechnung benötigt werden. Die dadurch häufiger notwendige automatische Freispeichersammlung *(garbage collection)* zur Beseitigung dieser Objekte ist zeitaufwändig. Diese Zeit- und Speicherkosten hängen von der gewählten Implementierung ab und konnten deshalb in Abschnitt 4.4 nicht berücksichtigt werden. Insbesondere für den Speicheraufwand besteht Optimierungspotenzial.

Ohnehin musste die Software aus Zeitgründen ohne besondere Rücksicht auf die Eigenschaften von Compiler und Hardware entwickelt werden. Speicherzugriff und -verwaltung sind deshalb wahrscheinlich erhebliche Engpässe des *Combiners.*

Ein Beispiel für unnötigen Speicheraufwand ist die Bildung durchgehender Linienzüge im Generalisierungsergebnis: Diese sind in Kapitel 4.3 spezifiziert als Mengen zusammenhängender Segmente und sind auch so in `AbstractLine` implementiert. Tatsächlich

1 In der später erschienenen Version 10.0 von GeoTools ist die Shapefile-Unterstützung neu implementiert. [cf. Deo13] Dem ersten Anschein nach löst dies die beschriebenen Probleme nicht.

müssen diese Segmente jedoch für `AbstractLine` in jedem Fall erst anhand einer Liste von Stützpunkten erzeugt werden. Da für `ResultLine` die Segmente später ohnehin nur benötigt werden, um daraus gerade eine Liste von Stützpunkten zu erzeugen, wäre eine Optimierung von `AbstractLine` durch *lazy initialisation* angebracht: Es würde in `AbstractLine` zunächst die Liste von *nodes* gespeichert und die SEGMENTIERUNG nur dann durchgeführt, wenn tatsächlich Segmente angefordert werden.

6 Ergebnisdiskussion

Bei der Evaluierung der Software ist zu bedenken, dass die OpenStreetMap-Datenbank ständig weiter bearbeitet wird. Konsistente Ergebnisse für ein bestimmtes Gebiet erfordern dagegen, dass immer mit Geodaten vom selben Stand gearbeitet wird.

In diesem Kapitel wird nach und nach anhand von Einzelbeispielen die Funktion der mit dieser Arbeit entwickelten Software (dem *Combiner*) besprochen. Die genutzten Geodaten für das Straßennetz beschränken sich auf einen Testdatensatz aus OpenStreetMap für das Land Nordrhein-Westfalen vom November 2012.

6.1 Anwendung in einfachen Situationen

Bei Anwendung des *Combiners* auf Geodaten aus der OpenStreetMap-Datenbank zeigt sich, dass die implementierten Algorithmen grundsätzlich funktionieren.

In Abbildung 48 sind links OpenStreetMap-Linienzüge in der einfachen Situation einer Autobahn ohne Anschlussstellen nebst innerstädtischen Sammelstraßen zu sehen. Durch Anwendung des *Combiners* ergibt sich das rechts dargestellte automatisiert zusammengefasste Ergebnis. Anstelle der beiden Richtungsfahrbahnen der Autobahn gibt es nun nur noch einen einzigen Linienzug als Straßenachse. Die Nähe zu nachgeordneten Straßen und deren planfreie Kreuzung mit der Autobahn (bei Markierung ①) stört die Generalisierung nicht. Auch eine Strecke paralleler Fahrbahnen im nachgeordneten Netz wurde trotz eines scharfen Knicks erfolgreich als parallel erkannt und zusammengefasst (Markierung ②).

Abbildung 49 zeigt beispielhaft, wie sich dieses Generalisierungsergebnis sinnvoll visualisieren ließe. Der *Combiner* kennzeichnet die zusammengefassten Linienzüge als generalisiert. Zusätzlich gibt der *Combiner* einzelne *tags* der OpenStreetMap-Quelldaten mit aus. Anhand dieser Attribute können leicht Regeln mit jeweils passenden Linearsignaturen definiert werden.

Bei näherer Betrachtung ist die korrekte Arbeitsweise der implementierten Algorithmen in diesem einfachen Fall gut zu erkennen (Abbildung 50): Linienzüge werden durch Splitten derart unterteilt, dass möglichst jeweils zwei Punkte auf Parallelen einander gegenüber liegen (gut zu sehen bei ①). Die dabei entstehenden einander gegenüberliegenden Fragmente sind durch ihre Ähnlichkeit offensichtlich einfach auf Parallelität zu prüfen. Selbst dann, wenn beim Splitten keine optimalen Paare entstehen, gelingt dies, solange wenigstens jeweils ein *Teil* der miteinander verglichenen Segmente zueinander parallel ist (②). Von den so entstehenden Zuordnungen der einander gegenüberliegenden OpenStreetMap-*nodes* lässt sich durch Verbindung ihrer Mittelpunkte leicht eine Mittellinie im Verlauf der Straßenachse als Ergebnis ableiten.

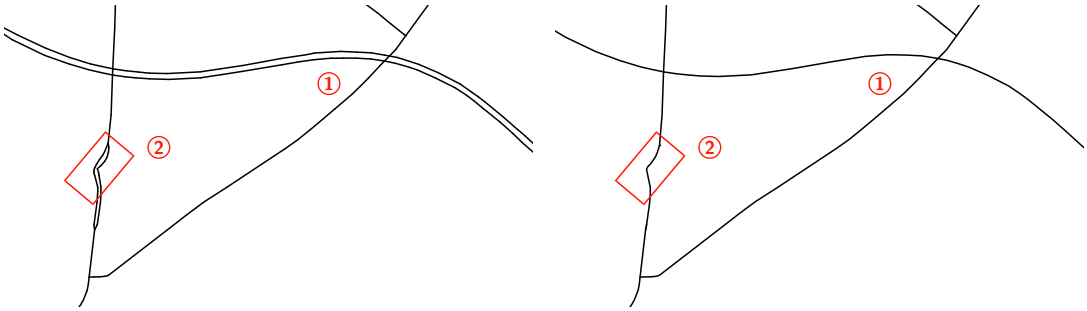

Abbildung 48: Ergebnis des *Combiners* im einfachen Fall (links Eingangsdaten, rechts Generalisierungsergebnis; Köln-Gremberg mit Autobahn)

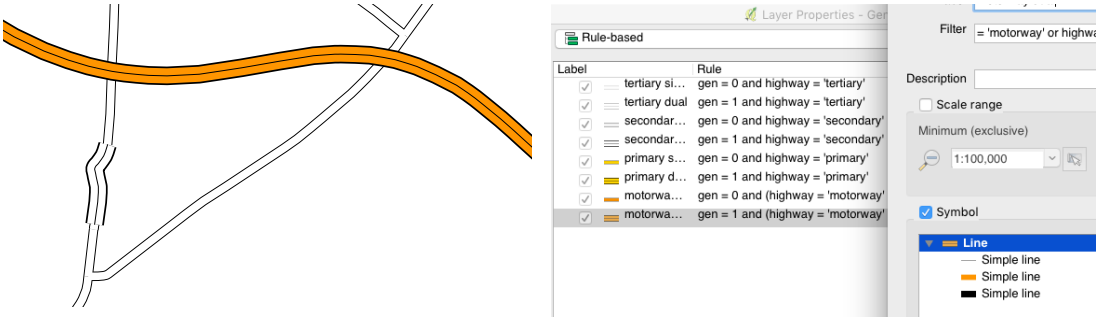

Abbildung 49: Visualisierung des Generalisierungsergebnisses (links Kartendarstellung, rechts Screenshot der Zeichenregeln in QGIS)

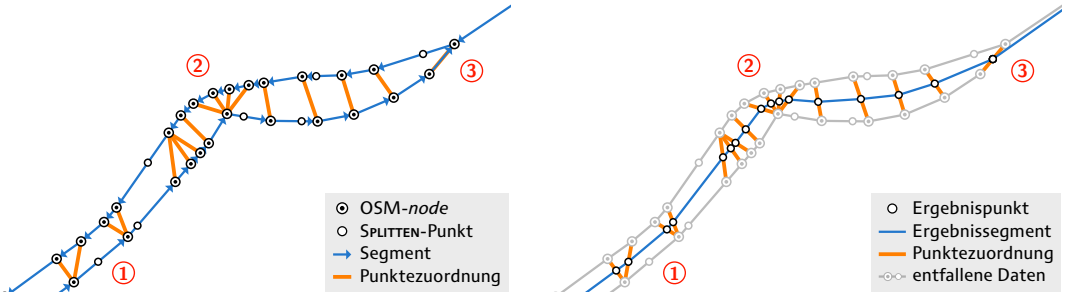

Abbildung 50: Detaildarstellung Generalisierung mit Punktezuordnungen (links Eingangsdaten, rechts Generalisierungsergebnis; ② in Abbildung 48)

Der Übergang zwischen generalisierten und nicht generalisierten Linienzügen (③) wird unten im Abschnitt 6.3.1 diskutiert.

Für den NRW-Testdatensatz wurden die in Abschnitt 4.3.2 genannten Beispielwerte von höchstens um 15° abweichender Ausrichtung und höchstens 40 m Abstand zweier Paralleler Segmente empirisch als grundsätzlich tauglich bestätigt. Für Autobahnen genügt überwiegend bereits ein Limit von nur 10° Abweichung. Sonstige Straßen besitzen wesentlich engere Kurvenradien und haben dabei vereinzelt Segmente, welche als parallel gelten könnten, deren Ausrichtung jedoch um 30° oder mehr voneinander abweicht.

Um Praxistauglichkeit zu erreichen, müsste die Definition von Parallel folglich vom Straßentyp abhängen. Dies wurde aus Zeitgründen nicht mehr als Teil dieser Arbeit umgesetzt.

Der NRW-Testdatensatz enthält als *highway = motorway* attribuierte Linienzüge mit einer Gesamtlänge von 4515 km. Der *Combiner* erkennt 4461 km davon als parallel; das ausgegebene Ergebnis besteht auf 97,6 % Länge aus zusammengefassten Linienzügen. Nachdem alle nordrhein-westfälischen Autobahnen zweibahnig ausgebaut sind, wäre bei naiver Betrachtung eine Erkennungsrate von 100 % zu erwarten gewesen. Die Untersuchung der ausgegebenen Ergebnisdaten hat gezeigt, dass Linienzüge im Wesentlichen in den folgenden Fällen als *nicht* parallel erkannt werden:

- fehlerhafte Klassifizierung von Auf- und Abfahrten (*highway* = ∗)
- fehlerhaftes Attribut für die Straßennummer (*ref* = ∗)
- definierte geometrische Kriterien für Parallelität nicht erfüllt
  (z. B. ungewöhnlich großer Abstand der Richtungsfahrbahnen)

Letzteres ist zu erwarten und offensichtlich korrekt. Die ersten beiden Fälle werden in den folgenden Abschnitten 6.2 und 6.3 näher betrachtet.

6.2 Berücksichtigung von Attributen

Die in Abschnitt 4.3.2 beschriebenen Algorithmen berücksichtigen neben der Klassifizierung der Straße als einziges Attribut die Straßennummer. Im NRW-Testdatensatz erkennt der *Combiner* 54 km der Autobahn-Fahrbahnen nicht als parallel; wird die Straßennummer nicht berücksichtigt, sinkt diese Länge auf 31 km.

An mehreren der dann neu erkannten Stellen trägt nur eine der beiden Richtungsfahrbahnen die Straßennummer als Attribut. In Abschnitt 3.2 wurde dargelegt, dass im Straßennetz grundsätzlich eine vergleichsweise hohe Datenqualität zu erwarten gewesen wäre, weil Fehler darin in der Standard-Karte auf `osm.org` störend sichtbar sind und deswegen zügig von OpenStreetMap-Beitragenden behoben werden. Die nur für *eine* Richtungsfahrbahn fehlende Straßennummer fällt jedoch dem Beitragenden nicht unbedingt auf, solange die Straßennummer der *anderen* Richtungsfahrbahn korrekt angezeigt wird.

Vereinzelt existieren allerdings sogar widersprüchliche Straßennummern der beiden Richtungsfahrbahnen. So ist auf mehreren Abschnitten bei Dülmen die eine Fahrbahn *ref = A 43* gekennzeichnet, die andere jedoch *ref = A 43;B 474*; bei Bliesheim trägt eine der Richtungsfahrbahnen der A 553 das Attribut *ref = A 1*.

Da in diesen Fällen die Geometrie der Fahrbahnen offensichtlich parallel ist, muss hinterfragt werden, welche Bedeutung den Attributen beigemessen werden sollte. Der *Combiner* berücksichtigt derzeit entgegen der ursprünglichen Definition standardmäßig nicht mehr die Straßennummer. Dies kann mit dem Schalter `--tags` kontrolliert werden.

6.3 Verhalten an Straßenkreuzungen

6.3.1 Beseitigung von Topologielücken

Beim Zusammenfassen von Linienzügen erzeugt der *Combiner* neue Stützpunkte entlang einer Mittellinie. Werden die Stützpunkte der ursprünglichen Linienzüge noch für weitere Zwecke verwendet, dürfen diese nicht entfallen, sondern müssen erhalten bleiben.

An *nodes*, die sowohl beim Zusammenfassen entfallene Linien als auch andere Linien benutzen, wird dies zum Problem: Sie werden von nicht generalisierten Linien weiterbenutzt, die generalisierten Linien hingegen verwenden die neu erzeugten Punkte, so dass an den Übergängen topologische Lücken im Straßennetz entstehen. Der *Combiner* versucht dem zu begegnen, indem die *nodes* an den Enden *nicht* generalisierter Linienzüge auf diese Situation hin geprüft und nötigenfalls durch geeignete Punkte auf der generalisierten Mittellinie ersetzt werden.

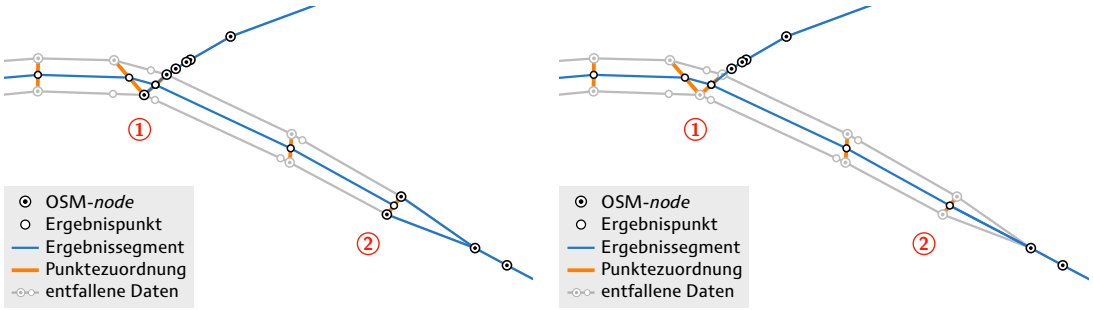

Abbildung 51: Detaildarstellung der Beseitigung von Topologielücken (links vorher, rechts nachher; L 111 in Köln-Deutz)

Abbildung 51 zeigt, dass diese Lösung grundsätzlich funktioniert. Die Straßenkreuzung bei ① wird auf einen einzelnen Punkt zusammengefasst, so dass beide Straßen korrekt miteinander verknüpft sind. Auch der bereits in Abschnitt 4.3.4 angesprochene und zunächst offen gelassene Übergang eines einzelnen Linienzugs in zwei parallele Linienzüge bei ② wird so gelöst.

Dieses zusätzliche, in Abschnitt 4.3 nicht beschriebene Verfahren ist standardmäßig aktiv, kann aber mit dem Schalter `--no-cleanup` kontrolliert werden. Auch in Abbildung 50 kam es zum Einsatz. Darin ist bei ③ gut zu sehen, wie am Beginn des generalisierten Abschnitts die Geometrie geringfügig geändert wird, um die Topologie zu erhalten.

Topologielücken können jedoch nur dann mit diesem Verfahren geschlossen werden, wenn tatsächlich ein *geeigneter* Punkt auf der generalisierten Mittellinie gefunden wird. Wie die folgenden Abschnitte zeigen werden, ist dies oft nicht der Fall.

Mit diesem Verfahren können überdies in der gegenwärtigen Implementierung nur solche Topologielücken beseitigt werden, die erst beim Zusammenfassen von Parallelen entstanden sind. Besonders auffallend ist diese Einschränkung bei Verbindungsrampen an planfreien Kreuzungen. Sie sind keine Richtungsfahrbahnen und werden deshalb – wie in Abschnitt 3.2 festgelegt – in dieser Arbeit nicht betrachtet. Sie sind bereits in den Eingangsdaten für die Zusammenfassung durch Objektauswahl entfallen und damit auch nicht im Generalisierungsergebnis enthalten (Abbildung 52).

Dass einander kreuzende Autobahnen miteinander verknüpft sind, lehrt die Lebenserfahrung; aus der Topologie der generalisierten Geodaten geht es indes nicht hervor.

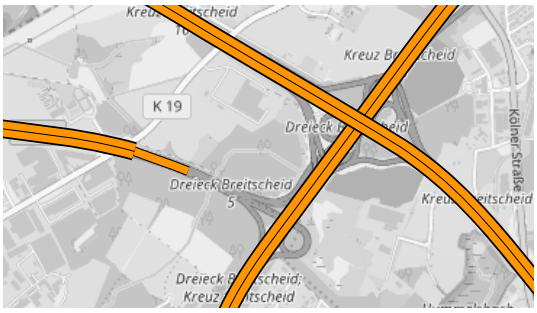

Abbildung 52: Fehlende Verbindungsrampen im Generalisierungsergebnis
[Hintergrund cf. osm18, entsättigt]

6.3.2 Fehlende Kreuzungserkennung

Das in Abschnitt 6.3.1 beschriebene Verfahren funktioniert auch innerstädtisch nicht in jeder Situation, wie Abbildung 53 zeigt. Zwar führt es zu einem gelungenen Ergebnis bei ①. Bei ② und ③ wird hingegen aufgrund der spezifischen Kreuzungsgeometrie kein „geeigneter“ Punkt auf einer der generalisierten Linien gefunden. Die dortigen Lücken in der Topologie bleiben somit bestehen. Ähnliche Probleme existieren an vielen weiteren innerstädtischen Kreuzungen.

Zwar ließe sich argumentieren, dass in vielen dieser Fälle die Attribute der OpenStreetMap-Daten fehlerhaft sind. Beispielsweise sollte in Abbildung 53 die ① und ③ verbindende Abbiegefahrbahn eigentlich als solche klassifiziert werden (*highway=primary_link* statt *primary*), wodurch vermieden würde, dass sie bei ③ als parallel zur Hauptfahrbahn erkannt wird. [cf. osm12b]

Allerdings ist gerade dieser Fehler im NRW-Testdatensatz stark verbreitet. So sind in der Kölner Innenstadt (bis einschließlich der Ringe) von insgesamt 79 erfassten Abbiegefahrbahnen ganze 26 nicht als solche klassifiziert. Es ist auch fraglich, ob hier mit fortschreitender Bearbeitung der OpenStreetMap-Datenbank Verbesserungen zu erwarten sind, denn Abbiegefahrbahnen und Hauptfahrbahnen werden auf `osm.org` mit

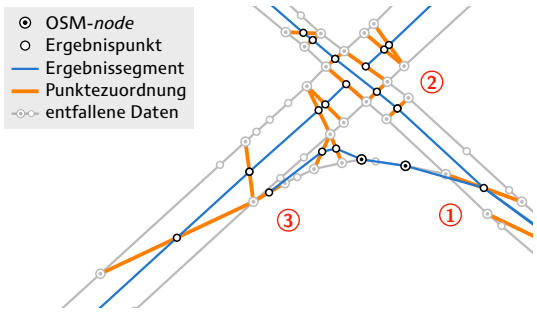

Abbildung 53: Verbleibende Topologielücken (Innere Kanalstraße, Köln)

nahezu identischer Linearsignatur gezeichnet, so dass eine falsche Klassifizierung beim Betrachten nicht auffällt. Vor dem Hintergrund der Aufgabenstellung, die das Ziel *praxistauglicher* Algorithmen vorgibt, ist folglich offensichtlich, dass dieses Problem einer automatisierten Lösung bedarf.

An Kreisverkehren gibt es ebenfalls typische Probleme. Abbildung 54 zeigt, wie alle im Kreis einander gegenüberliegenden Segmente als parallel erkannt werden. Mit dieser Masse widersprüchlicher Punktezuordnungen kann der Algorithmus zur Zusammenfassung keine brauchbaren Ergebnisse mehr liefern. Auch der in Abschnitt 6.3.1 besprochene Ansatz zur Beseitigung von Lücken in der Topologie läuft ins Leere.

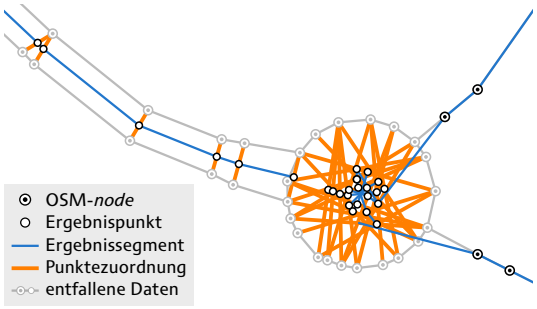

Abbildung 54: Verhalten am Kreisverkehr (Friedrichstraße, Köln-Porz)

Zwar wäre es möglich, die meisten Kreisverkehre über OpenStreetMap-*tags* zu erkennen und von der Generalisierung auszuschließen (was aber aus Zeitgründen nicht mehr als Teil dieser Arbeit umgesetzt wurde). Eine Generalisierung der Kreisverkehre – zum Beispiel durch Qualitätsumschlag der linearen Kreisfahrbahn in einen einzelnen Kreuzungspunkt im Zentrum – könnte jedoch überaus nützlich sein, etwa bei der Herstellung einer Straßenkarte im mittleren Maßstabsbereich.

Auffallend ist, dass alle diskutierten Probleme an Kreuzungen auftreten. Auf freier Strecke funktioniert der *Combiner* weitgehend problemlos. Gäbe es eine zuverlässige Methode, automatisiert Kreuzungen zu erkennen, ließe sich wahrscheinlich leicht Praxistauglichkeit erzielen.

Anhang D zeigt weitere Beispiele von Kreuzungssituationen, mit denen der *Combiner* nicht gut zurechtkommt.

6.4 Effizienz

Die Implementierung der vorgestellten Algorithmen im *Combiner* hat bei Anwendung auf den NRW-Testdatensatz wie in Abschnitt 4.4 erwartet einen Zeitaufwand, der im Verhältnis zur Zahl der Segmente nicht schneller als linear wächst. Tabelle 1 stellt die gemessenen Ausführungszeiten t am Beispiel von Fernstraßen (bis einschließlich *highway=primary*, ohne *links*) in der Umgebung von Köln dar. Angefangen mit einer Gruppe von Stadtteilen bis hin zum Bundesland Nordrhein-Westfalen wurden steigende Flächengrößen berechnet bei ansonsten gleichen Bedingungen.

				Wachstumsfaktor				
Gebiet	Fläche	$\lvert S\rvert$	t	für $\lvert S\rvert$	für t	$\frac{\lvert S'\rvert}{\lvert S\rvert}$	ν	ψ
Stadtteile	161 km^2	3516	0,41 s	–	–	1,56	10	1,31
Großstadt	647 km^2	8776	0,78 s	2,5	1,9	1,48	8	1,25
Reg.-Bez.	7556 km^2	54247	2,2 s	6,2	2,8	1,23	8	1,37
Bundesland	34881 km^2	157014	5,7 s	2,9	2,6	1,27	6	1,27

Tabelle 1: Zeitbedarf für unterschiedlich große Gebiete

Es bestätigt sich, dass die in Abschnitt 4.4 eingeführten Größen ν und ψ nicht von $|S|$ abhängig sind. Der Wachstumsfaktor für t beim Schritt von der Großstadt zum Regierungsbezirk ist mit 2,8 deutlich kleiner als der Faktor für $|S|$ mit 6,2 (also besser als für die anderen Schritte). Wie das Verhältnis $|S'| : |S|$ der Segmentanzahl nach bzw. vor dem Splitten zeigt, liegt dies an dem mit diesem Schritt erstmals inkludierten ländlichen Raum. Auch dies entspricht den Erwartungen.

Wird neben der Segmentanzahl in den Eingangsdaten auch deren Detaillierungsgrad geändert, ergibt sich jedoch ein anderes Bild. Tabelle 2 zeigt das Verhalten des *Combiners* beim Hinzufügen von Abbiege- und Verbindungsfahrbahnen *(links)*. Der Zeitbedarf wächst hier überproportional. Dem deutlich gestiegenen Verhältnis $|S'| : |S|$ zufolge

			Wachstumsfaktor				
links	$\lvert S\rvert$	t	für $\lvert S\rvert$	für t	$\frac{\lvert S'\rvert}{\lvert S\rvert}$	ν	ψ
ohne	54247	2,2 s	–	–	1,23	8	1,37
mit	73408	12,8 s	1,4	5,9	1,85	11	1,42

Tabelle 2: Zeitbedarf ohne und mit Verbindungsfahrbahnen

könnte dies an dem bereits in Abschnitt 5.4.3 angedeuteten hohen Speicheraufwand der gewählten Implementierung liegen.

Auch dies ist ein deutlicher Hinweis auf die oben diskutierte Notwendigkeit einer Kreuzungserkennung.

6.5 Anwendung auf andere Spezialfälle

Der nach Abschnitt 3.2 zu untersuchende Spezialfall von *Richtungs*fahrbahnen impliziert, dass die entwickelten Algorithmen genau zwei entgegengerichtete Fahrbahnen derselben Straße zusammenzufassen sollen. Ebenda wurde jedoch bereits angedeutet, dass sie möglicherweise auch auf andere Spezialfälle anwendbar sein könnten. Abschließend soll deshalb an zwei Beispielen betrachtet werden, wie sich die im *Combiner* implementierten Algorithmen verhalten, wenn die Eingangsdaten nicht entsprechend dem Spezialfall aufgebaut sind, für den die Algorithmen entwickelt wurden.

6.5.1 Fahrbahnen für unterschiedliche Arten von Verkehr

Verteilerfahrbahnen und langgezogene Rampen für abbiegenden Verkehr sind verbreitete, schon in Abschnitt 3.1.3 genannte Beispiele für parallele Fahrbahnen für unterschiedliche Verkehrsarten.

Nachdem der *Combiner* dazu vorgesehen ist, n parallele Linienzüge zu $n-1$ Linienzügen zu generalisieren (mit $n = 2$ im Normalfall), liegt es nahe, die Algorithmen zur Generalisierung einfach $n-1$ Mal nacheinander anzuwenden, so dass im besten Fall auch bei beliebig vielen Parallelen am Ende nur noch ein einziger Linienzug übrig bleibt. Abbildung 55 demonstriert, dass diese Strategie gelingen könnte.

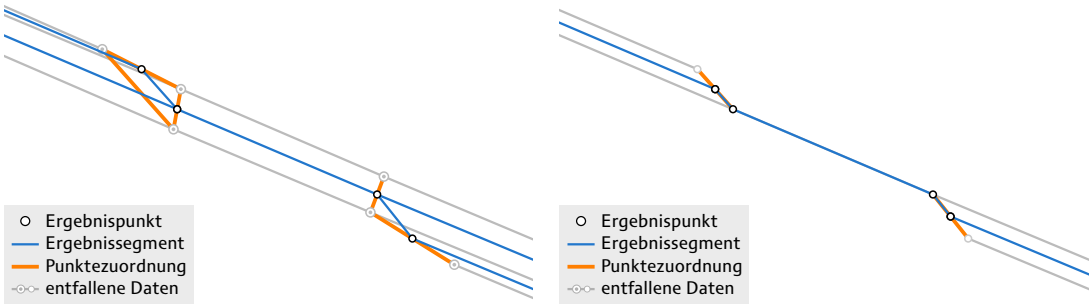

Abbildung 55: Detaildarstellung der wiederholten Ausführung für mehr als zwei Parallelen (links erste, rechts zweite Generalisierung; A 3 bei Leverkusen)

Wie zu sehen ist, wird in diesem Beispiel die zwischen 2 und 3 variierende Zahl von Parallelen in zwei Schritten korrekt auf einen einzelnen Linienzug zusammengefasst. Dieser liegt allerdings nicht mehr mittig im Verlauf der Straßenachse, sondern ist jeweils seitlich in Richtung der dritten Parallele verschoben. Eine praxistaugliche Lösung müsste

– wie in Abschnitt 4.3.4 schon angedeutet – den Graphen aus Punktzuordnungen und Segmenten in seiner gesamten Breite in nur *einem* Schritt beurteilen, um anhand von Attributen die tatsächliche Straßenachse ermitteln zu können. Attribute können vom *Combiner* bei wiederholter Ausführung gegenwärtig nicht sinnvoll berücksichtigt werden.

Nicht unerwähnt bleiben darf, dass sich das Unvermögen des in Abschnitt 6.3.1 vorgestellten Ansatzes zur Beseitigung von Topologielücken, in allen Situationen einen „geeigneten“ Punkt für den Lückenschluss zu finden, bei wiederholter Ausführung besonders stark negativ auf das Ergebnis auswirkt. Wie Abbildung 56 zeigt, kommt es vor, dass das Generalisierungsergebnis nach mehreren Wiederholungen mit zahlreichen Lücken durchsetzt ist. Dieses Bild zeigt sich sehr verbreitet, womit das Ergebnis ungeeignet zur Weiterverarbeitung ist.

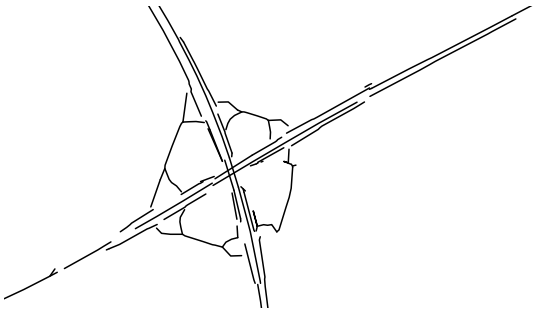

Abbildung 56: Überreste eines Kleeblatts nach zwei Generalisierungen (Autobahnkreuz Leverkusen)

Die Anzahl der wiederholt durchzuführenden Generalisierungen kann im *Combiner* mit dem Schalter `--iterations` kontrolliert werden.

6.5.2 Eisenbahnstrecken

Wie sich zeigt, ist die entwickelte Software grundsätzlich auch auf Eisenbahnstrecken anwendbar. Beschränkt man sich auf Haupt- und Streckengleise (*railway*=*rail*, *usage*=*main*) und berücksichtigt die Streckennummer als Attribut (*ref* = ∗), so werden viele doppelgleisige Streckenabschnitte erkannt und zusammengefasst.

Abbildung 57 zeigt das Ergebnis der Anwendung auf die südliche Einfahrt zum Güterbahnhof Köln-Kalk (①). Während die Reisezug-Gleise ②–③ und auch die drei Zulaufstrecken ④, ⑤ und ⑥ korrekt generalisiert wurden, funktioniert die Zuordnung im Weichenbereich bei ⑦ nicht richtig, weil sich hier Gleise mit unterschiedlichen Streckennummern kreuzen.

Dem für ein Straßennetz entwickelten Verfahren zur Beseitigung von Topologielücken aus Abschnitt 6.3.1 gelingt es auch aufgrund der spitzen Winkel oft nicht, „geeignete“ Punkte zur Beseitigung der Lücken zu finden, wodurch ein gelungenes Ergebnis verhindert wird (Abbildung 58).

Anders als im Straßennetz existieren derartige Probleme im Eisenbahnnetz nicht nur an Kreuzungen, sondern auch über längere Streckenabschnitte hinweg. Betroffen sind insbesondere langgestreckte Überwerfungsbauwerke sowie Strecken mit Richtungsbetrieb.

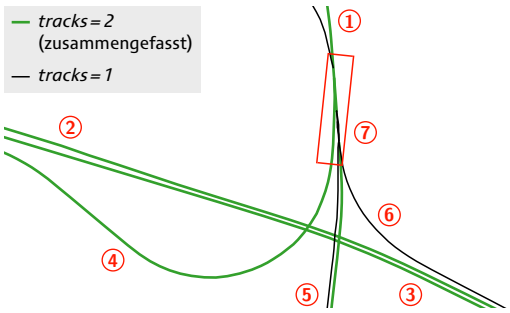

Abbildung 57: Generalisierungsergebnis bei Anwendung des *Combiners* auf Bahnstrecken

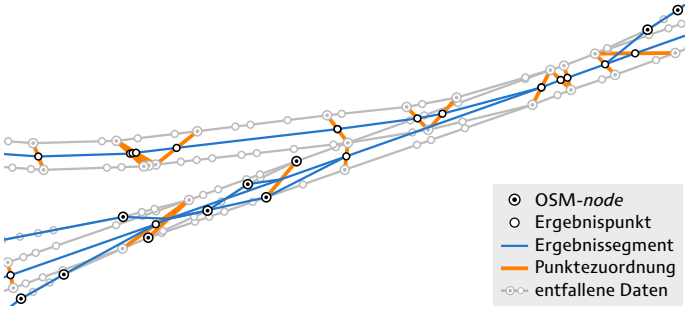

Abbildung 58: Detaildarstellung Generalisierung im Weichenbereich
(⑦ in Abbildung 57, gedehnt)

Bei letzteren liegt zwischen den beiden zusammenzufassenden Gleisen mit derselben Streckennummer jeweils ein Gleis einer Strecke mit einer *anderen* Nummer. [cf. wp16]

Auffallend ist ferner, dass der Zeitbedarf des *Combiners* für Bahnstrecken wesentlich höher ist als im Straßennetz. Für Bahnstrecken in Köln werden mehr als 20 Segmente als „nah“ (im Sinne von NaheSegmente in Abschnitt 4.3.1) beurteilt. Dies liegt daran, dass Eisenbahngleise oft gebündelt verlegt werden, offenbar um angesichts der großen Kurvenradien den Platzbedarf zu minimieren. Folglich sind auch entsprechend mehr Teilungen von Segmenten notwendig; die Annahme $|S'| \approx 2 \cdot |S|$ aus Abschnitt 4.4 ist selbst mit der Beschränkung auf Hauptgleise nicht mehr zutreffend.

Tabelle 3 zeigt, dass sich der Zeitbedarf durch bessere Anpassung der Definition von Parallel aus Abschnitt 4.3.2 auf die Bedingungen im Eisenbahnnetz zumindest geringfügig verbessern lässt.

	Parallel				Wachstumsfaktor				
Spezialfall	α	η	$\lvert S\rvert$	t	für $\lvert S\rvert$	für t	$\frac{\lvert S'\rvert}{\lvert S\rvert}$	ν	ψ
Straßen	15°	40 m	8776	0,78 s	–	–	1,48	8	1,25
Bahngleise	15°	40 m	11607	8,2 s	1,3	11	2,81	22	1,77
Bahngleise	8°	15 m	11607	2,5 s	1,3	3,2	2,14	11	1,66

Tabelle 3: Zeitbedarf für unterschiedliche Spezialfälle und Parallel-Definitionen

7 Schlussfolgerungen und Ausblick

7.1 Praktische Anwendbarkeit

Es konnte durch die Implementierung in Software gezeigt werden, dass die Erkennung paralleler Linienzüge auf der Basis eines geometrischen Vergleichs möglichst kurzer Fragmente prinzipiell funktioniert. Mit hierfür entwickelten Algorithmen können auch viele schwierige Situationen des in Abschnitt 3.1 ausgewählten Spezialfalls der „baulich getrennten Richtungsfahrbahnen im Straßenraum“ zufriedenstellend gelöst werden.

Im Vergleich zu den in Abschnitt 2.5 diskutierten Ansätzen sind keine offensichtlichen grundsätzlichen Nachteile der mit der vorliegenden Arbeit entwickelten Methode zu erkennen. Bei der Anwendung auf komplexe Straßennetze zeigt sich jedoch, dass erhebliche Probleme im Kreuzungsbereich, wie sie bereits von anderen Ansätzen ähnlich berichtet wurden, auch bei den in dieser Arbeit vorgestellten Algorithmen auftreten (siehe Abschnitt 6.3). Aus Zeitgründen konnten praxistaugliche Lösungen hierfür nicht mehr als Teil dieser Arbeit entwickelt werden.

Die praktische Anwendbarkeit der entwickelten Algorithmen ist deshalb beschränkt. Abschnitt 7.2 diskutiert Wege, um das mögliche Einsatzgebiet zu erweitern.

Ferner darf hinterfragt werden, ob die Vorgabe der Aufgabenstellung hinsichtlich der Praxistauglichkeit der zu entwickelnden Algorithmen in Anbetracht der zeitlichen Begrenzung angemessen war. Zwar ist Praxistauglichkeit immer anzustreben; wie schnell sie sich tatsächlich erreichen lässt, kann jedoch von Umständen abhängen, die nicht vollumfänglich vorherzusehen sind. Auch ist zu bedenken, dass Ladak und Martinez den Umfang ihrer Arbeit, welcher eine vergleichbare Problemstellung zugrunde liegt, auf etwa sieben Personenmonate schätzen (siehe Abschnitt 2.5.2). [cf. LM96] Angesichts der im Rahmen der vorliegenden Arbeit aufgetretenen technischen Probleme ist nicht zu erwarten, dass der Aufwand für die Entwicklung einer praxistauglichen Lösung wesentlich geringer wäre.

7.2 Möglichkeiten zur Weiterentwicklung

7.2.1 Berücksichtigung unterschiedlicher Straßentypen

Wie schon in Abschnitt 6.1 erläutert, unterscheiden sich die optimalen Werte für die in Abschnitt 4.3.2 zur Definition von Parallel benutzten baulichen Kriterien für Abstand und Winkel der Richtungsfahrbahnen je nach Straßentyp. Es wäre deshalb sinnvoll, diese Kriterien je nach Klassifikation der Straßen in der OpenStreetMap-Datenbank entsprechend zu variieren. Die Fehlerrate in der Erkennung würde dadurch sinken.

Diese geringfügige Verbesserung konnte lediglich aus Zeitgründen nicht mehr als Teil dieser Arbeit umgesetzt werden.

7.2.2 Statistische Auswertung

Attribute werden bei der ANALYSE nach Abschnitt 4.3.2 nur als hartes Ausschlusskriterium verwendet. Abschnitt 6.2 erläutert, dass diese Methode nicht zufriedenstellend funktioniert.

Als Alternative bietet es sich an, aus der Geometrie und den Attributen eine *Wahrscheinlichkeit* abzuleiten, mit der die verglichenen Segmente insgesamt parallel sind, und als einziges *hartes* Kriterium einen Schwellenwert für diese Wahrscheinlichkeit vorzusehen. Es könnten dann auch leicht weitere Attribute wie Straßenname oder die vertikale Ebene in angemessenem Umfang berücksichtigt werden, ohne dass einzelne Aspekte, die falsch attribuiert sind, ein korrektes Ergebnis verhindern.

Diese Idee erscheint vielversprechend, konnte jedoch aus Zeitgründen nicht mehr als Teil dieser Arbeit umgesetzt werden.

7.2.3 Kreuzungserkennung

Das in Abschnitt 6.3.1 beschriebene Verfahren zur Schließung von Topologielücken arbeitet in vielen Fällen nicht zufriedenstellend. Die implementierte Strategie ist es, immer die nahesten Stützpunkte des generalisierten Linienzugs (also die Mitte der kürzesten verknüpften Punktezuordnung) als „geeignet“ zu betrachten. Wie sich zeigte, sind dies jedoch in vielen Fällen eben gerade nicht die „geeigneten“ Punkte. Eine verbesserte Suche nach einem „geeigneten“ Punkt zur Verschiebung der Endpunkte wäre vermutlich relativ einfach möglich.

Nötig wäre eigentlich eine Analyse des Graphen des Straßennetzes, um anhand der *vorher* existierenden Verknüpfungen im Netz *hinterher* die bei der Generalisierung entstandenen Lücken wirklich sinnvoll schließen zu können.

Dass Kreuzungen nicht automatisiert als solche erkannt werden, hat sich als das größte Problem des entwickelten Verfahrens herausgestellt (Abschnitt 6.3.2). Diesen Mangel teilt der vom Verfasser gewählte Ansatz mit vielen der in Abschnitt 2.5 besprochenen älteren Ansätze.

Die Implementierung einer praxistauglichen Kreuzungserkennung wäre demnach die Voraussetzung für eine gelungene automatisierte Zusammenfassung paralleler Linienzüge. Sie scheint die vielversprechendste Möglichkeit zur Weiterentwicklung zu sein.

Um Kreuzungen zu erkennen, sind unterschiedliche Ansätze denkbar. Beispielsweise lassen sich Kreuzungen wahrscheinlich durch eine graphentheoretische Analyse des Straßennetzes auf kleinste Maschen mit anschließender geometrischer Auswertung erkennen: Unterschreitet die Fläche der Masche eine bestimmte Mindestgröße, so handelt es sich bei deren Knoten möglicherweise um Bestandteile einer einzigen Straßenkreuzung, die auf einen einzelnen Punkt zusammengefasst werden können.

Einfacher wäre möglicherweise, sich zunutze zu machen, dass der *Combiner* die Segmente im Generalisierungsergebnis immer zu möglichst langen Linienzügen verkettet. Es liegt nahe, dass dort, wo sehr kurze Linienzüge übrig bleiben, wahrscheinlich eine Kreuzung liegt. Abbildung 59 zeigt beispielhaft einen Puffer mit 125 m Distanz von allen Ergebnis-Linienzügen, deren Länge höchstens 80 m beträgt. Wie zu sehen ist, de-

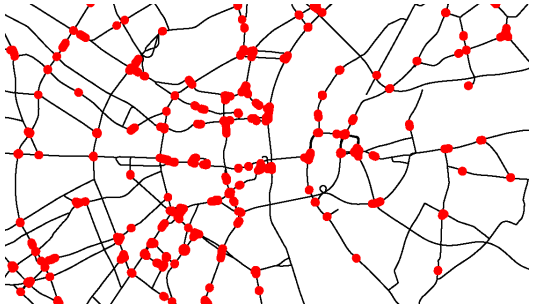

Abbildung 59: Pufferung kurzer Linienzüge (rot) im Generalisierungsergebnis (Köln 1 : 115 000)

cken die Puffer tatsächlich viele komplexe Kreuzungen ab. Eventuell würde es nach der Vereinigung der Pufferflächen bereits genügen, anstelle einer Skelettierung (siehe Abschnitt 2.5.2) alle Linienzüge, welche den Puffer schneiden, in dessen Schwerpunkt zusammenzuführen. Zu prüfen wäre jedoch, ob eine verbesserte Auswertung von Attributen (siehe Abschnitt 7.2.2) diesen Ansatz erschwert, indem die Linienzüge häufiger bei Attributwechseln aufgeteilt werden und damit kürzer werden.

Schließlich sei daran erinnert, dass sich Mackaness und Mackechnie bereits mit dem Thema der Kreuzungserkennung im Kontext der automatisierten Generalisierung von Straßennetzen auseinandergesetzt haben (siehe Abschnitt 2.5.5). [cf. MM99]

7.2.4 Allgemeine Anwendbarkeit

Um die Anwendung auf andere Spezialfälle als die in Abschnitt 3.2 ausgewählten Richtungsfahrbahnen zu ermöglichen, wäre eine Verallgemeinerung der entwickelten Algorithmen und ihrer Implementierung im *Combiner* nötig.

Abschnitt 6.5 beschreibt zwei konkrete Beispiele für Spezialfälle, auf welche die Ergebnisse dieser Arbeit wahrscheinlich prinzipiell übertragbar sind. Um Praxistauglichkeit für diese Fälle zu erzielen, müsste der *Combiner* es erlauben, die auf den Spezialfall der Richtungsfahrbahnen zugeschnittenen Module zur Laufzeit durch entsprechende andere Module zu ersetzen. Diese Möglichkeit wurde bei der Implementierung ausdrücklich vorgesehen (Abschnitt 5.3.3), ist jedoch noch nicht voll nutzbar, da dies im Rahmen dieser Arbeit nicht erforderlich war (Abschnitt 5.4.3). Außerdem wäre eine Lösung zur Kreuzungserkennung erforderlich (Abschnitt 7.2.3), möglicherweise wiederum auf den jeweiligen Spezialfall zugeschnitten.

Es ist jedoch nicht anzunehmen, dass die Anwendung der entwickelten Algorithmen in allen in Abschnitt 3.1 aufgeführten Spezialfällen erfolgreich wäre. Die Definition für Parallel in Abschnitt 4.3.2 bringt mit sich, dass gegenüberliegende Segmente nur genau dann analysiert werden, wenn sie sich wenigstens teilweise überlappen. Bei anthropogenen Parallelen wie gerade den für diese Arbeit zu betrachtenden Richtungsfahrbahnen ist dies normalerweise der Fall. Segmente von Linienzügen, die natürlich Gewachsenes darstellen, können hingegen – bei präziser Erfassung, wie sie von OpenStreetMap im Grundsatz angestrebt wird – derart windschief zueinander liegen, dass eine Überlappung

gegenüberliegender Segmente nicht mehr durchgehend gegeben ist. Eine Übertragung des in dieser Arbeit entwickelten Ansatzes auf solche Spezialfälle wie die in Abschnitt 3.1.6 besprochenen Vegetationsgrenzen ist deshalb wahrscheinlich unmöglich.

7.2.5 Softwarequalität

In der Aufgabenstellung war nicht die Entwicklung einer *Softwarelösung,* sondern die Entwicklung von *Algorithmen* gefordert. Die Implementierung in Software musste erfolgen, um die Praxistauglichkeit der Algorithmen beurteilen zu können (vgl. 4.4), jedoch stand dabei die Qualität der Software selbst nicht im Vordergrund.

In den Abschnitten 5.2 und 5.4 waren bereits Entscheidungen diskutiert worden, die zugunsten einer schnelleren Implementierung qualitative Nachteile mit sich brachten. Eine Weiterentwicklung der folgenden Aspekte kann die Verwendbarkeit des *Combiners* durch den Endnutzer verbessern und wäre deshalb wünschenswert.

- Die Umwandlung von OpenStreetMap-Daten in Shapefiles (vgl. Abschnitt 5.2) ist zum Teil umständlich. Es wäre wünschenswert, dass der *Combiner* die für OpenStreetMap üblichen Formate XML oder PBF direkt verwenden kann. Für kleinere Bereiche wäre auch die Möglichkeit einer direkten Abfrage der OpenStreetMap-Datenbank denkbar. [cf. RT09, 233]

- Die Datenausgabe ist ebenfalls verbesserungsfähig. Sie sollte in einem Format erfolgen, das die Weiterverwendung innerhalb von OpenStreetMap einfacher macht als die derzeitige Ausgabe als Shapefile. Das Verketten aller Segmente (vgl. Abschnitt 5.3.5) sollte womöglich eine optionale Operation sein, da sie nicht für jede denkbare Weiterverarbeitung benötigt wird. Durch das Verketten ist es zudem schwierig, *tags* aus den Eingangsdaten beizubehalten, da sich *tags* im Verlauf einer Straße oft ändern. Deshalb sind bisher nur wenige Attribute überhaupt im Ergebnis enthalten.

- Entsprechend Abschnitt 5.2 wird derzeit als internes Koordinatensystem nur die UTM-Zone 32 verwendet, welche gut zu dem in Kapitel 6 verwendeten Testdatensatz passt. Um Geodaten aus anderen Regionen der Welt verarbeiten zu können, muss der *Combiner* für eine andere UTM-Zone neu kompiliert werden. Die jeweils am besten passende UTM-Zone ließe sich jedoch auch automatisch anhand der Eingangsdaten ermitteln.

- In Abschnitt 5.4.3 wurde bereits erwähnt, dass die Effizienz des *Combiners* Raum für Verbesserungen bietet, insbesondere im Hinblick auf den Speicheraufwand. Die Laufzeitmessungen in Abschnitt 6.5.2 legen nahe, dass dieser unnötig hoch ist. Obwohl bereits Änderungen an Parametern dies verbessern können, erscheint eine generelle Verbesserung des Designs sinnvoll. Beispielsweise könnte die auf Basis von Objekten implementierte Vektor-Mathematik relativ leicht durch statische Methoden ersetzt werden, die ohne zusätzlichen Speicher auskommen.

  Auch der Speicheraufwand im Java Collections Framework hat Optimierungspotenzial. Gegenwärtig werden vom *Combiner* zur Implementierung vieler Mengen

Datenstrukturen vom Typ `LinkedList` genutzt. Diese bedürfen intern für jedes Element eines zusätzlichen Containerobjekts (`LinkedList.Entry`). Möglicherweise ließe sich der Speicheraufwand durch einen Wechsel auf `ArrayList` verringern, geschickt gewählte Anfangskapazitäten vorausgesetzt.

Teile der Algorithmen eignen sich ferner für nebenläufige Ausführung. Mit entsprechenden Anpassungen könnte die Leistungsfähigkeit moderner Hardware besser ausgenutzt werden.

- Die Bedienung über die Kommandozeile (CLI) ist zwar ausreichend für das Testen der Algorithmen durch den Entwickler, ist jedoch für den Endnutzer unnötig anspruchsvoll. [cf. CR03, 363] Eine graphische Benutzeroberfläche (GUI) würde die Einstiegshürde erheblich senken. Mit einer Integration in JOSM als *plug-in* (Abschnitt 5.2) würden sich gleichzeitig die zuvor beschriebenen Probleme beim Einlesen der Ausgangsdaten lösen lassen, da JOSM alle angesprochenen Formate bereits unterstützt.

Des Weiteren ist gegenwärtig die Wiederverwendbarkeit des *Combiners* und seiner Bestandteile für Entwickler nicht optimal. Verbesserungen an den folgenden Punkten können sich indirekt auch auf den Endnutzer auswirken, indem sie die Entwicklung von Software mit weniger Fehlern fördern und die Weiterentwicklung vereinfachen.

- Es besteht gegenwärtig keine nennenswerte Testabdeckung *(test coverage).*
- Vielerorts wird im *Combiner* das objektorientierte Prinzip der Kapselung nicht beachtet, um in Debugging-Ausgaben leichter lesbaren Code zu erzielen. Während dies in bestimmten Umständen sinnvoll sein kann, ist es für öffentliche Klassen zu vermeiden. [cf. Blo01, 98]
- Wie in Abschnitt 5.3.1 erwähnt, könnte ein Implementieren der Programmierschnittstelle (*application programming interface,* API) für JTS und GeoTools die Verwendung von Bestandteilen des *Combiners* in anderen Projekten erleichtern.
- Die Dokumentation der API des *Combiners* ist noch unvollständig.

7.2.6 OSM Inspector

Der *OSM Inspector* ist ein von der Geofabrik entwickeltes Web-Werkzeug, das Beitragenden helfen kann, bestimmte Fehler in der OpenStreetMap-Datenbank zu entdecken. [cf. RT09, 142] Auch einige Attribute von Linienzügen im OpenStreetMap-Straßennetz können damit überprüft werden: „Highways [...] are one of the most important features in OSM. This [OSM Inspector view] helps finding problems on highways such as missing names, unusal highway types or wrong oneway tags.“ [osm10]

Bisher kann der *OSM Inspector* für Straßen immer nur einen einzelnen *way* prüfen. Zusammenhänge zwischen mehreren *ways*, die Teil derselben Straße sind, können nicht erkannt werden. Der in Abschnitt 6.2 beschriebene Fehler widersprüchlicher Attribute zweier Richtungsfahrbahnen bleibt deshalb im *OSM Inspector* unentdeckt.

Es ist denkbar, dass die Integration einer geometrischen Prüfung auf Parallelität – wie sie die in dieser Arbeit entwickelten Algorithmen bieten – in den *OSM Inspector* sinnvoll sein könnte. Es besteht die Chance, dass viele der angesprochenen Fälle fehlerhafter *tags* so durch die OpenStreetMap-Community beseitigt werden können. Dies würde die Datenqualität in OpenStreetMap insgesamt erhöhen und damit die automatisierte Weiterverarbeitung nicht nur mit dem *Combiner* vereinfachen.

Allerdings ist der *Combiner* gegenwärtig nicht auf die Integration in den *OSM Inspector* ausgelegt. Es ist auch fraglich, ob die mit einer Integration verbundenen betriebswirtschaftlichen Kosten den potenziellen Nutzen rechtfertigen.

7.3 Andere Ansätze und neuere Forschung

Neben den in Abschnitt 2.5 besprochenen Ansätzen und der mit dieser Arbeit entwickelten Methode des geometrischen Vergleichs möglichst kurzer Fragmente kommen für das Erkennen und Zusammenfassen paralleler Linienzüge noch weitere Ansätze in Frage.

So wäre denkbar, durch Graphenanalyse Maschen im Straßennetz zu ermitteln und zu prüfen, ob die von ihnen umschlossene Fläche eine solch schmale und lange Form hat, dass es sich um die Fläche zwischen zwei parallelen Linienzügen handeln könnte.

Auch kann der in Abschnitt 2.5.3 erwähnte Ansatz von van Kreveld und Peschier nicht mehr grundsätzlich ausgeschlossen werden. Es ist nicht offensichtlich, dass die mit ihm verbundenen Kosten höher wären als die Kosten der mit der vorliegenden Arbeit entwickelten Methode (siehe Abschnitt 6.4).

Ein neuerer Ansatz wurde von Czioska et al. für Eisenbahnstrecken entwickelt. Er ähnelt vom Ablauf her der in Abschnitt 2.5.2 beschriebenen Thiessen-Polygon-Methode, jedoch werden die Knoten weniger stark verdichtet und es wird direkt mit dem Ergebnis der Delaunay-Triangulierung weitergearbeitet, anstatt Polygone daraus zu erzeugen. Der Vergleich zwischen dem entstandenen Dreiecksnetz und den Eingangsdaten ermöglicht es in den meisten Fällen, Gleiskreuzungen und Weichenverbindungen korrekt zu erkennen und zusammenzufassen. Auf die Möglichkeit der Übertragung dieses Ansatzes auf ein Straßennetz gehen Czioska et al. nicht ein. [cf. CTGV14]

Matheisen setzte für den gleichen Problemfall der Bahnachsenermittlung eine neue Funktion[1] in der Geodatenbank PostGIS ein, die anhand eines *straight skeleton* ein Polygon auf dessen Mittellinien reduziert (siehe Abschnitt 2.5.2). Diese Funktion wendet Matheisen nur auf Haupt- und Streckengleise an (vgl. 6.5.2). Auch sie führt zu Problemen unter anderem im Kreuzungsbereich. Angesichts der von Matheisen hierfür gewählten Lösungen erscheint dem Verfasser eine direkte Übertragbarkeit dieses Ansatzes auf den Straßenraum wenig wahrscheinlich. [cf. Mat17]

Nachdem auch diese neueren Ansätze Probleme mit Kreuzungen beschreiben und nur für konkrete Spezialfälle vorgesehen sind, scheint in Anbetracht der Ergebnisse der vorliegenden Arbeit eine automatisierte Kreuzungserkennung ein sinnvoller nächster Schritt zur Lösung der automatisierten Zusammenfassung von Linienzügen zu sein (siehe Ab-

1 Die Funktion `ST_ApproximateMedialAxis` wurde in PostGIS-Version 2.2.0 hinzugefügt (in 2015).

schnitt 7.2.3). Ob sich der von Czioska et al. für Eisenbahnen hierzu beschriebene Ansatz verallgemeinern ließe, wäre zu prüfen.

Eine offensichtliche Lösung mit allgemeiner Anwendbarkeit für das Problem der Zusammenfassung zeichnet sich derzeit nicht ab. Mit einer zuverlässigen Kreuzungserkennung wird die eigentliche Zusammenfassung jedoch wahrscheinlich zu einem leicht lösbaren Problem, für das aus verschiedenen Ansätzen derjenige ausgewählt werden kann, der am besten auf den jeweils vorliegenden Problemfall passt. Im Wesentlichen ginge es dann um Optimierung der Kosten des Verfahrens und um geschickte Ausnutzung der vorhandenen Attribute. [cf. Tho05, 14]

8 Zusammenfassung

Das Projekt OpenStreetMap (OSM) hat das Ziel der Erstellung einer freien Geodatenbank auf Basis von *volunteered geographic information* (VGI). Die weitere Verarbeitung und Visualisierung von OpenStreetMap-Daten läuft in aller Regel voll automatisiert ab. Sie wird erschwert durch den teilweise sehr hohen Detailreichtum, die daraus folgende Fragmentierung von Linienzügen sowie unvollständige Verknüpfungen zusammenhängender Geodaten wie etwa parallelen Richtungsfahrbahnen über Relationen im Datenmodell.

Eine kartographische Generalisierung von OpenStreetMap-Daten findet bisher nur in geringstem Umfang statt. Dies fällt unter anderem bei parallelen Richtungsfahrbahnen auf, deren Straßenachse in OpenStreetMap nicht erfasst ist und bisher auch nicht in zufriedenstellender Weise automatisiert abgeleitet werden kann. Ansätze zur automatisierten Zusammenfassung von Linienzügen existieren, sind jedoch auf OpenStreetMap-Daten nicht gut anwendbar. Insbesondere können sie Kreuzungssituationen oft nicht ohne besondere Attribute lösen.

Diese Arbeit stellt eine Methode zur Erkennung paralleler Linienzüge auf der Basis eines geometrischen Vergleichs kurzer Fragmente vor. Linienzüge aus OpenStreetMap werden so lange unterteilt, bis sich Stützpunkte auf Parallelen derart einander gegenüberliegen, dass eine Prüfung auf Parallelität leicht möglich ist. Die anschließende Zusammenfassung der erkannten Parallelen ist dann einfach zu lösen. Der Rechenaufwand der entwickelten Algorithmen wächst linear mit der Anzahl der Stützpunkte ($\mathcal{O}(n)$).

Zum Nachweis ihrer Funktionsfähigkeit und zum Test mit *real world*–Daten aus OpenStreetMap erfolgte ihre ausführbare Implementierung. Aufgrund einiger technischer Schwierigkeiten war dies aufwändiger als erwartet. Die mit Java entwickelte Software („Combiner“) hat erhebliches Optimierungspotenzial.

Wie sich zeigt, führt die mit dieser Arbeit entwickelte Methode in vielen Fällen zu einem guten Generalisierungsergebnis. Jedoch leidet auch diese Methode an erheblichen Problemen in Kreuzungssituationen. Aus Zeitgründen war es nicht möglich, eine praxistaugliche Lösung für diese Probleme zu finden.

Auch für andere, parallel entwickelte Methoden neueren Datums wird von ähnlichen Problemen in Kreuzungssituationen berichtet. Eine offensichtliche Lösung mit allgemeiner Anwendbarkeit für das Problem der Zusammenfassung paralleler Linienzüge zeichnet sich derzeit nicht ab. Es ist jedoch anzunehmen, dass eine zuverlässige automatisierte Kreuzungserkennung die Zusammenfassung zu einem leicht lösbaren Problem machen würde. Diese Arbeit benennt dazu mehrere unterschiedliche mögliche Ansätze.

Summary

The goal of the OpenStreetMap project (OSM) is the creation of a free spatial database using volunteered geographic information (VGI). Further processing and visualisation of OpenStreetMap data almost always takes place fully automated. This is impeded by the in parts very high amount of detail, the resulting fragmentation of line strings as well as incomplete linkage of spatial data belonging together by means of relations in the data model.

So far, cartographic generalisation of OpenStreetMap data only happens to the most limited extent. Among other situations this is noticeable at dual carriageways, whose centreline is not included in OpenStreetMap and cannot yet be automatically derived in a satisfactory manner. Approaches for the automatic generalisation of line strings exist, but are not well suited for OpenStreetMap data. In particular they are often unable to find solutions for junctions without special attributes.

This thesis presents a method for the detection of parallel line strings on the basis of a geometric comparison. Line strings from OpenStreetMap are fragmented further until vertices on parallels exist opposite to each other such that verifying the parallelism is simple. The subsequent merging of the detected parallels is then easy to solve. The computational complexity of the developed algorithms grows linear with the number of vertices ($\mathcal{O}(n)$).

An executable implementation of the algorithms demonstrates their operability and allows for testing with real world data from OpenStreetMap. Due to some technical difficulties development was more time-consuming than expected. The software ("Combiner") has considerable potential for optimisation.

The approach developed in this thesis leads to a good generalisation result in many cases. However, this approach has significant problems at junctions as well. Developing a viable solution for these problems was not possible due to time constraints.

There are also reports of similar problems at junctions for other approaches that were developed in parallel to this thesis. An obvious solution with general applicability for the problem of merging parallel line strings is not currently apparent. It can however be expected that a reliable automated junction detection would turn the merging into a simple problem. This thesis lists several distinct potential approaches to this end.

A Mathematische Konventionen

Symbol	Beschreibung
Bezeichner	
$\alpha, \beta, \gamma, ...$	Griechische Kleinbuchstaben bezeichnen reelle Größen mit oder ohne Dimension wie Winkel, Längen oder Verhältniszahlen.
$a, b, c, ...$	Lateinische Kleinbuchstaben bezeichnen in aller Regel Vektoren und Segmente. Lediglich solche reelle Größen, für die ein lateinischer Buchstabe im jeweiligen Kontext fest etabliert ist, werden mit lateinischen Buchstaben bezeichnet.
$A, B, C, ...$	Lateinische Großbuchstaben bezeichnen Mengen.
$\mathcal{A}, \mathcal{B}, \mathcal{C}, ...$	Kalligraphische Buchstaben bezeichnen Mengen von Mengen.
Arithmetik und Trigonometrie	
$\|\alpha\|$	absoluter Betrag der Zahl α
$\|a\|$	geometrische Länge des Vektors a
$\sphericalangle(a, b)$	gerichteter rechtsdrehender Winkel von a zu b im Intervall $(-\pi, \pi)$
Logische Aussagen	
$\neg$	nicht $\neg \text{ja} = \text{nein}$
$\wedge$	und
$\forall M$	für alle Elemente der Menge M Allquantor
$\exists\, a$	es existiert ein Wert für a Existenzquantor
$\nexists\, a$	es existiert *kein* Wert für a
:	so dass gilt
Mengen	
Mengen sind ungeordnet und können kein Element mehrfach enthalten.	
$\{x \mid y\}$	Menge aller x, für die Bedingung y gilt
$\{a, b\}$	Menge bestehend aus den Elementen a und b
$\{\}$	leere Menge
$\|M\|$	Anzahl der Elemente der Menge M
$\cup$	Vereinigung $\{a, b\} \cup \{b, c\} = \{a, b, c\}$
$\cap$	Verschneidung $\{a, b\} \cap \{b, c\} = \{b\}$
$\setminus$	Restmenge (gelesen „ohne“) $\{a, b\} \setminus \{b, c\} = \{a\}$
$\in$	Element in Menge $a \in \{a, b\}$
$\subset$	Teilmenge $\{a\} \subset \{a, b\}$

(fortgesetzt)

Symbol	Beschreibung
Definitionen	
$a \coloneqq b$	es sei a gleich b
$\rightarrow$	Definition einer Relation (Assoziation) der genannten Datentypen
$\equiv$	Definition eines Algorithmus
Algorithmen	
NAME	Kurzbezeichnung eines Algorithmus durch dessen Namen
$\triangleright$	Kommentar
$\hookrightarrow$	Fortsetzung der vorgehenden Zeile
$\leftarrow$	Wertzuweisung
$\varnothing$	fehlender Wert (Nullwert)
für alle	wiederhole die folgenden eingerückten Schritte nacheinander einmal für jedes Element der angegebenen Menge
bis	wiederhole die eingerückten Schritte nacheinander solange, bis die angegebene Bedingung wahr ist
falls	führe die folgenden eingerückten Schritte nur aus genau dann, wenn die angegebene Bedingung wahr ist
Ergebnis	Ergebnis des Algorithmus (Rückgabewert)
Landau-Notation	
$\mathcal{O}(f)$	Menge aller Algorithmen, deren Rechenaufwand nicht wesentlich schneller als f wächst
$\mathrm{o}(f)$	Menge aller Algorithmen, deren Rechenaufwand wesentlich langsamer als f wächst
Einheiten und Datentypen	
px	Pixel die kleinste rechteckige Fläche in gerasterter Darstellung, z. B. auf dem Bildschirm (für OpenStreetMap in der Regel quadratisch); als Längeneinheit: die Länge einer Pixelreihe von 1 px Breite
boolean	ja oder nein (bezogen auf die Wahrheit logischer Aussagen)

Tabelle 4: Mathematische Abkürzungen und Symbole

B Bezeichner im Quelltext

In	Bezeichner	Klasse / Interface	Methode
4.3.1	Ende	`Segment`	`end`
4.3.1	Fusspunkt	`AbstractSegment`	`findPerpendicularFoot`
4.3.1	Hülle	`SourceSegment`	`envelope`
4.3.1	NaheSegmente	`Combiner`	`regionaliseSegments`
4.3.1	Nodes	`NodePair`	`other`
4.3.1	Segmente	`highway.Highway`	`segmentation`
4.3.1	Splitten	`Combiner`	`splitSegments`
4.3.1	Start	`Segment`	`start`
4.3.2	Analyse	`AbstractSegment`	`analyse`
4.3.2	Distanz	`highway.HighwayAnalyser`	`distance`
4.3.2	Mittelpunkt	`NodePair`	`midPoint`
4.3.2	Parallel	`highway.HighwayAnalyser` `SourceSegment`	`shouldEvaluate` `closeParallels`
4.3.3	NodesZuordnen	`NodeGraph`	`createGraph`
4.3.4	Fortsetzung	`GeneralisedSection`	`traverseGraph`
4.3.4	Zusammenfassen	`GeneralisedLines`	`traverse`
4.4	Generalisierung	`Combiner`	`run`

Tabelle 5: Aufschlüsselung der Bezeichner in dieser Arbeit zu denen im Quelltext

Der Quelltext der entwickelten Software ist zusammen mit der zugehörigen API-Dokumentation zugänglich unter:
`http://arne.johannessen.de/thesis`

C Datenmodell und Klassenstruktur

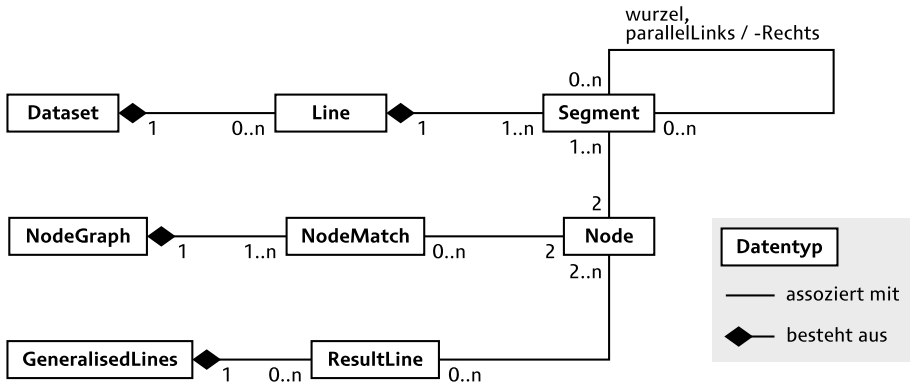

Abbildung 60: Datenmodell

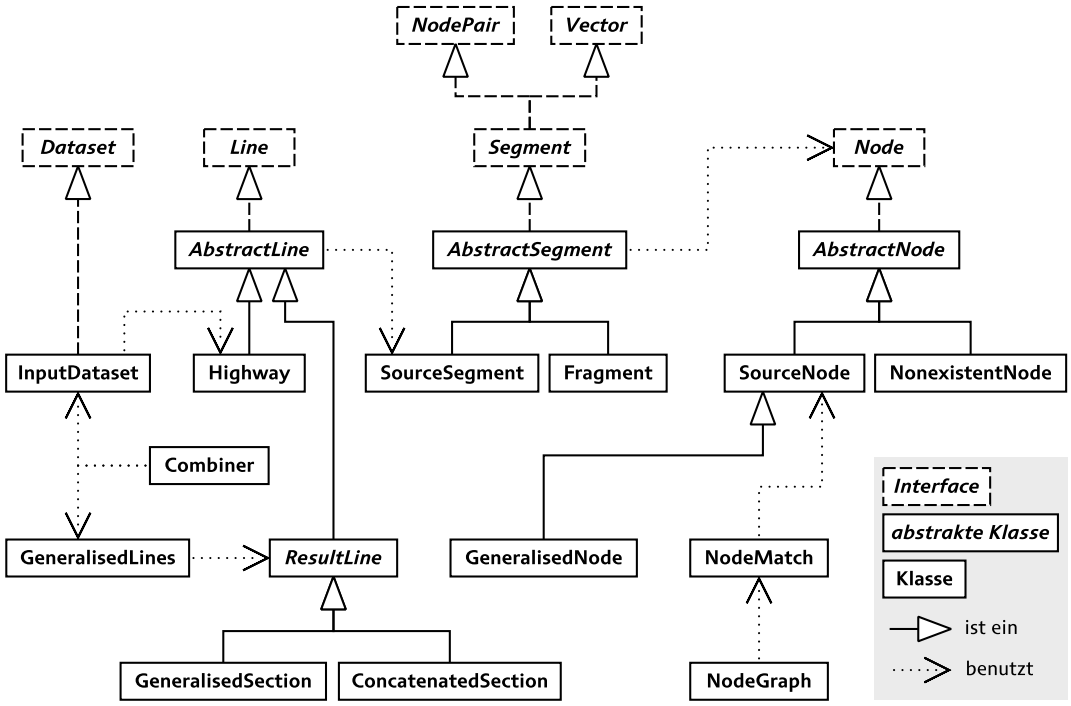

Abbildung 61: Klassenstrukturdiagramm für das Paket `comb` (aus Gründen der Übersichtlichkeit ist nur eine Auswahl der „benutzt“-Beziehungen dargestellt)

D Beispiele für problematische Kreuzungssituationen

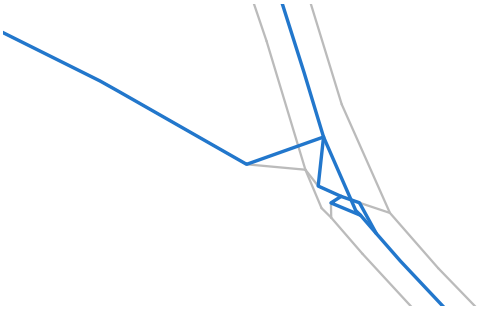

Abbildung 62: Ergebnis mit falsch attribuierten Abbiegefahrbahnen und misslungener Topologielücken-Beseitigung

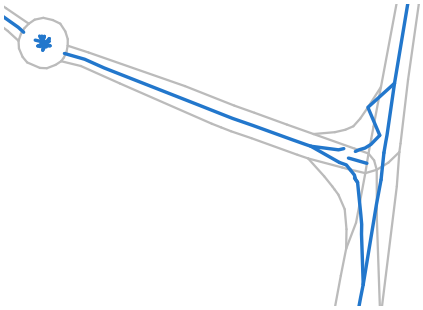

Abbildung 63: ungelöste Topologielücken bei Abbiegefahrbahn und Kreisverkehr

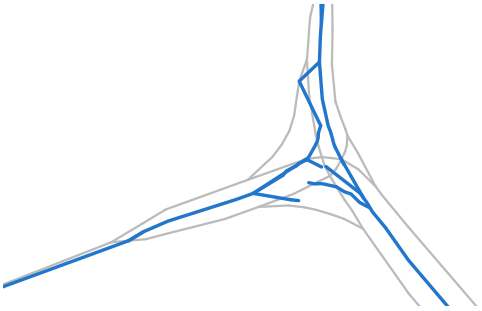

Abbildung 64: Abbiegefahrbahnen erschweren eine Lösung

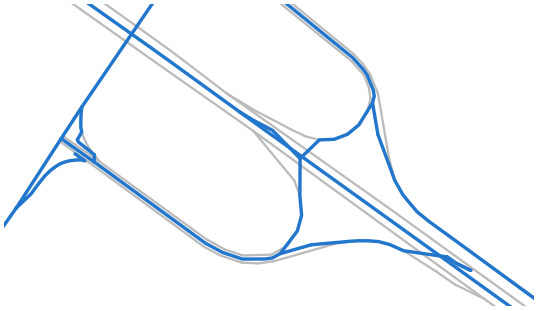

Abbildung 65: Probleme an Autobahn-Anschlussstelle

Literaturverzeichnis

Die Literaturangaben sind alphabetisch nach dem Kürzel sortiert. Das Kürzel wird gebildet aus den ersten drei Buchstaben des Nachnamens des Autors, bei mehreren Autoren aus jeweils den Anfangsbuchstaben der Nachnamen, bei Körperschaften aus einer mnemonisch gewählten Folge von Kleinbuchstaben; jeweils ergänzt durch die letzten beiden Ziffern des Jahres der Veröffentlichung.

Um ein eventuelles Nachschlagen zu erleichtern, sind die Referenzen wo immer möglich durch Angabe von Orten ergänzt, an denen eine Kopie des jeweiligen Werks am 1. März 2018 aufzufinden war. In der PDF-Ausgabe dieses Dokuments sind die URLs Hyperlinks. Die Signaturen beziehen sich auf die Bibliothek des Karlsruher Instituts für Technologie und deren Standort „Fachbibliothek HsKA“.

[adv08] Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland. *Dokumentation zur Modellierung der Geoinformation des amtlichen Vermessungswesens (GeoInfoDok). Erläuterungen zum ATKIS® Basis-DLM*. Version 6.0. 2008. 89 S.
URL: `https://www.geodatenzentrum.de/docpdf/Erlaeuterungen%20zum%20ATKIS%20Basis-DLM%206_0.pdf`

[All15] Andy Allan, Hrsg. *openstreetmap-carto: roads.mss*. Software-Quelltext. 21. Oktober 2015 22:51.
URL: `https://github.com/gravitystorm/openstreetmap-carto/blob/fe4f8d48757cf85bb692129b5d9755ba5cadbbee/roads.mss`

[All17] Andy Allan. *Thunderforest Landscape*. Webkarte. © Thunderforest, OpenStreetMap contributors. Kartendaten: Open Database Licence [cf. okf09]; Kartengrafik: Creative Commons Attribution-ShareAlike 2.0 [cf. cc04]. 2017.
URL: `https://www.thunderforest.com/maps/landscape/`

[asf04] Apache Software Foundation. *Apache License, Version 2.0*. 2004.
URL: `https://www.apache.org/licenses/LICENSE-2.0`

[BK01] Jürgen Bollmann und Wolf Günther Koch, Hrsg. *Lexikon der Kartographie und Geomatik*. Bd. 1: *A bis Karti*. Heidelberg: Spektrum Akademischer Verlag, 2001. 468 S.
SIGNATUR: 2013 CD 48, Fachgruppe geod 1

[Blo01] Joshua Bloch. *Effective Java*. Boston, Mass.: Addison-Wesley, 2001. 272 S.

[cc04] Creative Commons. *Attribution-ShareAlike 2.0 License (CC BY-SA 2.0)*. 2004.
URL: https://creativecommons.org/licenses/by-sa/2.0/

[cc07] Creative Commons. *Attribution-ShareAlike 3.0 License (CC BY-SA 3.0)*. 2007.
URL: https://creativecommons.org/licenses/by-sa/3.0/

[cc13a] Creative Commons. *Attribution-NoDerivatives 4.0 License (CC BY-ND 4.0)*. 2013.
URL: https://creativecommons.org/licenses/by-nd/4.0/

[cc13b] Creative Commons. *Attribution-ShareAlike 4.0 License (CC BY-SA 4.0)*. 2013.
URL: https://creativecommons.org/licenses/by-sa/4.0/

[CM05] Omair Chaudhry und William A Mackaness. *Rural and Urban Road Network Generalisation: Deriving 1:250,000 from OS MasterMap*. In: Proceedings of the 22nd International Cartographic Conference. 2005.
URL: https://hdl.handle.net/1842/1137

[CR03] Alan Cooper und Robert Reimann. *About Face 2.0: The Essentials of Interaction Design*. Indianapolis, Ind.: Wiley, 2003. 576 S.

[CTGV14] Paul Czioska, Frank Thiemann, Robin Giese und Hermann Vogt. *Ableitung eines routingfähigen Bahnnetzes aus nutzergenerierten Gleisdaten (OpenStreetMap) durch Generalisierung*. In: DGPF Tagungsband 23. Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation. 2014.
URL: http://www.ikg.uni-hannover.de/fileadmin/ikg/staff/publications/Konferenzbeitraege_abstract_review/czioska_dgpf2014.pdf

[Deo13] Justin Deoliveira. *GeoTools 10.0 Released*. Blog-Artikel. Open Source Geospatial Foundation. 20. September 2013.
URL: https://geotoolsnews.blogspot.de/2013/09/geotools-100-released.html

[Dÿk78] Edsger W Dijkstra. *An introduction to implementation issues*. EWD 656. Privat verbreitet. Nuenen, Niederlande, 1978.
URL: https://www.cs.utexas.edu/users/EWD/ewd06xx/EWD656.PDF

[Dÿk82] Edsger W Dijkstra. *Why numbering should start at zero*. EWD 831. Privat verbreitet. Nuenen, Niederlande, 1982.
URL: https://www.cs.utexas.edu/users/EWD/ewd08xx/EWD831.PDF

[EE99] David Eppstein und Jeff Erickson. *Raising Roofs, Crashing Cycles, and Playing Pool*. In: Discrete & Computational Geometry Jg. 22, Nr. 4 (1999), S. 569–592.
URL: http://jeffe.cs.illinois.edu/pubs/pdf/cycles.pdf

[EM00] Alistair J Edwardes und William A Mackaness. *Intelligent Generalisation of Urban Road Networks*. In: Proceedings of GIS Research UK 2000 Conference University of York. 2000, S. 81–85.
URL: https://www.geos.ed.ac.uk/homes/wam/EdwardesMack2000b.pdf

[F12] Dave F. *Advice & clarification of the railway tracks=* tag required*. Beitrag zur OpenStreetMap-Mailingliste „Tagging“, 7. August 2012 22:56.
URL: https://lists.openstreetmap.org/pipermail/tagging/2012-August/011066.html

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, Mass.: Addison-Wesley, 1995. 395 S.
SIGNATUR: 95 A 2017, Fachgruppe inf 3.76

[GJSB05] James Gosling, Bill Joy, Guy Steele und Gilad Bracha. *The Java™ Language Specification*. 3. Aufl. San Francisco, Calif.: Addison-Wesley, 2005.
URL: https://docs.oracle.com/javase/specs/

[HAS05] Frauke Heinzle, Karl-Heinrich Anders und Monika Sester. *Graph Based Approaches for Recognition of Patterns and Implicit Information in Road Networks*. In: Proceedings of the 22nd International Cartographic Conference. 2005.
URL: https://www.ikg.uni-hannover.de/fileadmin/ikg/staff/publications/Konferenzbeitraege_abstract_review/HeinzleAndersSester_ICC2005.pdf

[HGM02] Günter Hake, Dietmar Grünreich und Liqiu Meng. *Kartographie: Visualisierung raum-zeitlicher Informationen*. 8. Aufl. Berlin: de Gruyter, 2002. 617 S.
SIGNATUR: 70 A 1363(8), Fachgruppe geod 2.03

[HS04] Jan-Henrik Haunert und Monika Sester. *Using the Straight Skeleton for Generalisation in a Multiple Representation Environment*. In: 7th ICA Workshop on Generalisation and Multiple Representation. 2004.
URL: https://www.ikg.uni-hannover.de/fileadmin/ikg/staff/publications/Konferenzbeitraege_abstract_review/HaunertSester_ICA_workshop2004.pdf

[Hub04] Stefan Huber. *Straight Skeleton Definition*. © Creative Commons AttributionShareAlike 3.0 [cf. cc07]. 5. April 2004 09:14.
URL: https://commons.wikimedia.org/wiki/File%3AStraightSkeletonDefinition.png

[ica05] *Proceedings of the 22nd International Cartographic Conference*. 2005.
URL: https://icaci.org/icc2005/

[iho13] International Hydrographic Bureau. *Regulations of the IHO for International (INT) Charts and Chart Specifications of the IHO*. S 4. Version 4.4.0. Monaco, 2013. 435 S.
URL: https://iho.int/iho_pubs/standard/S-4/S-4_e4.4.0_EN_Sep13.pdf

[ito12] Ito World. *Railway track (tracks)*. Webkarte. © OpenStreetMap contributors. Kartendaten: Open Database Licence [cf. okf09].
URL: http://www.itoworld.com/map/14 (abgerufen in 2012; nicht mehr erreichbar)

[ito16] Ito World. *Railway track (passenger_lines)*. Webkarte. © OpenStreetMap contributors. Kartendaten: Open Database Licence [cf. okf09].
URL: http://product.itoworld.com/map/231 (abgerufen am 20. Januar 2016)

[JC04] Bin Jiang und C A Claramunt. *A Structural Approach to the Model Generalization of an Urban Street Network*. In: GeoInformatica Jg. 8, Nr. 2 (2004), S. 157–171.

[jts03] Vivid Solutions. *JTS Topology Suite Developer's Guide*. 2003. 12 S.
URL: https://raw.githubusercontent.com/locationtech/jts/c2e8e1d0695f60ed79e848e80a0c152da0c1d7db/doc/JTS%20Developer%20Guide.pdf

[jts15a] Vivid Solutions. *JTS 1.14 API – Package com.vividsolutions.jts.geom*. 2015.
URL: http://atetric.com/atetric/javadoc/com.vividsolutions/jts-core/1.14.0/com/vividsolutions/jts/geom/package-summary.html

[jts15b] Vivid Solutions. *JTS 1.14 API – Package com.vividsolutions.jts.index.strtree, Class STRtree*. 2015.
URL: http://atetric.com/atetric/javadoc/com.vividsolutions/jts-core/1.14.0/com/vividsolutions/jts/index/strtree/STRtree.html

[jts15c] Vivid Solutions. *JTS 1.14 API – Package com.vividsolutions.jts.math, Class Vector2D*. 2015.
URL: http://atetric.com/atetric/javadoc/com.vividsolutions/jts-core/1.14.0/com/vividsolutions/jts/math/Vector2D.html#method.summary

[Kla11] Ralf Klammer. *Integration von Generalisierungsfunktionalität für die automatische Ableitung verschiedener Levels of Detail von OpenstreetMap Webkarten*. Diplomarbeit. Technische Universität Dresden, 2011. 117 S.
URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-69596

[LM96] Alnoor Ladak und Roberto B Martinez. *Automated Derivation of High Accuracy Road Centrelines Thiessen Polygons Technique*. In: 1996 Esri International User Conference.
URL: https://web.archive.org/web/20130203084856id_/http://proceedings.esri.com:80/library/userconf/proc96/TO400/PAP370/P370.HTM

[Mat17] Alexander Matheisen. *Zusammenführung und Vereinheitlichung von Eisenbahn-Streckennetzdaten*. Vortrag auf der FOSSGIS-Konferenz. Universität Passau, 24. März 2017.
URL: https://frab.fossgis-konferenz.de/de/2017/public/events/5237

[ME13] Michal Migurski und Schuyler Erle. *Skeletron*. Software. 17. März 2013.
URL: https://github.com/migurski/Skeletron/tree/976b7029afa8e52fd723137781cb01234539a655

[Mig12] Michal Migurski. Webseite. 29. März 2012.
URL: http://mike.teczno.com/img/terrain-stamen-post/

[ML12] Ajay Mathur und Mary Lou. *Automatische Herstellung und Fortführung der topographischen Kartenwerke aus dem AAA-Datenmodell*. In: Kartographische Nachrichten Jg. 62, Nr. 5 (2012), S. 263–267.

[mm16] *maps.me*. Smartphone-Anwendung. Kartenausschnitt © My.com B.V. (Mail.Ru Group), OpenStreetMap contributors. Kartendaten: Open Database Licence [cf. okf09]; Kartengrafik: Apache License, Version 2.0 [cf. asf04]. 2016.
URL: https://mapsme.de/

[MM99] William A Mackaness und Gordon A Mackechnie. *Automating the Detection and Simplification of Junctions in Road Networks*. In: GeoInformatica Jg. 3, Nr. 2 (1999), S. 185–200.
URL: https://www.geos.ed.ac.uk/homes/wam/MackMack1999.pdf

[Nei15] Pascal Neis. *OSMstats: Report for Nov, 07th 2015*. Webseite.
URL: https://osmstats.neis-one.org/?item=changesets&date=7-11-2015

[nma17] Norwegian Mapping Authority. *N250 Raster*. Topographische Übersichtskarte 1 : 250 000. © Kartverket. Kartendaten und Kartengrafik: Creative Commons Attribution-ShareAlike 4.0 [cc13b]. 2017.
URL: https://kartkatalog.geonorge.no/metadata/uuid/d2ae29bd-0692-40de-a173-833afcddfe22

[NZ12] Pascal Neis und Alexander Zipf. *Analyzing the Contributor Activity of a Volunteered Geographic Information Project – The Case of OpenStreet-*

Map. English. In: ISPRS International Journal of Geo-Information Jg. 1, Nr. 2 (2012), S. 146–165.
URL: http://www.mdpi.com/2220-9964/1/2/146/

[NZZ12] Pascal Neis, Dennis Zielstra und Alexander Zipf. *The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007–2011*. English. In: Future Internet Jg. 4, Nr. 4 (2012).
URL: http://www.mdpi.com/1999-5903/4/1/1/

[OHSZ10] Martin Over, Mohammed El Hachadi, Arne Schilling und Alexander Zipf. *Automatisierte Generalisierungsverfahren von 3D-Stadt-und Landschaftsmodellen*. In: Angewandte Geoinformatik 2010. Beiträge zum 22. AGIT-Symposium, S. 999–1003.
URL: https://www.geog.uni-heidelberg.de/md/chemgeo/geog/gis/over-etal.pdf

[okf09] Open Knowledge Foundation. *Open Data Commons Open Database License (ODbL) v1.0*. 2009.
URL: https://opendatacommons.org/licenses/odbl/1.0/

[osm10] OpenStreetMap. *OSM Inspector/Views/Highways*. Wiki-Artikel. 5. Januar 2010 15:45.
URL: https://wiki.openstreetmap.org/w/index.php?title=OSM_Inspector/Views/Highways&oldid=400792#Overview

[osm12a] OpenStreetMap. *DE:Open Database License*. Wiki-Artikel. 15. September 2012 18:22.
URL: https://wiki.openstreetmap.org/w/index.php?title=DE:Open_Database_License&oldid=809285

[osm12b] OpenStreetMap. *Highway link*. Wiki-Artikel. 24. Oktober 2012 12:19.
URL: https://wiki.openstreetmap.org/w/index.php?title=Highway_link&oldid=824541

[osm12c] OpenStreetMap. *Tile disk usage*. Wiki-Artikel. 31. Dezember 2012 07:46.
URL: https://wiki.openstreetmap.org/w/index.php?title=Tile_disk_usage&oldid=849586

[osm17a] *OpenSeaMap*. © OpenStreetMap contributors. Kartendaten: Open Database Licence [cf. okf09]; Kartengrafik: Creative Commons Attribution-ShareAlike 2.0 [cf. cc04]. 2017.
URL: https://openseamap.org/

[osm17b] *OpenStreetMap-Karte — deutscher Kartenstil*. © OpenStreetMap contributors. Kartendaten: Open Database Licence [cf. okf09]; Kartengrafik: Creative Commons Attribution-ShareAlike 2.0 [cf. cc04]. 2017.
URL: https://www.openstreetmap.de/karte.html

[osm18] *OpenStreetMap-Karte — Standard-Darstellung*. © OpenStreetMap contributors. Kartendaten: Open Database Licence [cf. okf09]; Kartengrafik: Creative Commons Attribution-ShareAlike 2.0 [cf. cc04]. 2018.
URL: https://www.openstreetmap.org/

[PM17] Hippolyte Pruvost und Peter Mooney. *Exploring Data Model Relations in OpenStreetMap*. In: Future Internet Jg. 9, Nr. 4 (2017).
URL: http://www.mdpi.com/1999-5903/9/4/70

[Ram72] Urs Ramer. *An Iterative Procedure for the Polygonal Approximation of Plane Curves*. In: Computer Graphics and Image Processing Jg. 1, Nr. 3 (1972), S. 244–256.

[RKE06] *Progress in Spatial Data Handling*. 2006.
SIGNATUR: 2009 A 2945, Fachgruppe geog 0.5

[RSV02] Philippe Rigaux, Michel Scholl und Agnès Voisard. *Spatial Databases With Application To GIS*. San Francisco, Calif.: Morgan Kaufmann, 2002. 440 S.
SIGNATUR: 2002 A 1676, Fachgruppe inf 7.2

[RT09] Frederik Ramm und Jochen Topf. *OpenStreetMap. Die freie Weltkarte nutzen und mitgestalten*. 2. Aufl. Berlin: Lehmanns Media, 2009.
SIGNATUR: 2008 A 3267(2), Fachgruppe geod 1.4

[Sch09] Sebastian Schwarz. *Öffentlicher Personennahverkehr in OpenStreetMap*. Diplomarbeit. Hochschule Karlsruhe – Technik und Wirtschaft, 2009. 149 S.
URL: https://kahlfrost.de/diplom/

[Sch12] Mareike Schoof. *ATKIS-Basis-DLM und OpenStreetMap – Ein Datenvergleich anhand ausgewählter Gebiete in Niedersachsen*. In: Kartographische Nachrichten Nr. 1 (2012), S. 20–26.

[sgk02] Schweizerische Gesellschaft für Kartografie. *Topografische Karten – Kartengrafik und Generalisierung*. Selbstverlag (CD-ROM). 2002. 121 S.

[Sny87] John P Snyder. *Map projections—A Working Manual*. Professional paper 1395. Washington, D.C.: U.S. Geological Survey, 1987. 383 S.
URL: https://pubs.er.usgs.gov/publication/pp1395

[SP11] Sabine Stengel und Sascha Pomplun. *Die freie Weltkarte OpenStreetMap – Potenziale und Risiken*. In: Kartographische Nachrichten Jg. 61, Nr. 3 (2011), S. 115–120.

[Tho05] Stuart Thom. *A Strategy for Collapsing OS Integrated Transport Network Dual Carriageways*. In: 8th ICA Workshop on Generalisation and Multiple Representation. 2005.
URL: https://www.researchgate.net/publication/228987992

[Tho06a] Stuart Thom. *Conflict Identification and Representation for Roads Based on a Skeleton*. In: Progress in Spatial Data Handling. 2006, S. 659–680.
SIGNATUR: 2009 A 2945, Fachgruppe geog 0.5

[Tho06b] Robert C Thomson. *The 'stroke' concept in Geographic Network Generalization and Analysis*. In: Progress in Spatial Data Handling. 2006, S. 681–697.
SIGNATUR: 2009 A 2945, Fachgruppe geog 0.5

[Top09] Jochen Topf. *Das OpenStreetMap-Projekt: Geodaten von und für jedermann*. In: Kartographische Nachrichten Jg. 59, Nr. 1 (2009), S. 47–49.

[TR95] Robert C Thomson und Dianne E Richardson. *A Graph Theory Approach to Road Network Generalization*. In: Proceedings of the 16th International Cartographic Conference. 1995, S. 1871–1880

[TR99] Robert C Thomson und Dianne E Richardson. *The 'Good Continuation' Principle of Perceptual Organization applied to the Generalization of Road Networks*. In: Proceedings of the 19th International Cartographic Conference. 1999, S. 1215–1223.
URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.202.4737&rep=rep1&type=pdf`

[UD06] Sabine Urbanke und Katja Dieckhoff. *Das AdV-Projekt ATKIS-Generalisierung, Teilprojekt Modellgeneralisierung*. In: Kartographische Nachrichten Jg. 56, Nr. 4 (2006), S. 191–196.

[vKP98] Marc van Kreveld und Jarno Peschier. *On the Automated Generalization of Road Network Maps*. In: Proceedings of the 3rd International Conference on GeoComputation. 1998.
URL: `http://www.geocomputation.org/1998/21/gc_21.htm`

[wp13a] Wikipedia. *Anschlussstelle (Autobahn)*. Wiki-Artikel. 16. April 2013 18:42.
URL: `http://de.wikipedia.org/w/index.php?title=Anschlussstelle_(Autobahn)&oldid=117592609#Einpunktvollanschlussstelle_(SPUI)`

[wp13b] Wikipedia. *Landau-Symbole*. Wiki-Artikel. 4. April 2013 17:30.
URL: `https://de.wikipedia.org/w/index.php?title=Landau-Symbole&oldid=117035768#Definition`

[wp13c] Wikipedia. *Straight skeleton*. Wiki-Artikel. 23. März 2013 19:43.
URL: `https://en.wikipedia.org/w/index.php?title=Straight_skeleton&oldid=546598291`

[wp16] Wikipedia. *Richtungs- und Linienbetrieb*. Wiki-Artikel. 20. August 2016 18:35.
URL: `https://de.wikipedia.org/w/index.php?title=Richtungs-_und_Linienbetrieb&oldid=157212328`