

**Masterarbeit**  
**Studiengang Praktische Informatik**

**Clusterbildung von Geoobjekten im Straßennetz**

Anonymisierung durch Aggregation

April 2022

Georg Ottinger

Matrikel-Nr. 5338450

Betreuer: Prof. Dr. Christian Icking, Dr. Lihong Ma  
Lehrgebiet Kooperative Systeme  
Fakultät für Mathematik und Informatik

---



## Kurzfassung

Ziel dieser Arbeit ist eine Methode zur Clusterbildung von Geoobjekten, wie z. B. Häusern, zu entwickeln, welche die Entferungen im Straßenverlauf berücksichtigt. Da solche Cluster einer Aggregation von mehreren Objekten entsprechen, können diese, gemäß den gesetzlichen Vorgaben, als ein Mittel zur Anonymisierung von räumlichen, personenbezogenen Daten herangezogen werden. Die Clusterbildung erfolgt in einem mehrstufigen Verfahren, welches die Geoobjekte und ihre Distanzen zueinander in einen Graphen überführt. Für jedes Geoobjekt wird ein Teilgraph mit seinen  $k$  nächsten Nachbarn ermittelt. Anschließend wird mit einer Auswahl dieser Teilgraphen der gesamte Graph eindeutig abgedeckt. Als Zielfunktion für diese Auswahl dient die Minimierung der Distanzen der zentralen Geoobjekte zu den nächsten Nachbarn.

**Schlagwörter:** Anonymisierung, Aggregation, Datensparsamkeit,  $k$  nächste Nachbarn, Distanzen im Graph, Clusterbildung, Straßennetz, Straßengraph, OpenStreetMap, PostGIS, räumliche Datenbank.

## Abstract

This work aims to develop a method for clustering geo-objects, such as houses, which takes into account distances within the road network. Since such clusters correspond to an aggregation of multiple objects, they can be used to anonymize personal spatial data, according to legal requirements. Clustering involves a multi-stage procedure, which transforms geo-objects and corresponding distances into a graph. A subgraph with its  $k$  nearest neighbors is determined for each geo-object. Subsequently, a subset of subgraphs is selected to achieve a non-overlapping cover of the entire graph. This selection's objective function is to minimize distances of central geo-objects to nearest neighbors.

**Keywords:** Anonymization, Aggregation, Privacy by Design,  $k$  Nearest Neighbours, Distances within a Graph, Clustering, Road Network, Road Graph, OpenStreetMap, PostGIS, Spatial Database.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Datenschutz im Bedarfsverkehr als Motivation . . . . .	1
1.2 Gesetzliche Grundlagen der Aggregation von Daten . . . . .	1
1.3 Ausgangslage . . . . .	3
<b>2 Verortung der Aufgabenstellung</b>	<b>3</b>
2.1 Aufgabendefinition . . . . .	5
2.2 Similarity-Partitioning . . . . .	8
2.3 Einbindung des Straßennetzes . . . . .	10
2.4 Benachbarte Probleme . . . . .	12
<b>3 Datenquellen</b>	<b>16</b>
3.1 OpenStreetmap . . . . .	16
3.2 Basemap.at . . . . .	17
<b>4 PostgreSQL und PostGIS</b>	<b>18</b>
4.1 Similarity-Partitioning-Implementierung . . . . .	18
4.2 Räumliche Indexe . . . . .	21
4.3 GeoHash . . . . .	22
4.4 Koordinatenreferenzsystem . . . . .	23
4.5 Import von OSM-Daten in PostGIS . . . . .	25
4.6 Export des GSD-Graphen . . . . .	27
<b>5 GOA Überblick</b>	<b>37</b>
<b>6 GOA Berechnung</b>	<b>41</b>
6.1 Bestimmen der ( $a$ -anonymen) NN Teilgraphen . . . . .	41
6.2 Vorauswahl der eindeutig bestimmten Teilgraphen . . . . .	42
6.3 Optimierte Auswahl der Teilgraphen . . . . .	43
6.4 Zuordnung der Residuen . . . . .	46
6.5 Neuberechnung der zentralen Knoten . . . . .	48
6.6 „Stehlen“ von Knoten . . . . .	48
6.7 Berechnung der „minsum“-Zentren . . . . .	49
6.8 Erstellung der Territorien in der Ebene . . . . .	49
<b>7 Verwendete Algorithmen</b>	<b>52</b>
7.1 Bestimmen der $k$ nächsten Nachbarn . . . . .	52

7.2	Genetische Algorithmen . . . . .	57
<b>8</b>	<b>Evaluierung des GOA-Verfahrens</b>	<b>60</b>
8.1	Visuelle Analyse . . . . .	60
8.2	Quantitative Betrachtung . . . . .	65
8.3	Kritische Betrachtung der Häuser als Haushalte . . . . .	68
<b>9</b>	<b>Verbesserungspotential</b>	<b>68</b>
9.1	Laufzeitverbesserungen . . . . .	69
9.2	Qualitative Verbesserungen . . . . .	71
9.3	Alternativer Ansatz . . . . .	73
<b>10</b>	<b>Einsatz in der Praxis</b>	<b>74</b>
<b>11</b>	<b>Zusammenfassung und Ausblick</b>	<b>75</b>
<b>A</b>	<b>Anhang</b>	<b>77</b>
A.1	Softwareinstallation unter Ubuntu 20.04 . . . . .	77
A.2	Download von OpenStreetMap-Daten . . . . .	77
A.3	Entfernen von Elementen außerhalb der Administrationsgrenzen . . . . .	78
A.4	Commandline GOA Tools . . . . .	78
A.5	Laufzeit Messungen . . . . .	79
A.6	Auffinden von nicht zusammenhängenden Territorien . . . . .	80
A.7	Weitere Ergebnisvisualisierungen . . . . .	81

## Danksagung

Ein großes Dankeschön möchte ich meinen Betreuern Herrn Christian Icking und Frau Lihong Ma aussprechen, welche mich über mehrere Jahre hinweg von der ersten Ideenskizze bis zur Umsetzung dieser Arbeit begleitet haben. Tobias Haider möchte ich für die Inspiration zu dieser Arbeit und seinen unermüdlichen Einsatz für die Mobilitätswende danken. Herrn Alexander Dommnich, meiner Schwester Katharina Ottinger sowie meiner Mutter Marianne Ottinger danke ich für das Lektorat. Besonderer Dank gilt meiner Partnerin Birgit Walter, welche mich in vielfacher Weise in der Umsetzung dieser Arbeit unterstützt und mir den notwendigen Freiraum in unserem Familienalltag ermöglicht hat. Gewidmet ist diese Arbeit unseren beiden Söhnen Ilja und Arik, in der Hoffnung, dass diese Arbeit einen kleinen Beitrag zur Umsetzung des einen oder anderen Bedarfsverkehrssystems leistet.



# 1 Einleitung

## 1.1 Datenschutz im Bedarfsverkehr als Motivation

Die Motivation zur vorliegenden Arbeit besteht darin, ein Verfahren zu entwickeln, welches statistische Daten lokaler Bedarfsverkehrssysteme im Sinn des Datenschutzes anonymisiert. Die betreffenden Bedarfsverkehrssysteme werden vorwiegend im ländlichen Raum betrieben und sie verfügen meistens über keine festgelegten Haltestellen. Da es Fahrgästen freisteht, wo sie ein- und aussteigen, ist es die Regel, dass Fahrten direkt vor ihrem jeweiligen Haus bzw. ihrer Wohnung beginnen oder enden.

Die FahrerInnen der Kleinbusse verbuchen zu Statistikzwecken auf mobilen Endgeräten das Zu- und Aussteigen der Fahrgäste. Neben der Uhrzeit und den gefahrenen Kilometern werden auch Start- und Endkoordinaten der Fahrt erhoben. Diese gesammelten Statistiken werden unter anderem dazu verwendet, Entscheidungsgrundlagen für lokale politische Akteure zu generieren, um vorhandene Verkehrskonzepte weiterzuentwickeln.

Im Sinne des Datenschutzes dürfen solche Daten – sofern nicht ein explizites Einverständnis vorliegt – jedoch nicht weiter verarbeitet werden, falls eine Zuordnung von einer konkreten Fahrt zu einem Fahrgast möglich ist. Diese würde die Anonymität der Fahrgäste verletzen und einen Eingriff in ihre Privatsphäre darstellen. Zum Schutz der Anonymität dürfen demnach die erhobenen Koordinaten nicht einem einzelnen Haus zuordenbar sein. Durch Aggregation von mehreren Häusern wird eine Unschärfe der Datensätze erzeugt, sodass den rechtlichen Grundlagen des Datenschutzes Genüge getan wird.

## 1.2 Gesetzliche Grundlagen der Aggregation von Daten

Die Aggregation von Daten ist ein notwendiges Mittel, damit die Zuordnung von Fahrgastdaten zu einer spezifischen Person verunmöglich wird. Die Datenschutz-Grundverordnung verwendet in diesem Zusammenhang den Begriff der „personenbezogenen Daten“, der wie folgt definiert ist:

„Personenbezogene Daten“ [sind] alle Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person (im Folgenden ‚betroffene Person‘) beziehen; als identifizierbar wird eine natürliche Person angesehen, die direkt oder indirekt, insbesondere mittels Zuordnung zu einer Kennung wie einem Namen, zu einer Kennnummer, zu Standortdaten, zu einer Online-Kennung oder zu einem oder mehreren besonderen Merkmalen, die Ausdruck der physischen, physiologischen, genetischen, psychischen, wirtschaftlichen, kulturellen oder sozialen Identität dieser

natürlichen Person sind, identifiziert werden kann“ (Art. 4 Nr. 1 DSVGO) [43].

„Geht man grundsätzlich davon aus, dass Geodaten bzw. Geoinformationen personenbezogene Daten sind, kann die Frage, ob Dritten Zugang gewährt werden darf, in eine Abwägung führen. In dieser sind gemäß den gesetzlichen Vorgaben die Interessen des/der Betroffenen dem öffentlichen Interesse an der Bekanntgabe gegenüberzustellen und zu gewichten“ [26, S. 10 f.].

Das Ziel der Aggregation von Geoobjekten ist es, dass die erhobenen Fahrgastdaten, aus einer juristischen Perspektive gesehen, nicht mehr als „personenbezogene Daten“ gelten bzw. das öffentliche Interesse überwiegt. Dazu muss der Raumbezug der erhobenen Daten „grob“ genug gefasst sein, damit eine Re-Identifikation der infrage kommenden Personen nicht mehr möglich ist [41, vgl. S. 218].

Dabei ist zu beachten, dass die Definition von „nicht mehr zuordenbar“ keine technische ist, sondern auf einer juristischen Ebene in den Aufgabenbereich des Gesetzgebers bzw. der Rechtsprechung fällt. Der konkrete Anlassfall für die Erstellung dieser Arbeit ist ein Bedarfsverkehrssystem in Österreich, damit muss klarerweise österreichisches bzw. EU-Recht zur Anwendung kommen. In der Literaturrecherche konnten allerdings nur Berichte, Behördenleitfäden und Studien für Deutschland ausfindig gemacht werden, welche sich mit diesem Thema beschäftigen. Wie immer in juristischen Fragen gilt es die verschiedenen Interessen abzuwägen.

„Außer Zweifel steht, dass der Datenschutz gerade bei kleinräumigen Geodaten (z. B. Statistikdaten) gewahrt sein muss, damit ein Rückschluss auf einzelne Personen ausgeschlossen ist“ [21, S. 32].

„Es gibt keine eindeutigen rechtlichen Vorgaben, wann von einer Anonymität hinreichend gewährleistenden Aggregierung gesprochen werden kann. Unstreitig ist, dass eine Zusammenfassung von zwei Personeneinheiten zu einer Merkmalseinheit noch nicht ausreicht. Entsprechend der statistikrechtlichen Praxis kann man davon ausgehen, dass bei einer Zusammenführung von mindestens vier Personeneinheiten zu einem Datensatz der Personenbezug hinreichend verschleiert wird“ [45, S. 25].

„Um mehr Rechtssicherheit zu erreichen, sollten Auflösungsschwellen, ab denen, bei Personenbeziehbarkeit eines Datums, eine persönlichkeitsrechtliche Relevanz gegeben ist, für die gängigen Georeferenzierungen abgestimmt werden. Keine Schutzbedürftigkeit könnte aus datenschutzrechtlicher Sicht bestehen bei [...] mindestens auf vier Haushalte aggregierten Informationen“ [21, S. 31].

Auf Grund dieser Einschätzungen wird in der vorliegenden Arbeit versucht, als Standardparameter, mindestens fünf bewohnte Gebäude für die Aggregation heranzuziehen.

### 1.3 Ausgangslage

Die Idee zu der vorliegenden Masterarbeit entstammt einem Gespräch mit Tobias Haider von der Firma „mobyome“. Diese Firma begleitet Gemeinden in Form von Mobilitätsworkshops und stellt Werkzeuge für die Umsetzung von Bedarfsverkehrssystemen zur Verfügung. Die sogenannte „Aufzeichnungs-App“ ermöglicht eine Protokollierung der Fahrten durch den Fahrer oder die Fahrerin beim Ein- und Aussteigen von Fahrgästen. Damit die einzelnen Fahrten nicht direkt einem Haus zugeordnet werden müssen, werden sogenannte „virtuelle Haltestellen“ (siehe Abb. 1 auf Seite 4) verwendet.

Die virtuellen Haltestellen sind Zentren von ungewichteten Voronoi-Regionen, in welchen sich immer eine Mindestanzahl von Häusern befinden muss. In der Praxis wird derzeit bei der Erstellung der virtuellen Haltestellen darauf geachtet, dass sich immer mindestens 5 Häuser in einer Region befinden. Gleichzeitig wird versucht, möglichst viele Zentren zu erstellen, um die Genauigkeit und Aussagekraft der Statistiken hoch zu halten. Diese Zentren werden vorab für die betreffende Gemeinde angelegt. In der Phase der Datenerhebung erfolgt ein Mapping der GPS-Koordinaten über die Voronoi-Region auf deren Zentrum. Dieses Mapping geschieht unmittelbar nach der Datenerhebung am Endgerät, damit gewährleistet wird, dass nur anonymisierte Datensätze übertragen werden. Für die weitere Datenverarbeitung wird nur die Koordinate dieses Zentrums – der virtuellen Haltestelle – gespeichert. Kurz gesagt werden die Fahrgastdaten durch Aggregation anonymisiert.

Mit dieser Arbeit soll ein Verfahren entwickelt werden, welches das beschriebene Vorgehen automatisiert und unter der Berücksichtigung des Datenschutzes eine Aggregation der Daten vornimmt, ohne dabei die statistische Aussagekraft wesentlich herabzusetzen.

## 2 Verortung der Aufgabenstellung

Im Falle der Anonymisierung von Fahrgastdaten wird, wie in der Motivation beschrieben, verlangt, dass mehrere Häuser in einer Region aggregiert werden. Allgemein kann jedoch davon gesprochen werden, dass stationäre Geoobjekte und deren Einzugsbereiche, welche im Straßennetz zueinander in Beziehung stehen, zusammengefasst werden. Für die Speicherung und weitere Verarbeitung der Fahrgastdaten sind diese nun nicht mehr exakt einem Geoobjekt zugeordnet, sondern einem Cluster, womit dem Datenschutz Genüge getan ist.

Aus dieser Überlegung lässt sich folgern, dass ein gewisses Gebiet (z. B. ein

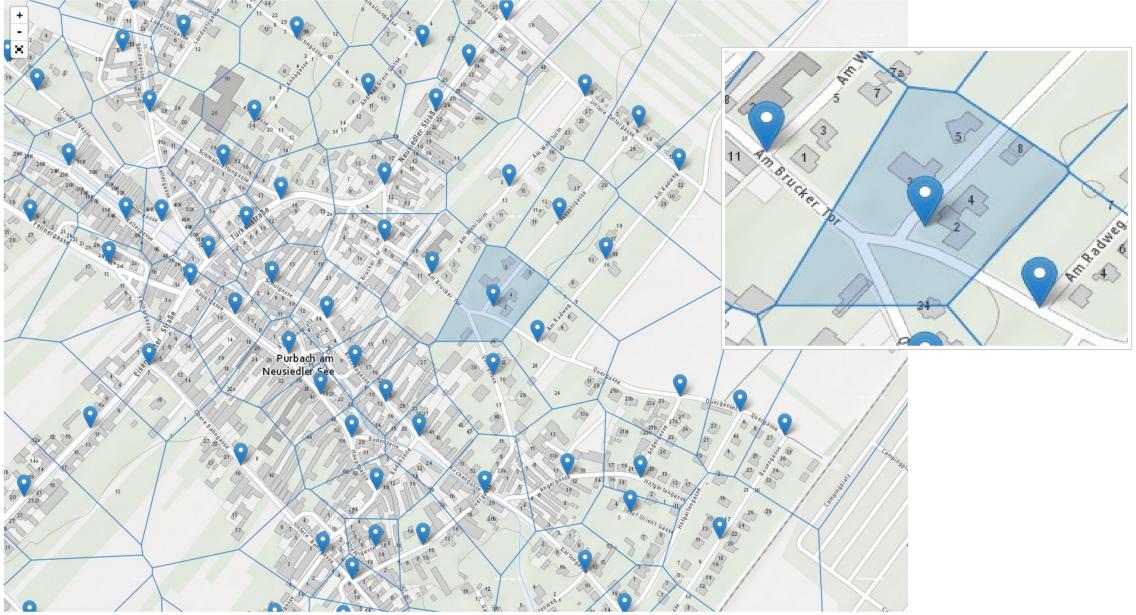


Abb. 1: Virtuelle Haltestellen der Aufzeichnungs-App [50]

Gemeindegebiet) in mehrere kleine Einheiten aufgeteilt werden muss. Damit wird offensichtlich, dass das hier beschriebene Problem Ähnlichkeiten zum Problem der Gebietsplanung (engl. territory design problem) hat. „Gebietsplanung bezeichnet die Aggregation kleiner Einheiten (Basisgebiete) zu übergeordneten Einheiten, den Distrikten, Territorien oder Gebieten, wobei jene vorher definierten Kriterien genügen sollen“ [71, S. 12].

Das Problem der Gebietsplanung steht wiederum in einem Verwandtschaftsverhältnis zum Problem der Standortplanung (engl. facility location problem). Bei der Gebietsplanung sind die Zentren – sofern vorhanden – vorher bekannt und werden im Laufe der Planung nicht verändert, während bei der Standortplanung die Zentren gewählt werden können [71, S. 12].

Das Gebietsplanungsproblem wird seit vielen Jahren untersucht und eine prominente Anwendung, neben vielen anderen, ist die Einteilung von Wahlkreisen [28]. Die meisten Anwendungen versuchen kompakte, zusammenhängende und ausbalancierte Gebiete zu erstellen [44, S. 3 f.].

Während sich die Eigenschaften zusammenhängend und ausbalanciert gut formal beschreiben lassen, gibt es in der Literatur unterschiedliche Ansätze, die Kompaktheit eines Gebietes mathematisch auszudrücken. In [73, S. 33 ff.] gibt es eine Übersicht zu einer Reihe von gebräuchlichen Kompaktheitsmaßen.

Es stellt sich die Frage, welche zentralen Eigenschaften für den vorliegenden Anwendungsfall wesentlich sind. Primär wird die Anonymisierung von Fahrgastdaten

zum Zwecke der statistischen Weiterverarbeitung angestrebt. Daher sind die Anonymität und die statistische Weiterverarbeitbarkeit die zwei primären Eigenschaften, welche es zu berücksichtigen gilt.

Zusammenfassend lässt sich das vorliegende Problem demnach als ein Problem der Standortplanung unter der Berücksichtigung von Anonymität und der statistischen Weiterverarbeitbarkeit definieren. Etwas später soll in dieser Arbeit gezeigt werden, dass die als Zweites genannte Forderung als ein Spezialfall der Kompaktheit verstanden werden kann.

Die in der allgemeinen Gebietsplanung üblichen Forderungen nach zusammenhängenden und ausbalancierten Gebieten sind, obwohl diese Ziele wünschenswert sind, allerdings nur von untergeordneter Bedeutung.

Wichtig ist an dieser Stelle auch die Beobachtung, dass im vorliegenden Fall die Erstellung von Gebietszentren mit keinerlei monetären Kosten, verglichen mit der Errichtung eines Firmenstandorts, verbunden ist, daher können beliebig viele Zentren erstellt werden, solange die geforderte Anonymität gewährleistet bleibt. Im Gegensatz dazu ist vielfach in der „klassischen“ Standortplanung die Anzahl der Zentren vorgegeben oder nach oben hin begrenzt. Verfahren, welche zum Beispiel  $p$ -median oder  $k$ -facilities in ihrem Namen tragen, deuten an, dass ein Gebiet in maximal  $p$  bzw.  $k$  Teile aufgeteilt werden soll.

## 2.1 Aufgabendefinition

Ausgehend von den Definitionen der Gebietsplanung [44] soll nun versucht werden, einen ersten Satz an Definitionen für das vorliegende Problem zu entwickeln. Dabei werden Begriffe, welche passend erscheinen, entliehen und deren deutsche Übersetzungen aus [73] finden Verwendung.

### Definition 1 (Basisgebiet)

*Ein Basisgebiet  $B_i$  entspricht der kleinsten zu aggregierenden Einheit, es wird durch den zentralen Punkt  $b_i$  repräsentiert. Das Basisgebiet entspricht der Voronoi-Region von  $b_i$ . Die Menge  $B_{all}$  beinhaltet alle Basisgebiete.*

In der Regel wird jedem Geoobjekt bzw. jedem Haus ein Basisgebiet zugeordnet. Der zentrale Punkt muss dabei nicht zwingend der Koordinate etwa des Hauses entsprechen, sondern kann auch ein Punkt sein, welcher sich zum Beispiel auf dem Straßennetz befindet (siehe Abschnitt 2.3 auf Seite 10).

### **Definition 2 (Territorium)**

*Ein Territorium  $T_j \subset B_{all}$  ist die Vereinigung von mehreren Basisgebieten. Ein Territorium wird durch einen zentralen Punkt  $c_j$  eines der Basisgebiete repräsentiert. Obwohl erwünscht, muss ein Territorium nicht zwingend zusammenhängend sein.*

### **Definition 3 (Vollständige Zuweisung)**

*Vollständige Zuweisung bedeutet, dass jedes Basisgebiet genau einem Territorium zugewiesen werden muss. Somit teilen die Territorien die Menge  $V$  der Basisgebiete komplett auf. Im Falle von  $n$  Territorien gilt  $T_1 \cup \dots \cup T_n = B_{all}$  und  $T_i \cap T_j = \emptyset, \forall i \neq j$ .*

### **Definition 4 ( $a$ -anonym)**

*$a$ -anonym bedeutet, dass jedem Territorium mindestens  $a$  Basisgebiete zugeordnet sind  $|T_i| \geq a$ .*

Zur Veranschaulichung: Eine 1-Anonymität bedeutet, dass keine Aggregation stattgefunden hat, da jedes Geoobjekt zu einem Cluster der Größe 1 zusammengefasst wurde und somit jedes Geoobjekt wieder für sich steht. Daher bedeutet 1-anonym nicht anonym.

Die Wahl des Parameters  $a$  wird im Wesentlichen von gesetzlichen Vorgaben bestimmt (siehe Abschnitt 1.2 auf Seite 1). Die Evaluierung der Testdatensätze erfolgt im weiteren Verlauf dieser Arbeit immer mit dem Wert  $a = 5$ .

Um der Forderung der statistischen Weiterverarbeitbarkeit Genüge zu tun, muss zunächst überlegt werden, welche statistischen Daten in dem vorgestellten Anwendungsfällen erhoben werden. Es handelt sich dabei um die einzelnen Fahrten der jeweiligen Fahrgäste mit Ein- und Ausstiegskoordinaten sowie dem Datum und der Uhrzeit vom Beginn und Ende der Fahrt. Ohne eine Anonymisierung würden diese Ein- und Ausstiegskoordinaten einfach als Punkte  $e, a \in V$  mit der Fahrstrecke  $d(e, a)$  gespeichert und weiterverarbeitet. Durch die Geoobjekte-Aggregation (kurz GOA) werden nunmehr nur die Indizes der Territorien gespeichert, *ie* und *ia*,  $e \in T_{ie}, a \in T_{ia}$ , und die Fahrstrecke wird mit  $d(c_{ie}, c_{ia})$  über die Zentren der betreffenden Territorien berechnet. Um den statistischen Fehler, der dadurch entsteht, klein zu halten, ist es wünschenswert, die Summe der Abstände der Zentren der Basisregionen zu den Zentren der ihnen zugeordneten Territorien klein zu halten. In der Gebiets- bzw. Standortplanung wird diese Zielfunktion „median“ oder „minisum“ genannt [71]. Damit wird ersichtlich, dass es sich bei dieser Forderung um ein in der Standortplanung gebräuchliches Kompaktheitsmaß handelt und demnach für die GOA die Kompaktheit wie folgt definiert werden kann:

### **Definition 5 (Kompaktheitsmaß)**

Das Kompaktheitsmaß eines Territoriums  $komp(T_j)$  berechnet sich aus der Summe der Distanzen der Zentren der zugeordneten Basisgebiete zum Zentrum des Territoriums.

$$komp(T_j) = \sum d(b_i, c_j), \forall i, B_i \in T_j$$

### **Definition 6 (Kompaktheit)**

Unter Kompaktheit wird die Minimierung der Summe aller Kompaktheitsmaße der Territorien verstanden.

Es ist anzumerken, dass es im vorliegenden Fall relativ einfach ist, eine schlüssige Definition des Kompaktheitsmaßes zu finden, je nach Anwendung können in der Gebietsplanung jedoch unterschiedliche Kompaktheitsmaße Anwendung finden [73, 83].

### **Definition 7 (Geoobjekte-Aggregation)**

Die Geoobjekte-Aggregation (kurz GOA) teilt alle Basisgebiete  $B_{all}$  in Territorien auf, sodass diese vollständig zugewiesen,  $a$ -anonym und kompakt sind.

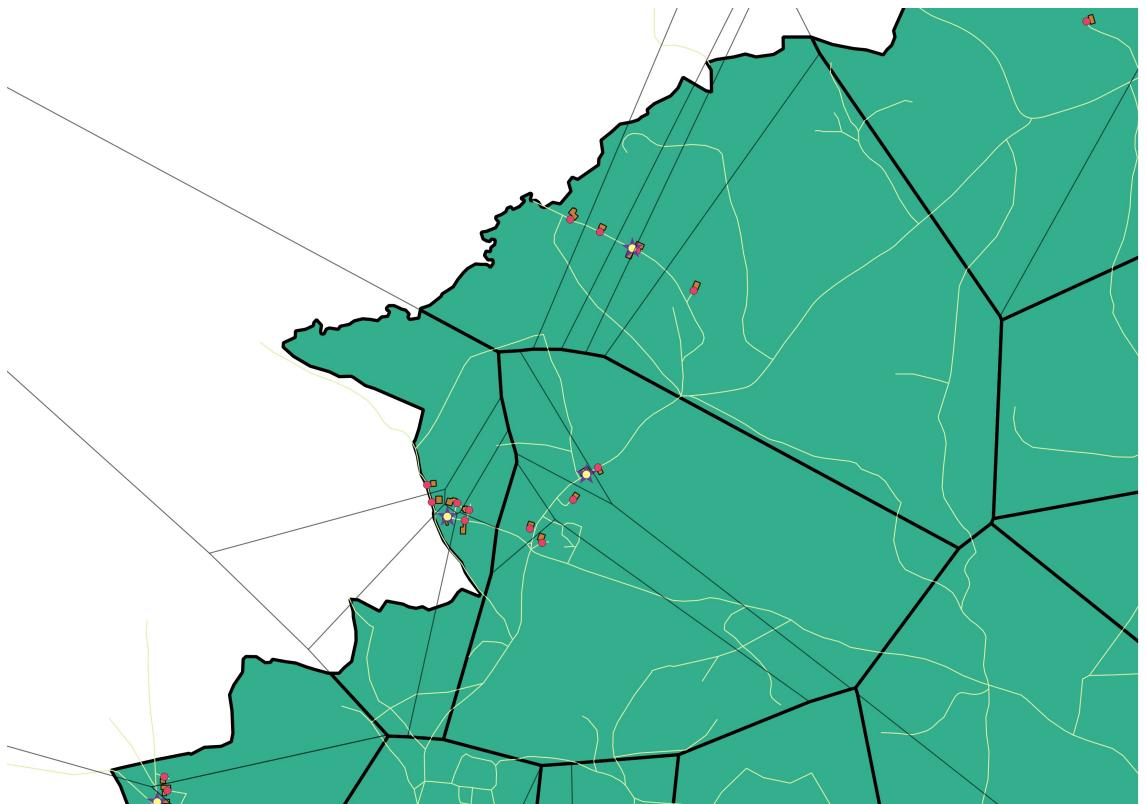


Abb. 2: Einteilung der Basisgebiete in Territorien (Ausschnitt)

Abbildung 2 auf Seite 7 zeigt die Einteilung von Basisgebieten zu Territorien. Dünn angedeutet sind die Basisgebiete zu erkennen, welchen einzelne Häusern zugeordnet sind. Die Territorien mit ihren Zentren (violett-gelber Stern) sind durch die dicken schwarzen Striche gekennzeichnet. In diesem Ausschnitt sind auch die Gemeindegrenzen zu erkennen, welche die Territorien zusätzlich begrenzen können.

## 2.2 Similarity-Partitioning

Ein Beispiel für ein Verfahren, welches Gebäude in der Ebene – also ohne die Berücksichtigung des Straßennetzes – zusammenfasst, ist das „Similarity-Partitioning“. Güting präsentiert in [34, S. 41 f.] einen einfachen Algorithmus, welcher eine Anzahl von Gebäuden mit Hilfe einer Kreisscheibe in der Ebene zusammenfasst. Dieser lässt sich wie folgt darstellen:

Zunächst wird eine bestimmte Zahl  $k$  festgelegt, welche angibt, wie viele Gebäude in einer Kreisscheibe zusammengefasst werden sollen. In der Abbildung siehe Abb. 3 auf Seite 9 wurde  $k = 5$  gewählt. Anschließend wird für jedes Gebäude der Abstand zu seinen  $k - 1$  nächsten Gebäuden berechnet. Der maximale Abstand ergibt den Radius  $r$  einer Kreisscheibe mit der Koordinate des gewählten Gebäudes  $c_i$  als Zentrum. Dadurch entstehen  $|T^*| = n$  Kreisscheiben. Aus der Menge an Kreisscheiben  $T^*$  werden der Reihe nach zufällig einzelne Kreisscheiben in der Ebene platziert. Eine neu hinzugefügte Kreisscheibe darf sich nicht mit einer bereits platzierten Kreisscheibe schneiden; sollte sie sich schneiden, dann wird sie verworfen. Als Ergebnis erhält man eine Auswahl der Kreisscheiben  $T \subset T^*$ , welche in einem Bereich mit einer hohen Gebäudedichte einen kleineren Radius aufweisen und umgekehrt in einem Bereich mit einer geringen Gebäudedichte einen größeren Radius.

Auf eine Darstellung des Algorithmus in Pseudocode wird an dieser Stelle verzichtet. Er wird jedoch in einem späteren Kapitel für die Vorstellung der PostGIS-Erweiterung als PostgreSQL-Implementierung im Detail präsentiert, siehe Abschnitt 4.1 auf Seite 18. Angewendet auf den Testdatensatz „Neukirchen“ (siehe Abschnitt 4.6.6 auf Seite 35) erzeugt das Similarity-Partitioning folgendes Ergebnis.

Offensichtlich werden mit diesem Verfahren viele, aber nicht alle Gebäude abgedeckt. Da jede Kreisscheibe  $k$  Gebäude abdeckt, muss die Anzahl der platzierbaren Kreisscheiben  $|T| \leq |T^*|/k$  sein.

Durch die zufällige Auswahl von Kreisscheiben gibt es auch nicht eine einzige Lösung, sondern viele verschiedene Lösungsmöglichkeiten. Würde zum Beispiel als Kostenfunktion die Anzahl der abgedeckten Gebäude gewählt, dann ist davon auszugehen, dass es bessere und schlechtere Lösungen bei der Auswahl der Kreisscheiben

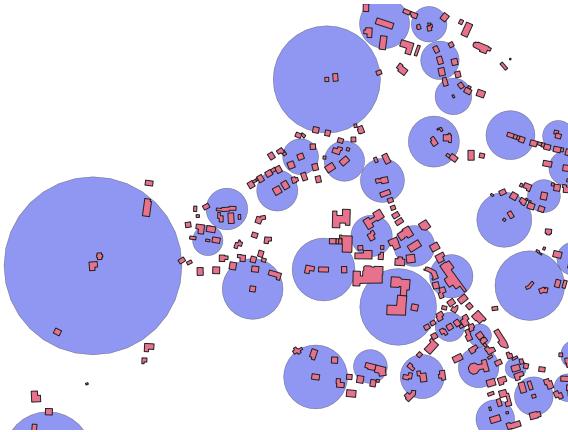


Abb. 3: Similarity-Partitioning nach Gütting (Ausschnitt)

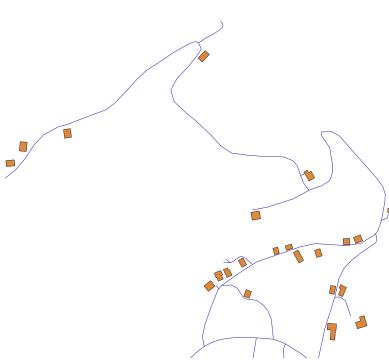


Abb. 4: Straße mäandert am Berg

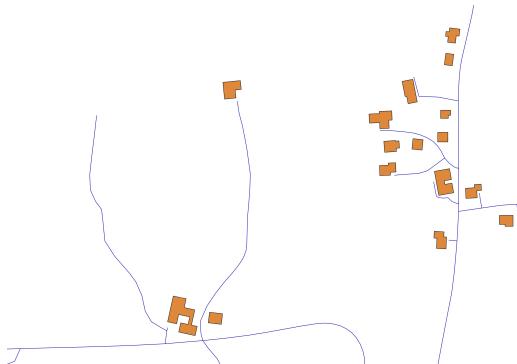


Abb. 5: Haus im Wald

$T \subset T^*$  gilt. Oder anders betrachtet: Die Anzahl der vorhandenen Kreisscheiben  $|T^*|$  ist mindestens  $k$ -mal größer als die Anzahl der für eine Lösung verwendeten  $|T|$ .

Da der gegebene Anwendungsfall Daten produziert, welche durch den Betrieb eines Bedarfsverkehrssystems entstehen, liegt es nahe, für die verwendeten Cluster auch das Straßennetz zu berücksichtigen. Spontan fallen zwei Szenarien ein, welche ungünstige Lösungen produzieren würden, wenn die Straßenentfernung nicht berücksichtigt würde.

Beispiel 1: Ein Gebäude befindet sich auf einem Hügel. Die Straße verläuft in Serpentinen den Hügel hinauf. Obwohl das Gebäude sich auf dem Hügel befindet, besteht eine kurze Luftlinie zu einem weiteren Gebäude im Tal (siehe Abb. 4 auf Seite 9).

Beispiel 2: Ein Gebäude befindet sich abgeschieden im Wald. Seine Nachbarn sind über die Zufahrtsstraße zu erreichen. In der Luftlinie jedoch liegen andere Gebäude näher (siehe Abb. 5 auf Seite 9).

Damit die Straßenentfernung berücksichtigt werden kann, muss das Problem der Aggregation von Geoobjekten das Straßennetz in Form eines Straßengraphen mit einbeziehen. Während das Similarity-Partitioning die Ebene einteilt, wird für die GOA eine entsprechende Partitionierung des Straßengraphen benötigt. Es liegt damit auch ein Problem aus dem weiten Feld der Graph-Partitionierung vor.

## 2.3 Einbindung des Straßennetzes

Um Geoobjekte im Straßennetz aggregieren zu können, muss zunächst der Bezug vom Objekt zum Straßenverlauf hergestellt werden. Im Kontext einer Geodatenbank werden Geoobjekte häufig als Polygone und das Straßennetz wird typischerweise als eine Menge von Polygonzügen definiert. Die Zuordnung erfolgt als eine Projektion des Schwerpunkts des Geoobjektes auf den nächstgelegenen Punkt im Straßennetz.

### Definition 8 (Straßenzug)

*Ein Straßenzug ist eine Vereinigung von Strecken  $SZ = [p_1, p_2] \cup [p_2, p_3] \cup \dots \cup [p_{m-1}, p_m]$ .*

### Definition 9 (Straßennetz)

*Das Straßennetz ist die Vereinigung aller  $n$  Straßenzüge  $SN = \bigcup_{i=1}^n SZ_i$ .*

### Definition 10 (Geoobjekt-Straßennetz-Projektion)

*Die Geoobjekt-Straßennetz-Projektion bestimmt zunächst für das Polygon des Geoobjektes dessen geometrischen Schwerpunkt  $s$ . Es wird ein Punkt  $p \in SN$  gesucht, sodass der Abstand  $d(s, p)$  minimal ist.*

Die Projektionen der Geoobjekte gemeinsam mit den Kreuzungen der Straße bilden die Knoten für den Geoobjekte-Straßendistanz-Graphen, der wie folgt definiert wird:

### Definition 11 (Geoobjekte-Straßendistanz-Graph)

*Ein Geoobjekte-Straßendistanz-Graph (kurz GSD-Graph) ist ein ungerichteter Graph  $G(V, E, w_i, d_j)$  bestehend aus einer Knotenmenge  $V$ , einer Kantenmenge  $E$ , dem Knotengewicht  $w_i \in \{0, 1\}$  und der Kantendistanz  $d_j \in \mathbb{R}_0^+$ . Die Knotenmenge  $V = V_g \cup V_k$  enthält die Menge der auf die Straße projizierten Geoobjekte  $V_g$  sowie die Menge der Straßenkreuzungen  $V_k$ . Für alle Knoten  $v_{gi} \in V_g$  gilt, dass ihr Knotengewicht  $w_{gi} = 1$  ist. Für alle Knoten  $v_{ki} \in V_k$  gilt, dass ihr Knotengewicht  $w_{ki} = 0$  ist. Die Kantenmenge  $E$  besteht aus den Straßenabschnitten zwischen den Knoten. Für jede Kante  $e_j \in E$  existiert eine Kantendistanz  $d_j$ , welche der Straßen-distanz zwischen den jeweiligen Knoten entspricht.*

Es ist festzuhalten, dass es eine bewusste Entscheidung ist, hier nur mit einem ungerichteten Graphen zu arbeiten. Grundsätzlich wäre es denkbar, auch Einbahnstraßen mit Hilfe eines gerichteten Graphen zu modellieren. Allerdings würde das einige Schwierigkeiten mit sich bringen, da es sich bei der GOA im Endeffekt um eine Gebietseinteilung der Ebene handelt, für die das Konzept von „Richtung“ nicht definiert ist. Aus einer praktischen Überlegung heraus kann auch argumentiert werden, dass der in der Motivation skizzierte Anwendungsfall von ländlichen Bedarfsverkehrssystemen selten mit Einbahnstraßen konfrontiert ist.

Bei der Geoobjekt-Straßennetz-Projektion kann es vorkommen, dass mehrere Objekte auf denselben Punkt im Straßennetz projiziert werden. Das ist meistens dann der Fall, wenn der Anfangs- oder Endpunkt eines Straßenzuges den nächstgelegenen Punkt darstellt. In der Abbildung (siehe Abb. 6 auf Seite 11) werden die Schwerpunkte von 3 Gebäuden auf den gleichen Punkt im Straßennetz projiziert. Um diesen Fall im GSD-Graphen abbilden zu können, ist es erlaubt, dass die Kantendistanz  $d_j = 0$  ist. Alternativ dazu besteht die Möglichkeit,  $n$  Knoten mit dem Gewicht  $w = 1$  zu einem Knoten mit dem Gewicht  $w = n$  zusammenzufassen. Aus Gründen der einfachen Implementierbarkeit wurde diese Alternative allerdings verworfen, da im Falle von Knotengewichten  $w > 1$  keine einfache bijektive Zuordnung der Knoten zu den Objekten möglich wäre.

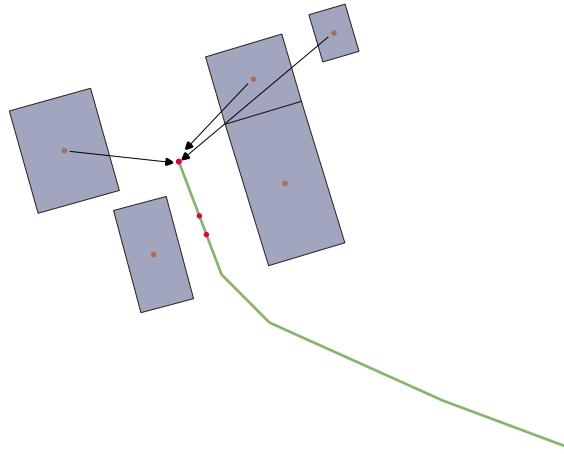


Abb. 6: Häuser werden demselben Punkt auf der Straße zugeordnet

Eine Beobachtung ist, dass der Grad aller den Geoobjekten zugeordneten Knoten  $\deg(v) \leq 2, v \in V_g$  ist. Mit anderen Worten sind die Knoten  $v \in V_g$  keine Kreuzungen im GSD-Graphen. Für den Fall, dass eine Projektion direkt auf eine Kreuzung  $k \in V_k$  im Straßennetz trifft, gilt, dass  $d(v, k) = 0$  ist, jedoch der Knoten des Objektes und der Knoten der Kreuzung zwei verschiedene Knoten sind. Neben „echten Kreuzungsknoten“ mit einem Grad  $\deg(k) \geq 3$  sind auch „unechte Kreuzungskno-

ten“ mit einem Grad  $\deg(k) = 2$  zulässig. Diese Knoten können dann entstehen, wenn sich in den Ausgangsdaten zwei Straßenzüge hintereinander berühren, ohne eine Kreuzung zu bilden.

Anstelle des GSD-Graphen ist es auch denkbar, mit einem Graphen zu arbeiten, in dem ausgehend von den Kreuzungsknoten paarweise Kanten, welche zu den Nachbarknoten führen, durch eine die Nachbarknoten direkt verbindende Kante ersetzt werden. Dadurch könnten alle Kreuzungsknoten mit dem Gewicht 0 eliminiert werden. Das hätte den Vorteil, dass der Graph nur mehr Kantengewichte und keine Knotengewichte mehr besitzen würde. Diese Idee ist interessant und verdient genauere Betrachtung, weil sich dadurch möglicherweise eine andere Perspektive zur Umsetzung der GOA ergeben können (siehe Abschnitt 9.1.2 auf Seite 69). Da allerdings die für die Implementierung der GOA verwendeten Werkzeuge mit kantengewichteten Knoten umgehen können, wird die Definition des GSD-Graphen im Rahmen dieser Arbeit so wie vorgestellt belassen.

## 2.4 Benachbarte Probleme

Die GOA ist mit mindestens zwei viel beforschten Problemfeldern verwandt. Der Aspekt des Findens von Zentren stellt die Beziehung zur Standortplanung und zu Verfahren wie  $k$ -center,  $k$ -median und auch  $k$ -facilities in den Vordergrund. Die Idee, Geoobjekte im Straßengraphen zu clustern, macht ein Näheverhältnis zum Problem der Graph-Partitionierung offensichtlich.

### 2.4.1 $k$ -center, $k$ -median, $k$ -facilities

Das  $k$ -center-Problem besteht aus einem vollständigen Graphen  $G = (V, E)$  mit Kantengewichten  $w_e > 0, e \in E$  und  $w_{(v,v)} = 0, v \in V$ . Die Aufgabe besteht darin, zentrale Knoten  $S \subset V$ , unter der Bedingung  $|S| \leq k$ , zu finden, sodass das maximale Gewicht der Kanten, welche von den Zentren zu den jeweils zugeordneten Knoten führen minimal ist:  $w(S) = \max_{i \in V} \min_{j \in S} w_{(i,j)}$ . Sofern für den Graphen  $G$  die Dreiecksungleichung  $w_{(i,k)} + w_{(k,j)} \geq w_{(i,j)}, i, j, k \in V$  gilt, wurde nachgewiesen, dass dieses Problem NP-vollständig ist und dass jede  $\delta$ -Approximation für  $\delta < 2$  NP-schwer ist [37, S. 1]. Algorithmen, welche eine 2-Approximation erlauben, werden in [37, 31, 72] vorgestellt.

Das  $k$ -center-Problem unterscheidet sich in einigen Punkten von der GOA. Während das  $k$ -center-Problem die Forderung  $|S| \leq k$  mit sich bringt, ist die Anzahl der Zentren für die GOA nachrangig, solange die  $a$ -Anonymität – es müssen mindestens  $a$  Knoten einem Zentrum zugeordnet sein – eingehalten wird. Weiters wird als

Zielfunktion die Minimierung des maximalen Kantengewichts bzw. Abstandes der Knoten zu ihren Zentren gewählt. Im Falle der GOA wird gewünscht, dass die Summe der Abstände aller Knoten zu ihren Zentren minimiert wird. Diese Zielfunktion  $w(S) = \sum_{i \in V} \min_{j \in S} w_{(i,j)}$  wird in der Literatur unter anderem „minsum“ genannt und findet beim  $k$ -median-Problem seine Anwendung, welches sich dadurch von  $k$ -centers abgrenzt.

Das  $k$ -facilities-Problem fügt eine Bewertung der Kosten hinzu, welche sich durch die Eröffnung eines Standorts ergeben. Wie auch beim  $k$ -median-Problem ist die Anzahl der Standorte/Zentren nach oben mit  $k$  begrenzt.

In [15, S. 2] wird festgestellt, dass die Probleme  $k$ -center,  $k$ -median und  $k$ -facilities grundsätzlich eine ähnliche Aufgabe der Clusterbildung übernehmen, sich jedoch deutlich unterschiedlich im Kontext von Approximationsalgorithmen darstellen.

Alle genannten  $k$ -X-Verfahren existieren auch in einer „soft capacitated“ und einer „hard capacitated“ Variante. „Capacitated“ bedeutet im Fall von  $k$ -centers und  $k$ -median, dass jedem Zentrum nur bis zu einer Kapazitätsgrenze  $L$  Knoten zugeordnet werden dürfen.

Verallgemeinert kann ein „capacitated“  $k$ -clustering, wie folgt definiert werden [3, S. 1]: „Given an integer  $L > 0$ , a  $L$ -capacitated  $k$ -clustering is a  $k$ -clustering in which there are at most  $L$  points in each cluster. The capacitated  $k$ -center problem is to compute a  $k$ -clustering of the smallest size so that each cluster has at most  $L$  points.“

Das „soft capacitated“ Verfahren erlaubt mehrere Einrichtungen pro Standort. Bei der „hard capacitated“ Variante ist nur maximal eine Einrichtung pro Standort zulässig [15, S. 1].

Während die „capacitated“ Eigenschaft eine obere Schranke definiert, da einem Zentrum eine maximale Anzahl an Knoten bzw. Nachfrage zugewiesen werden darf, erfordert die GOA mit ihrer  $a$ -Anonymität eine untere Schranke, denn es müssen einem Zentrum mindestens  $a$  Knoten zugeordnet werden. In der Literatur finden sich viele Beispiele für die Beschäftigung mit den capacitated  $k$ -X-Problemen, neben [49, 85, 2, 27, 1] auch noch viele andere. Die Literaturrecherche zu einem  $k$ -X-Problem mit einer unteren Schranke der zugeordneten Knoten im Sinne der  $a$ -Anonymität führte zu keinem Ergebnis. Es ist jedoch möglich, dass noch nicht mit dem erforderlichen Vokabular an Suchbegriffen recherchiert worden ist.

Interessant im Zusammenhang mit der GOA sind die Arbeiten von [19] und [16]. Beide stellen Approximationsalgorithmen für das capacitated  $k$ -median-Problem vor, welche eine Verletzung der Kapazitätsbedingung erlauben. Da im Falle der

GOA über die  $a$ -Anonymität nur eine untere Schranke für die Kapazität definiert ist, wäre eine Verletzung der Kapazitätsbedingung nach oben hin nicht hinderlich. Die Frage, die sich stellt, ist, ob solche Algorithmen bei einer geschickten Wahl von  $k$  und den hier verwendeten Ausgangsdaten brauchbare  $a$ -anonyme Ergebnisse liefern können. Leider übersteigen die theoretischen Grundlagen dieser Arbeiten den Wissensstand des Autors dieser Arbeit deutlich, was ein schnelles Ausprobieren dieser Verfahren unmöglich macht. Vielleicht können jedoch einige Ideen in diesen Arbeiten als Inspirationsquellen dienen.

#### 2.4.2 Graph-Partitionierung, Graph-Clustern

Im Themenbereich der Graph-Partitionierung wird viel Forschung betrieben, da die möglichen Anwendungen mannigfaltig sind. Graph-Partitionierungsalgorithmen kommen zum Beispiel im Schaltungsdesign (VLSI), in Computernetzwerken, bei der Datenanalyse, im Bereich des maschinellen Lernens, in der Bioinformatik sowie in vielen weiteren Bereichen zum Einsatz. Je nach Anwendungsfall werden verschiedene Anforderungen an die Graph-Partitionierung gestellt. Die „Balanciertheit“ und die Minimierung der notwendigen Schnitte sind oft geforderte Eigenschaften.

Eine einfache Definition des Graph-Partitionierungsproblems findet sich in [24, S. 2], sie wird nun sinngemäß wiedergegeben.

#### Definition 12 (Graph-Partitionierung)

*Das Graph-Partitionierungsproblem besteht aus einem Graphen  $G = (V, E)$ .  $V$  entspricht einer Menge von gewichteten Knoten und  $E$  einer Menge von gewichteten Kanten. Die Knoten werden in  $p \in \mathbb{Z}^+$  Untermengen  $V_1, V_2, \dots, V_p$  von  $V$  aufgeteilt, sodass gilt:*

1. *Disjunktheit:  $\cup_{i=1}^p V_i = V$  und  $V_i \cap V_j = \emptyset, \forall i \neq j$*
2. *Balanciertheit:  $W(i) \approx W/p, i = 1, 2, \dots, p$ , wobei gilt:  $W(i)$  und  $W$  sind die Summen der Knotengewichte der Knoten in  $V_i$  bzw.  $V$  gesamt.*
3. *Minimierung der Schnitte (engl. „cut size“): Die Summe der Kantengewichte, welche durch die Partitionierung geschnitten werden, soll minimiert werden.*

Wie bereits erwähnt ist das Forschungsgebiet der Graph-Partitionierung groß, dadurch ist es für eine außenstehende Person zunächst schwierig, sich einen Überblick zu verschaffen. Als eine wertvolle Einstiegshilfe können sogenannte „Survey“- oder „Overview“-Papers dienen. Die AutorInnen dieser Arbeiten haben ausführliche

Recherchearbeit geleistet und versuchen je nach Intention entweder einen allgemeinen Überblick oder einen Überblick zum Stand der Technik zu liefern. Weiteres drückt sich oft durch das Wort „recent“ oder „advances“ im Titel der Arbeit aus, z. B.: „Recent Advances in Graph Partitioning“ [14]. Besonders erwähnenswert für einen allgemeinen Überblick sind die Arbeiten „An Overview of Graph Covering and Partitioning“ [67], „Algorithms for Graph Partitioning: A Survey“ [24] und „Graph clustering“ [66]. Diese Arbeiten waren für die vorliegende Arbeit treue Begleiter, die gleichsam wie Landkarten immer dann zur Hilfe genommen wurden, wenn Gefahr bestand, die Orientierung im Dickicht der massiven Forschungsleistungen im Bereich der Graph-Partitionierung zu verlieren.

Der Unterschied zwischen der Graph-Partitionierung und dem Graph-Clustering ist auf den ersten Blick schwer erkennbar, oftmals sind die Verfahren sehr ähnlich. Allerdings ist die Intention eine andere. Während z. B. bei einer balancierten Graph-Partitionierung versucht wird, die Daten in ähnlich große Blöcke zu teilen, ist die Intention des Graph-Clustering abweichend. Hier wird versucht, diejenigen Cluster zu finden, deren Datenpunkte eine gewisse Ähnlichkeit aufweisen. „Any nonuniform data contains underlying structure due to the heterogeneity of the data. The process of identifying this structure in terms of grouping the data elements is called clustering“ [66, S. 27]. Diese Erkenntnis führt zur Verlegenheit, dass möglicherweise der Titel dieser Arbeit nicht ganz treffend gewählt wurde, denn im Wesentlichen werden hier gleichwertige Geoobjekte aggregiert, was tendenziell in die Domäne der Partitionierung fallen würde. Nichtsdestoweniger ist es das Ziel, Cluster von Geoobjekten zu bilden. Welche Eigenschaften ein Cluster besitzen sollte, wird in [66, S. 33] erläutert und nun übersetzt wiedergegeben. Im Kontext von Graphen sollte jedes Cluster intuitiv verbunden sein. Es sollte mindestens einen, besser noch mehrere Pfade geben, welche Knoten paarweise innerhalb des Clusters verbinden. Wenn ein Knoten  $u$  von einem Knoten  $v$  nicht erreicht werden kann, dann sollen  $u, v$  nicht demselben Cluster angehören. Weiters sollen diese Pfade innerhalb des Clusters verlaufen. Es reicht nicht, wenn die Knoten des induzierten Teilgraphen  $C$  im gesamten Graphen  $G$  verbunden sind, sondern sie sollten innerhalb des Teilgraphen selbst verbunden sein. Es genügt weiters nicht, wenn zwei Knoten  $u$  und  $v$  über einen Pfad verbunden sind, der durch einen Knoten von  $V \setminus C$  führt, sondern der gesamte Pfad darf nur durch Knoten aus  $C$  führen. Betrachtet mensch diese Anforderungen an Verbundenheit und Dichte, dann ist die schwächste Definition eines Clusters die einer Zusammenhangskomponente und die stärkste Definition die einer Maximal Clique (ein Subgraph, wo jeder Knoten mit jedem anderen paarweise verbunden ist). „In most occasions, the semantically useful clusters lie somewhere in between these two

extremes“ [66, S. 33]. Es wird sich später zeigen, inwieweit die Clustereigenschaften bei dem hier vorgestellten GOA-Verfahren eingehalten werden.

Eine spezielle Form der Graph-Partitionierung, genannt  $(l, u)$ -Partitionierung, sollte im Kontext der GOA erwähnt werden. Hierbei handelt es sich um ein Verfahren, das Partitionen erstellt, in denen die Summe der Knotengewichte nach unten hin mit  $l$  und nach oben hin mit  $u$  beschränkt ist. Diese Problemstellung ist insofern interessant, als sie mit einer unteren Schranke  $l$  arbeitet und die GOA ebenfalls eine untere Schranke für die  $a$ -Anonymität benötigt. „Given a graph with nonnegative vertex weights and numbers  $0 \leq l \leq u$ , a partition into connected components, each of which has a cumulated weight in the range  $[l, u]$ , is called an  $(l, u)$ -partition. Typical objectives are to maximize or to minimize the number of components of the  $(l, u)$ -partition“ [67, S. 8]. Algorithmen für dieses Problem finden sich in [40, 18].

## 3 Datenquellen

Als Ausgangsdaten stehen uns mehrere Datenquellen zur Verfügung. Ein prominentes Beispiel ist OpenStreetMap, welches weltweit verfügbar ist. Für Österreich werden mit basemap.at Geodaten von der öffentlichen Hand als Datenquelle zur Verfügung gestellt.

### 3.1 OpenStreetmap

OpenStreetMap (kurz OSM) ist eine crowdsourced Geodatenbank, welche angelehnt an das Konzept von Wikipedia eine einfache Editiermöglichkeit und eine volle Revisionskontrolle bereitstellt. Die BenutzerInnen können geographische Einheiten bearbeiten, welche in Form von folgenden drei Basistypen abgebildet werden [70, S. 2]:

1. Knoten (engl. nodes) – Diese repräsentieren Punkte mit Koordinaten mit Längen- und Breitengraden im WGS 84 System.
2. Wege (engl. ways) – Ähnlich einem Streckenzug werden diese durch eine Liste von Referenzen auf Knoten definiert.
3. Relationen (engl. relations) – Diese stellen Zusammenhänge zwischen Knoten, Wegen und weiteren Relationen her. Dadurch können komplexe Objekte, wie z. B. Multipolygone, modelliert werden.

Jede dieser Einheiten ist mit einer eindeutigen Referenz („OSM ID“) und mit einer Menge an Key-Value-Eigenschaften (genannt tags) versehen. Ein Feldweg wird zum Beispiel als ein Weg mit dem Attribut `highway='track'` gespeichert.

Die Daten von OpenStreetMap werden großteils von Freiwilligen erfasst. Der Begriff „Volunteered Geographic Information (kurz VGI)“ beschreibt das Sammeln und Verbreiten von geographischen Daten und den dazugehörigen Werkzeugen auf freiwilliger und lizenziertlich offener Basis [70, S. 3]. Dieser Begriff wurde in [32] erstmalig präsentiert.

Diese Offenheit macht OSM zu einer guten Wahl für Forschungsanliegen, weil erstens sämtliche Daten öffentlich verfügbar sind, es zweitens eine Vielzahl an frei verfügbaren Open-Source-Tools gibt und drittens eine breite Community existiert, welche einen regen Austausch betreibt. Dadurch lassen sich viele Probleme durch eine einfache Internetrecherche schnell lösen. Für die vorliegende Arbeit wird ebenfalls auf Daten und Werkzeuge aus dem OSM-Ökosystem zurückgegriffen.

### 3.2 Basemap.at

Basemap.at ist eine Geodatenbank der amtlichen österreichischen Verwaltungen, welche „auf den Verwaltungs-Geodaten der neun Bundesländer, der Graphen-integrations-Plattform (GIP.at), [sic] sowie der Länderpartner, allen voran den Städten und Gemeinden [sic] basiert“ [10]. Als ein „Open-Government-Data-Produkt“ ist es frei öffentlich zugänglich.

Folgende Eigenschaften zeichnen Basemap.at aus [9]:

1. Geprüfte Qualität – wird von österreichischen Verwaltungen täglich verwendet und gewartet.
2. Aktualität – für Verwaltungsaufgaben müssen die Geodaten immer auf aktuellem Stand sein.
3. Homogenität – unabhängig vom Standort weisen alle Geodaten österreichweit denselben Detailgrad auf.

Gerade für den Anwendungsfall Bedarfsverkehrssysteme ist es interessant, mit öffentlichen Daten zu arbeiten.

Das in dieser Arbeit vorgestellte Verfahren greift auf eine lokale räumliche Datenbank zurück, welche eine Art Abstraktionsebene zu verschiedenen Datenquellen bietet. Mit überschaubarem Aufwand sollte es möglich sein, Geodaten aus basemap.at in die lokale Datenbank zu importieren und mit diesen, anstelle der OSM-Geodaten, zu arbeiten. Um den Fokus für diese Arbeit nicht zu verlieren, erfolgt zunächst eine Beschränkung auf OpenStreetMap als Datenquelle.

## 4 PostgreSQL und PostGIS

PostgreSQL ist ein freies, objektrelationales Datenbankmanagementsystem, welches durch die Erweiterung PostGIS zu einem räumlichen Datenbanksystem (oder Geodatenbanksystem / engl. „spatial database system“) erweitert wird. Nach Güting zeichnet sich ein räumliches Datenbanksystem durch folgende Eigenschaften aus [33, S. 1]:

1. Ein räumliches Datenbanksystem ist ein Datenbanksystem.
2. Es besitzt spezielle räumliche Datentypen in seinem Datenmodell und in der Abfragesprache.
3. Es muss für diese räumlichen Datentypen mindestens Implementierungen für räumliche Indexe und räumliche Joins bereitstellen.

Der erste Punkt wird klarerweise alleine schon durch PostgreSQL erfüllt. Die PostGIS-Erweiterung fügt, neben anderen, den Datentyp „geometry“ hinzu. Ein Datum vom Typ „geometry“ kann einen Punkt, einen Streckenzug, ein Polygon, ein Multipolygon und noch weitere Geometrien darstellen [54, S. 33 ff.]. In seiner aktuellen Version stellt PostGIS 3 verschiedene räumliche Indexe zur Verfügung (siehe Abschnitt 4.2 auf Seite 21). Neben den geforderten räumlichen Joins implementiert PostGIS eine Vielzahl an räumlichen Funktionen unter Zuhilfenahme von Open-Source-Softwareprojekten, wie „Proj“ (zur Projektion von verschiedenen Referenzsystemen), „GEOS“ (für Funktionen der algorithmischen Geometrie), „CGAL“ (wie zuvor Genanntes, aber vor allem auch für höherdimensionale Aufgaben), „GDAL“ (für Rasterfunktionen und Rasterfileformate) und einigen weiteren [61].

### 4.1 Similarity-Partitioning-Implementierung

Um einen ersten Eindruck zu vermitteln, was mit PostGIS möglich ist, soll an dieser Stelle eine Implementierung des bereits vorgestellten Similarity-Paritioning-Algorithmus gezeigt werden.

Als Voraussetzung muss zunächst gegeben sein, dass eine Datenbanktabelle mit dem Namen „buildings“ und den Spalten „id“ vom Typ „text“ und „centerpoint“ vom Typ „geometry“ existiert. In dieser Tabelle sind alle Gebäude mit einer eindeutigen ID und ihrem Zentrum als Punkt gespeichert. Idealerweise wurde dieser Tabelle bereits ein räumlicher Index hinzugefügt.

Ziel des Algorithmus ist es, eine Menge an kNN-Kreisscheiben zu identifizieren, sodass die Ebene durch diese kNN-Kreisscheiben eingeteilt wird.

Zunächst wird eine Datenbanktabelle mit dem Namen „discs“ definiert, welche alle möglichen Kreisscheiben mit der zugehörigen Gebäude-ID, deren Radius, der Kreisscheibe dargestellt als Polygon in der Ebene sowie einem booleschen Wert „chooseen“ enthält. Das Zwischenziel ist es, in diese Tabelle alle möglichen Kreisscheiben einzufügen und dann in einem nachgelagerten Schritt diejenigen Kreisscheiben auszuwählen, welche die Ebene einteilen sollen.

```
1 CREATE TABLE discs(id text, r float, circle geometry, chooseen
boolean, PRIMARY KEY(id));
```

Die Erstellung aller Kreisscheiben erfolgt mit folgendem Ausdruck. Die Schritte werden im Anschluss im Einzelnen erklärt.

```
1 INSERT INTO discs SELECT id,MAX(r) as r,ST_Buffer(centerpoint,MAX(r
)),FALSE as circle from (SELECT b.id AS id, b.centerpoint AS
centerpoint,ST_Distance(b.centerpoint, knn.centerpoint) AS r
FROM buildings AS b CROSS JOIN LATERAL (SELECT centerpoint FROM
buildings ORDER BY centerpoint <-> b.centerpoint LIMIT 5) AS knn
) AS discs GROUP BY id,centerpoint;
```

Um diesen komplexen Ausdruck zu verstehen, wird er von innen nach außen zerlegt und die einzelnen Operationen werden vorgestellt.

```
1 SELECT centerpoint FROM buildings ORDER BY centerpoint <-> b.
centerpoint LIMIT 5
```

Dieser Ausdruck sucht für den b.centerpoint die fünf nächsten Nachbarn (inkl. dem Punkt b.centerpoint selbst), der „nearest neighbour Operator“ wird mit „<->“ dargestellt. Informell trägt dieser Operator, auf Grund seiner Form, im Englischen den Namen „Tiefighter-Operator“. Das Schlüsselwort „LIMIT“ gefolgt von einer Ganzzahl definiert, wie viele nächste Nachbarn gesucht werden sollen.

Mit Hilfe von „CROSS JOIN“ wird das kartesische Produkt der Datensätze zweier Tabellen erzeugt, das bedeutet, dass jede Reihe eines Datensatzes mit jeder Reihe des anderen Datensatzes verknüpft wird. In dem vorliegenden Fall wird der Datensatz der Tabelle „buildings“ mit dem SELECT-Statement zum Auffinden der fünf nächsten Nachbarn verknüpft. Das Schlüsselwort „LATERAL“ ist notwendig, damit das SELECT-Statement auf das Datum b.centerpoint zugreifen kann, welches dem anderen Operanden des „CROSS JOIN“-Aufrufs entspringt.

Der folgende kombinierte Ausdruck liefert als Zeilen für jeden b.centerpoint den Abstand seiner fünf nächsten Nachbarn. Der Abstand wird mit der PostGIS-Funktion „ST\_Distance()“ berechnet.

```

1 (SELECT b.id AS id, b.centerpoint AS centerpoint, ST_Distance(b.
    centerpoint, knn.centerpoint) AS r FROM buildings AS b CROSS
    JOIN LATERAL (SELECT centerpoint FROM buildings ORDER BY
    centerpoint <-> b.centerpoint LIMIT 5) ass knn

```

Mit Hilfe von „GROUP BY“ werden diese Zeilen anhand ihrer ID und des centerpoint gruppiert. Die Aggregatfunktion „MAX()“ filtert die Zeile mit dem größten Abstand heraus.

Die PostGIS-Funktion „ST\_Buffer()“ nimmt eine Geometrie  $g$  und eine Zahl  $r$  als Parameter entgegen. Diese Geometrie wird radial um  $r$  Einheiten expandiert. In dem konkreten Anwendungsfall entsteht aus einem Punkt eine Kreisscheibe mit dem Radius  $r$ . Anm.: Es handelt sich dabei um keine exakte Kreisscheibe, sondern um eine Näherung, welche ein Polygon verwendet.

Bevor die Daten weiterverarbeitet werden, wird ein räumlicher Index erstellt. Dieser Schritt dient dazu, dass die PostGIS-internen Funktionen wie zum Beispiel „ST\_Intersects()“ effizient arbeiten können. Wie räumliche Indexe funktionieren, wird im Kapitel siehe Abschnitt 4.2 auf Seite 21 erklärt.

```

1 CREATE INDEX c_idx ON discs USING GIST(circle);

```

Damit ist der erste Zwischenschritt, die Erstellung aller möglichen Kreisscheiben, abgeschlossen. Nun gilt es aus der Menge an Kreisscheiben einzelne nach und nach in der Ebene zu platzieren und weitere hinzuzufügen, solange diese sich nicht mit bereits platzierten Kreisscheiben schneiden. Dieser Algorithmus wird nun in der „SQL Procedural Language PL/pgSQL“ implementiert, welche Bestandteil von PostgreSQL ist [63].

```

1 DO $$
2 DECLARE
3     disc RECORD;
4     n_overlaps INTEGER;
5 BEGIN
6     FOR disc IN SELECT * FROM discs
7     LOOP
8         SELECT COUNT(id) INTO n_overlaps FROM discs WHERE ST_Intersects
9             (circle, disc.circle) AND choosen;
10        IF n_overlaps = 0 THEN
11            UPDATE discs SET choosen = TRUE WHERE id = disc.id;
12        END IF;
13    END LOOP;
14 END$$;

```

Dieses einfache Programm iteriert über alle Kreisscheiben und berechnet, mit wie vielen bereits ausgewählten Kreisscheiben sich die aktuelle Kreisscheibe schneidet

(`ST_Intersects()`). Schneidet sich die Kreisscheibe mit keiner anderen ausgewählten, dann wird auch sie zu der Menge an ausgewählten Kreisscheiben hinzugefügt. Auf eine zufällige Auswahl der Kreisscheiben wurde in dieser Implementierung verzichtet.

## 4.2 Räumliche Indexe

Ein räumlicher Index ist eine der drei Anforderungen an ein räumliches Datenbanksystem. Ein räumlicher Index ermöglicht, dass auch Datenbanken mit einer großen Anzahl von räumlichen Daten effizient durchsucht werden können. Viele PostGIS-Funktionen wie zum Beispiel „`ST_Intersects()`“ oder auch der Nearest-Neighbour-Operator greifen, falls vorhanden, auf einen räumlichen Index zurück, um die Anfrage zu beschleunigen. Aktuelle Versionen von PostgreSQL/PostGIS unterstützen 3 verschiedene Arten von räumlichen Indexen. Der Index „Generalized Search Tree“ (kurz GIST) [36] versteht sich als eine vereinheitlichte und erweiterbare Datenstruktur, die unter anderem R-Trees [35], B+-Trees oder RD-Trees abbilden kann. Der „Spatial Partitioned Generalized Search Tree“ (kurz SP-GIST) [6] vereinheitlicht Datenstrukturen, die den Raum partitionieren, das sind zum Beispiel Quadtrees [23],  $k$ -D trees oder Präfixbäume. Der „Block Range Index“ (kurz BRIN) [84, S. 3] ist ein einfacher Index, der eine Liste von Begrenzungsrahmen für eine Anzahl von Einträgen (mit einer typischen Blockgröße) erstellt. Je nach Anwendungsfall weisen die verschiedenen Indexe Vor- und Nachteile auf. Der BRIN-Index ist schnell zu erstellen und benötigt wenig Speicherplatz, er erfordert aber, dass die Daten bereits in ihrer Reihenfolge den räumlichen Bezug enthalten, um vorteilhaft eingesetzt werden zu können. Als Nachteil ist zu sehen, dass Anfragen mit einem räumlichen Bezug nur eine moderate Suchbeschleunigung erfahren, da die gesamte Liste des Index durchgearbeitet werden muss. Die Kosten in Bezug auf Rechenzeit und Speicherplatz bei GIST und SP-GIST liegen in etwa in derselben Größenordnung, aber je nach Daten und Anwendungsfall kann sich ein Vorteil für den einen oder anderen Index ergeben. Datenstrukturen wie R-Trees arbeiten im Gegensatz zu, zum Beispiel, Quadtrees mit überlappenden Bereichen. Daher ist intuitiv anzunehmen, dass der GIST-Index sich performanter verhält, wenn sich die Elemente des Datensatzes stark überlappen. Aber in der Echtanwendung muss oftmals der Praxistest entscheiden, welche Indexvariante die am besten geeignete ist. Bei der Auswahl gilt es zwischen Erstell-dauer, Speicherplatz, Anfrageselektivität und weiteren Parametern abzuwählen. Im Kontext dieser Arbeit wird der Einfachheit halber auf einen Performancevergleich verzichtet und die GIST-Variante für Indexe gewählt. Als Grund steht weniger eine logische Überlegung im Vordergrund als die Tatsache, dass die GIST-Variante eta-

bliert und schon seit Mitte der 90er-Jahre für PostGIS verfügbar ist. Da sich die Geoobjekte in der Ebene  $\mathbb{R}^2$  befinden, arbeitet der GIST-Index in diesem Fall wie ein R-Tree.

### 4.3 GeoHash

Mit Hilfe von GeoHashes können Koordinaten im Längen- und Breitengrad-Referenzsystem in einen String kodiert werden. In der vorliegenden Arbeit werden GeoHashes dafür benutzt, IDs für Geoobjekte zu generieren, dadurch kann ausgehend von der ID auf die Koordinaten des Geoobjektes zurückgerechnet werden.

Die Implementierung des GOA-Verfahrens für diese Arbeit erfolgt in der Programmiersprache Python. Zunächst ist es jedoch erforderlich, dass die Daten aus der räumlichen Datenbank exportiert werden. Ein Punkt in der Datenbank wird dabei vom PostGIS-Datentyp „geometry“ repräsentiert, welcher keine direkte Entsprechung in Python hat. Daher wird der GeoHash dieses Punktes exportiert, welcher nun als ein String zur weiteren Verarbeitung zur Verfügung steht. Zur Erklärung, wie genau ein GeoHash funktioniert, wird nun auszugsweise eine Übersetzung von [8, S. 3] wiedergegeben, welche falls notwendig ergänzt wird. Ein GeoHash ist ein System zur Kodierung von Längen- und Breitengraden in einer kompakten Form, dazu teilt der GeoHash-Algorithmus die geographischen Regionen in einer hierarchischen Art ein. Es werden beginnend mit den höchstwertigen Bits abwechselnd Bits der Längenkoordinate und der Breitenkoordinate verwendet, um die geographische Ebene zu Beginn grob in 32 Regionen einzuteilen, welche durch ein base-32-Zeichen repräsentiert wird. Dieser Vorgang wird nun innerhalb der zuvor gewählten Region so lange wiederholt, bis die gewünschte Genauigkeit erreicht wird. Ein GeoHash repräsentiert genau genommen keinen Punkt, sondern immer einen Begrenzungsrahmen. Als Beispiel: Die Koordinate 45.557, 18.675 wird durch den GeoHash „u2j70vx29gfu“ repräsentiert, da sie in seinen Begrenzungsrahmen fällt. Würde der Vorgang weiter Iterationen durchlaufen, würden dem GeoHash weitere Zeichen hinzugefügt und der Begrenzungsrahmen würde entsprechend eingeengt. Die Einteilung der Ebene erfolgt durch die sogenannte „z-Kurve“ [51]. Eine günstige Eigenschaft von GeoHashes ist, dass nahe Punkte in der Regel durch ähnliche GeoHash-Strings repräsentiert werden. Zwei Punkte sind einander umso näher, je länger das identische Präfix zweier GeoHashes ist.

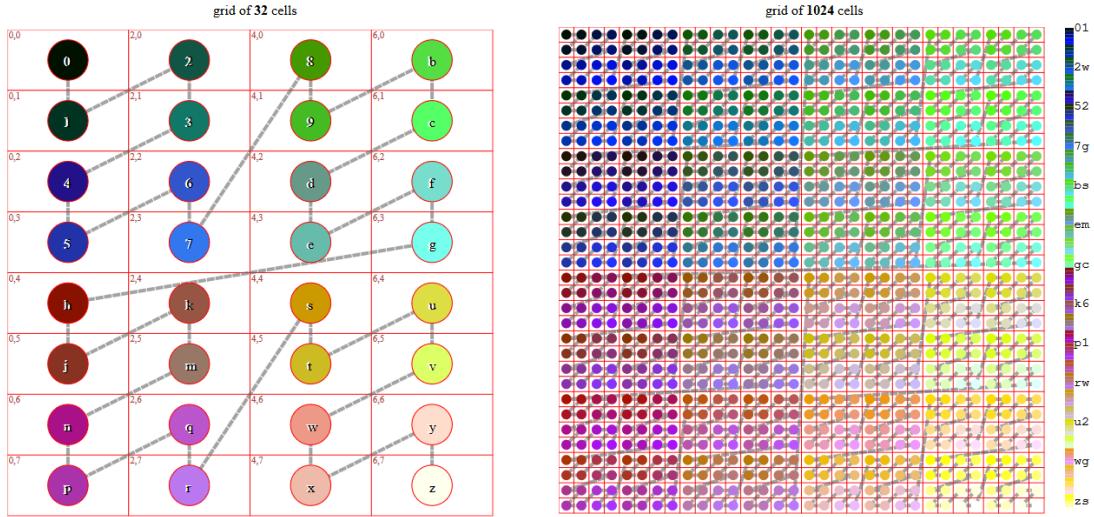


Abb. 7: GeoHash [62]

#### 4.4 Koordinatenreferenzsystem

OSM benutzt standardmäßig das Koordinatenreferenzsystem (kurz CRS) „EPSG: 4326 - WGS 84“. Das heißt, die Koordinaten werden in Längen- und Breitengraden gespeichert.

Das Kürzel eines CRS setzt sich aus folgenden Teilen zusammen [79, 82]:

- „EPSG“ steht für „European Petroleum Survey Group“, eine Interessengruppe, welche ursprünglich damit begonnen hat, die verschiedenen CRS-Systeme zu katalogisieren.
- Die Zahl steht für die Inventarnummer des CRS-Systems und wird auch „Spatial Reference System Identifier (kurz SRID)“ genannt.
- „WGS 84“ steht für den Bezug zum Referenzellipsoid des „World Geodetic System 1984“.

Für die weitere Verarbeitung der Daten wird ein Referenzsystem gewünscht, welches folgende Eigenschaften aufweist:

- Distanzen sollen in Metern angegeben sein.
- Die Berechnung von Distanzen soll effizient erfolgen.
- Das Referenzsystem soll winkeltreu sein, damit Objekte nicht verzerrt dargestellt werden.

Viele Webdienste (wie z. B.: OpenStreetMap oder Google Maps) verwenden das Referenzsystem „EPSG:3857 - WGS 84 / Pseudo-Mercator“, auch genannt „Web Mercator“, welches eine globale Gültigkeit hat [81]. Es hat sich herausgestellt, dass die Längen- bzw. Flächenberechnung in diesem Referenzsystem für die vorliegende Aufgabenstellung zu ungenau ist.

Aus diesem Grund wurde das Referenzsystem „EPSG: 32633 - WGS 84 / UTM zone 33N“ gewählt. Dieses Koordinatenreferenzsystem wird für die nördliche Hemisphäre – daher das „N“ nach der Zonennummer – zwischen dem 12. und 18. östlichen Längengrad vom Äquator bis zum 84. Breitengrad eingesetzt.

Da sich Österreich zwischen  $9^{\circ} 31'$  und  $17^{\circ} 9'$  östlicher Länge ausdehnt, wird damit, bis auf Vorarlberg und Teile Tirols, ein Großteil des Nationalgebietes abgedeckt. Für den Westen Österreichs wäre das CRS „EPSG: 32632 - WGS 84 / UTM zone 32N“ die bessere Wahl, welches für den Einsatz zwischen  $6^{\circ}$  und  $12^{\circ}$  östlicher Länge konzipiert ist.

Das UTM-System teilt die Erde von  $80^{\circ}$  südlicher Breite bis  $84^{\circ}$  nördlicher Breite in 60 Zonen ein, damit ergibt sich eine Einteilung in Zonen von  $6^{\circ}$  des Längengrads [48, S. 48]. UTM steht für „Universal Transverse Mercator“ und beschreibt eine Zylinderprojektion der Erdoberfläche. Transversal bedeutet in diesem Zusammenhang, dass die Achse des Zylinders senkrecht zur Erdachse steht. Der Durchmesser des Zylinders ist so gewählt, dass dieser mit einem Skalierungsfaktor von 0.9996 der Ausdehnung vom Süd- zum Nordpol des Erdellipsoids entspricht. Der Schnitt des Zylinders und des Erdellipsoids ergibt zwei Ellipsen. Für alle Punkte auf diesen beiden Ellipsen gilt, dass die Projektion und die ursprünglichen Punkte ident sind. Die Verzerrung der Projektion nimmt umso mehr zu, je weiter ein Punkt von diesen Ellipsen entfernt liegt, daher kommen, je nach Längengrad, 60 verschiedene Zonen zur Anwendung.

Es muss an dieser Stelle angemerkt werden, dass jede UTM Zone zusätzlich auch nach Breitengraden eingeteilt werden kann. Damit wird eine grobe geographische Zuordnung möglich. Diese Unterteilung erfolgt durch einen der Zonennummer nachgestellten Buchstaben. Ungünstigerweise gibt es in diesem Zusammenhang eine Namenskollision: Die „UTM Zone 32N“ kann beispielsweise entweder als der Teilbereich innerhalb der UTM Zone 32 vom Äquator bis  $8^{\circ}$  nördlicher Breite verstanden werden oder als der Bereich von der UTM Zone 32, welcher fast die gesamte nördliche Hemisphäre bis zum  $84^{\circ}$  Breitengrad abdeckt. Wie bereits erwähnt, ist im Zusammenhang mit dem EPSG Katalog Letzteres damit gemeint.

Für die vorliegende Arbeit muss daher Rücksicht darauf genommen werden, in welcher Zone sich die relevanten Datenpunkte befinden. Für die Aggregation

von Geoobjekten innerhalb eines Gemeindegebiets ist das jedoch kein Problem, da die Daten räumlich stark begrenzt sind. Für den ungünstigen Fall, dass die UTM-Zonengrenze genau durch eine Gemeinde verlaufen würde, wäre die zusätzliche Verzerrung vernachlässigbar, da die West-Ost-Ausdehnung von Gemeinden in der Regel nur wenige Kilometer beträgt.



Abb. 8: Traversale Mercator-Projektion [5]

Praktischerweise erfolgt die Umrechnung von Koordinaten von einem zu einem anderen Referenzsystem durch die PostGIS-Funktion „`ST_Transform()`“ und der Anwender oder die Anwenderin muss sich über die Berechnung keine Gedanken machen. Die aktuellen Formeln zur Berechnung der transversalen Mercator-Projektion sind unter [55, S. 53 ff.] zu finden.

## 4.5 Import von OSM-Daten in PostGIS

Vorbedingung für den Import von OSM-Daten ist, dass eine Datenbank mit der PostGIS-Erweiterung existiert. Um eine Datenbank mit dieser Erweiterung zu erstellen, muss zunächst eine reguläre PostgreSQL-Datenbank erstellt und einem bestehenden (Linux-)Nutzer oder einer Nutzerin zugeordnet werden. In einem zweiten Schritt wird die PostGIS-Erweiterung dieser Datenbank hinzugefügt. Es ist wichtig anzumerken, dass nur Datenbank-SuperuserInnen Extensions für bestehende Datenbanken erstellen können. Aus diesem Grund erfolgt der „`CREATE EXTENSION`“-Aufruf mit dem User „`postgres`“, welcher defaultmäßig Superuserrechte für die Datenbank besitzt.

```

1 $ sudo -u postgres psql --command "CREATE DATABASE [dbname] OWNER [username];"
2 $ sudo -u postgres psql -d [dbname] --command="CREATE EXTENSION postgis;"
```

Im nächsten Schritt erfolgt der Import der OSM-Daten durch das „osm2pgsql“-Tool. Die dazu benötigten OSM-Daten können über verschiedene Wege heruntergeladen werden (siehe Anhang).

Der Import erfolgt im nachfolgenden Beispiel in zwei Schritten, da der Datensatz in diesem Fall in zwei Teilen vorliegt.

```
1 $ osm2pgsql --database [dbname] --slim part1.osm --proj 32633
2 $ osm2pgsql --database [dbname] --slim --append part2.osm --proj
   32633
```

Die Option „–slim“ speichert temporäre Daten, welche beim Import anfallen, in der Datenbank und ist eine Voraussetzung dafür, dass ein zweiter Datensatz mit „–append“ hinzugefügt werden kann.

Die Option „–proj 32633“ legt das Koordinatenreferenzsystem fest. In diesem Fall wird „EPSG: 32633 - WGS 84 / UTM zone 33N“ verwendet.

Mit dem Import der OSM-Daten wurden im Wesentlichen folgende Tabellen in der Datenbank erstellt:

- planet\_osm\_point – Hier sind punktuelle Informationen, wie Verkehrsschilder, Bushaltestationen usw., hinterlegt.
- planet\_osm\_line – Hier werden Linien, wie z. B. Straßenzüge, gespeichert.
- planet\_osm\_polygon – Hier sind Grundflächen, Gemeindeflächen usw. als Polygone gespeichert.

Für die weitere Verarbeitung sind im Kontext dieser Arbeit vor allem die Straßenzüge, die Gebäude, Grundflächen und Gemeindegrenzen von Interesse. In der Regel wird es notwendig sein, die Aggregation der Geoobjekte auf einen bestimmten Bereich zu beschränken. Im vorgestellten Fall des kleinräumigen Bedarfsverkehrs ist dieser Bereich vermutlich auf das Verwaltungsgebiet von einer Gemeinde oder von mehreren aneinander grenzenden Gemeinden beschränkt, die sich zu einer Region zusammengeschlossen haben.

Ist solch eine Beschränkung gewünscht, dann sind prinzipiell zwei Vorgehensweisen denkbar. Entweder es wird bei jeder Datenbankabfrage über eine Klausel entsprechend dem zu betrachtenden Gebiet eine Beschränkung eingeführt oder alle Objekte, welche sich außerhalb des Gebiets von Interesse befinden, werden aus den Tabellen der Datenbank gelöscht (siehe Abschnitt A.3 auf Seite 78). Für ein produktives GIS-System ist vermutlich die erste Variante die bessere Wahl. Aus Gründen der Übersicht wird für diese Arbeit der zweite Ansatz gewählt, da dadurch die SQL-Abfragen vereinfacht werden.

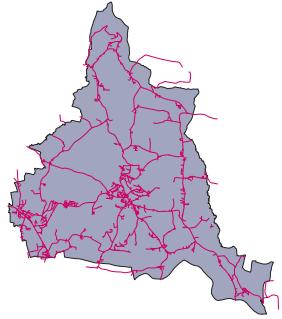


Abb. 9: Gemeinde mit Straßen-  
netz

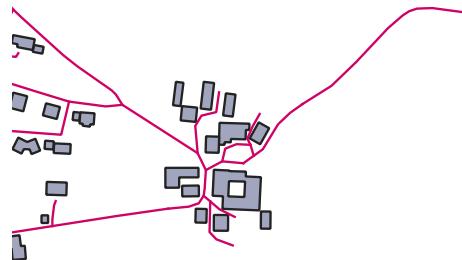


Abb. 10: Häuser an der Straße

Im OSM-Datensatz sind günstigerweise bereits solche administrativen Grenzen als Polygone in der Tabelle „planet\_osm\_polygon“ hinterlegt. Die Elemente werden über das Tag `anfboundary='administrative'` gekennzeichnet. Das Tag „admin\_level“ gibt dabei an, um welche Art von Grenze es sich handelt.

admin_level	Verwaltungseinheiten
4	Bundesland
6	Bezirk
8	Gemeinde
9	Stadt/Gemeindebezirk

Tab. 1: Ebenen der Verwaltungseinheiten

Die importierten Daten können nun mit einer Geoinformationssoftware (wie zum Beispiel QGIS) visualisiert werden.

## 4.6 Export des GSD-Graphen

Nach dem Import der Daten liegen nun die Straßen als Polygonzüge und die Häuser bzw. die Geoobjekte als Polygone von deren Grundrisse vor. Diese Informationen müssen nun schrittweise in einen GSD-Graphen überführt werden. Dazu müssen zunächst aus der Datenbank „planet\_osm\_line“ die relevanten Straßen extrahiert und alle Kreuzungen identifiziert werden. Darauf folgend werden die Geoobjekte auf das Straßennetz projiziert und dort ebenfalls als Knoten gespeichert. Als nächster Schritt werden aus den einzelnen Straßenzügen und den darauf befindlichen Knoten Subgraphen in Form von Kettengraphen gebildet. Aus diesen Subgraphen wird durch Vereinigung der GSD-Graph generiert. Dieser letzte Schritt erfolgt in unserer Implementierung in Python, alle zuvor genannten wurden als SQL-Abfragen umgesetzt.

#### 4.6.1 Extrahieren der Straßen

OSM speichert verschiedene Straßentypen mit dem Attribut „highway“ als Einträge der Tabelle „planet\_osm\_line“. Auch ein Fuß- oder ein Radweg ist mit diesem Attribut versehen. Mit dem Tool „exportGSD.py“ (siehe Abschnitt A.4 auf Seite 78) ist es möglich, alle Straßenzüge zu extrahieren und in einen GSD-Graphen zu überführen, dabei werden anhand einer Positivliste erlaubte Straßentypen aus der Tabelle „planet\_osm\_line“ herausgefiltert. Eine detaillierte Auflistung der möglichen Werte für das Attribut „highway“ findet sich in [58].

Zunächst wird eine Tabelle „streets“ erzeugt, welche neben einer ID auch den Polygonzug des Straßenabschnittes und gegebenenfalls das entsprechende „tunnel“-Attribut beinhaltet. Die möglichen Ausprägungen des Attributs „tunnel“ finden sich in [59]. Für die ID von Straßen wird ein zusammengesetzter String verwendet, der den GeoHash des Anfangspunktes und den GeoHash des Endpunktes des Polygonzugs getrennt durch ein „#“ enthält.

```
1 CREATE TABLE streets(id text, polychain geometry, tunnel text,  
PRIMARY KEY(id));
```

Mit dieser Anfrage wird die ganze Tabelle „planet\_osm\_line“ durchsucht und alle relevanten Einträge mit einem passenden Wert des Attributs „highway“ werden in die Tabelle „streets“ übernommen. Die entsprechende ID aus Anfangs- und Endpunkten berechnet wird und das „tunnel“-Attribut übernommen. Sollte dieses nicht existieren, dann wird es in der neuen Tabelle auf „no“ gesetzt.

```
1 INSERT INTO streets SELECT concat(ST_GeoHash(ST_Transform(  
ST_StartPoint(way), 4326)), '#', ST_GeoHash(ST_Transform(  
ST_EndPoint(way), 4326))), way, CASE WHEN tunnel IS NULL THEN 'no'  
ELSE tunnel END FROM planet_osm_line WHERE "highway" = 'primary'  
OR [...] OR "highway" = 'track';
```

Anm.: Die Filter für die Straßentypen(„highway“) werden im Programm „exportGSD.py“ aus einer Liste automatisch generiert und sind in dieser SQL-Anfrage nicht komplett aufgeführt.

Mit der Erstellung eines räumlichen Index für die Tabelle „streets“ ist der erste Zwischenschritt abgeschlossen.

```
1 CREATE INDEX pc_idx ON streets USING GIST(polychain);
```

#### 4.6.2 Finden von Kreuzungen

Da die Straßen nun als Polygonzüge vorliegen, ist der nächste Schritt das Auffinden von Kreuzungen. Eine mögliche Lösung dieses Problems wären geeignete Sweep-

Line-Algorithmen, welche zunächst die vorliegenden Straßenzüge in monotone Polygonzüge aufspalten, um danach mit einem Sweep-Line-Verfahren die Kreuzungspunkte zu identifizieren [11, 60]. Da hier allerdings ein räumlicher Index zur Verfügung steht, wird an dieser Stelle mit einem naiven Ansatz gearbeitet. Syntaktisch wird die Anfrage so gestaltet, dass jeder Polygonzug mit jedem anderen auf Kreuzungspunkte überprüft wird. Im Hintergrund erledigt jedoch die Datenbank mit Hilfe des räumlichen Index die Aufgabe, nur jene Polygonzüge zu überprüfen, die sich im selben Gebiet befinden. Es gibt in diesem Zusammenhang drei Spezialfälle, die berücksichtigt werden müssen:

1. Wenn der Polygonzug  $A$  auf Kreuzung mit dem Polygonzug  $B$  untersucht wird, kann ein Kreuzungsknoten entstehen. Derselbe Kreuzungsknoten würde entstehen, wenn Polygonzug  $B$  auf Kreuzung mit Polygonzug  $A$  untersucht wird. Es muss in diesem Fall vermieden werden, dass Duplikate erstellt werden. Das gilt auch für sogenannte Bypass-Situationen, bei denen sich Polygonzüge mehrmals schneiden können.
2. Im OSM-Datensatz werden Streckenabschnitte der Straßen durch Polygonzüge modelliert, dadurch können zwei Streckenabschnitte hintereinander liegen und der Endpunkt eines Abschnitts kann der Anfangspunkt des nächsten sein. Es handelt sich dabei um keine echte Kreuzung, weil der zugehörige Grad des Knotens 2 beträgt. Aus diesem Grund sind für den GSD-Graphen auch Kreuzungsknoten mit dem Grad 2 erlaubt.
3. Es muss sichergestellt werden, dass im Falle von Unterführungen oder Übergängen keine Kreuzungsknoten erstellt werden.

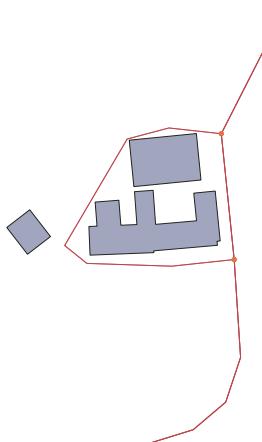


Abb. 11: Bypass-Situation



Abb. 12: Unterführung [57]

Für die Umsetzung wird zunächst eine Tabelle mit dem Namen „verticies“ erstellt. Diese soll sowohl Kreuzungsknoten als auch Knoten von projizierten Geoobjekten enthalten. Nach der Definition des GSD-Graphen erhalten die Kreuzungen das Gewicht 0.

```
1 CREATE TABLE verticies(id text, point geometry, weight float,
PRIMARY KEY(id));
```

Nun folgt die eigentliche SQL-Abfrage, welche die 3 oben gefundenen Spezialfälle berücksichtigen muss.

```
1 INSERT INTO verticies SELECT DISTINCT ST_GeoHash(ST_Transform((
ST_DumpPoints(ST_Intersection(a.polychain, b.polychain))).geom
,4326)),(ST_DumpPoints(ST_Intersection(a.polychain, b.polychain))
).geom,0 FROM streets as a, streets as b WHERE a.id != b.id AND
(ST_Touches(a.polychain,b.polychain) OR (ST_Intersects(a.
polychain,b.polychain) AND a.tunnel = b.tunnel));
```

Durch „DISTINCT“ wird der Spezialfall 1 behandelt; da jeder Knoten mit einer eindeutigen ID, bestehend aus dem GeoHash seiner Koordinate, identifiziert wird, können durch das Schlüsselwort „DISTINCT“ Duplikate vermieden werden. Der zweite Spezialfall von hintereinander liegenden Straßenabschnitten wird durch die Klausel „ST\_Touches“ abgedeckt, damit müssen sich die Polygonzüge nicht schneiden, sondern es genügt, dass sie sich in einem Punkt berühren. Der dritte Spezialfall wird mit der Klausel „ST\_Intersects(..) AND a.tunnel = b.tunnel“ behandelt. OSM stellt mit dem Attribut „tunnel“ die Information zur Verfügung, ob es sich um eine Unter- oder Überführung handelt. Nur wenn sich zwei sich kreuzende Polygonzüge auch auf derselben Ebene befinden, ist tatsächlich eine Kreuzung vorhanden.

Die Funktion „ST\_Intersection“ liefert die Menge an Kreuzungspunkten der beiden Polygonzüge. Mit Hilfe von „ST\_DumpPoints“ wird die Menge an Punkten als einzelne Anfragenzeilen zurückgegeben. Funktionen wie „ST\_DumpPoints“ werden als „Set Returning Functions“ bezeichnet.

Die Erstellung des räumlichen Index für die Tabelle „verticies“ wird erst etwas später vorgenommen, nachdem auch die Knoten der projizierten Geoobjekte eingefügt worden sind.

#### 4.6.3 Projektion der Geoobjekte auf das Straßennetz

Laut Definition der Geoobjekte-Straßennetz-Projektion müssen zunächst die Schwerpunkte der Polygone, die Geoobjekte repräsentieren, gefunden werden. Dazu wird die Tabelle „buildings“ erstellt, welche neben einer eindeutigen ID auch den Schwerpunkt des jeweiligen Polygons beinhaltet:



Abb. 13: Gefundene Kreuzungsknoten

```
1 CREATE TABLE buildings(id text, centerpoint geometry, PRIMARY KEY(
    id));
```

OSM stellt für das Attribut „building“ zur Verfügung, welches angibt, ob es sich bei einem Polygon der Tabelle „planet\_osm\_polygon“ um ein Gebäude handelt oder nicht. Zusätzlich verwenden wir an dieser Stelle einen Filter „addr:housenumber IS NOT NULL“. Damit soll sichergestellt werden, dass nur Gebäude, welche eine Hausnummer besitzen, in Betracht gezogen werden. Damit werden Gartenhäuschen, Garagen, landwirtschaftliche Gebäude usw. herausgefiltert.

```
1 INSERT INTO buildings SELECT ST_GeoHash(ST_Transform(ST_Centroid(
    way), 4326)), ST_Centroid(way) FROM planet_osm_polygon WHERE "
    building" = 'yes' AND addr:housenumber IS NOT NULL;
```

Eine Alternative zu diesem vorgeschlagenen Filter besteht darin, nach der Fläche der Gebäudepolygone zu filtern und alle sehr kleinen Gebäude, welche z. B. kleiner als  $10m^2$  sind, nicht zu betrachten.

```
1 WHERE "building" = 'yes' AND way_area >= 10
```

In den Testdaten ergab sich zum Beispiel ein Problem mit dem Datensatz des 15. Wiener Gemeindebezirks Rudolfsheim-Fünfhaus. Hier sind anscheinend die Daten für das Feld „addr:housenumber“ nicht komplett eingetragen. Diese Vorgehensweise ist für eine Evaluierung des hier vorgestellten Algorithmus gut genug, es würde sich allerdings für ein Produktivsystem nicht eignen.

Für die Tabelle „buildings“ wird nun ebenfalls ein räumlicher Index angelegt.

```
1 CREATE INDEX cp_idx ON buildings USING GIST(centerpoint);
```

Nun erfolgt die eigentliche Projektion auf das Straßennetz. Es wird zunächst für jeden Schwerpunkt mit Hilfe des Nearest-Neighbour-Operators der nächstgelegene

Straßen-Polygonzug ermittelt. Mit Hilfe von „ST\_ClosestPoint()“ wird der nächstgelegene Punkt zum Schwerpunkt innerhalb des Polygonzugs ermittelt. Dieser Punkt wird als ein Knoten mit dem Gewicht 1 in die Tabelle „verticies“ eingetragen. Es kann dabei sein, dass mehrere Geoobjekte demselben Punkt auf dem Straßennetz zugeordnet werden. Dieser Spezialfall wurde in der Arbeit bereits mit der Abbildung (siehe Abb. 6 auf Seite 11) aufgezeigt.

```
1 INSERT INTO verticies SELECT b.id, ST_ClosestPoint(s.polychain, b.
    centerpoint), 1 FROM buildings AS b CROSS JOIN LATERAL (SELECT
    polychain FROM streets ORDER BY polychain <-> b.centerpoint
    LIMIT 1) AS s;
```

Nun kann auch der räumliche Index für die Tabelle „verticies“ erstellt werden, da nun alle Geoobjekt- und Kreuzungsknoten erstellt worden sind.

```
1 CREATE INDEX v_idx ON verticies USING GIST(point);
```

#### 4.6.4 Erstellung der Subgraphen

Die Idee dieses Schrittes ist es, anhand jedes Straßenzuges und des ihn berührenden Knotens einen Subgraphen zu erstellen. Dieser Subgraph hat die Form eines Kettengraphen. Zunächst wird die Tabelle „subgraphs“ erstellt, in der für jeden Straßenzug „street\_id“ die zugeordneten Knoten „vertex\_id“ mit ihrem Gewicht „vertex\_weight“ und der absoluten Position des Knoten „vertex\_position“, ausgehend vom Anfangspunkt des Straßenzuges, abgelegt werden.

```
1 CREATE TABLE subgraphs(street_id text, vertex_id text,
    vertex_weight float, vertex_position float);
```

Der SQL-Aufruf zum Befüllen dieser Tabelle wertet für jeden Straßenzug die ihn berührenden Knoten aus.

```
1 INSERT INTO subgraphs SELECT s.id, v.id, v.weight, (
    ST_LineLocatePoint(s.polychain, v.point) * ST_Length(s.polychain)
) AS pos FROM streets AS s CROSS JOIN LATERAL (SELECT * FROM
verticies WHERE ST_Intersects(point, ST_Buffer(s.polychain, 0.001,
'endcap=round join=round'))) AS v ORDER BY s.id, pos;
```

In dieser Abfrage werden zwei noch nicht vorgestellte PostGIS-Funktionen verwendet:

1. „ST\_LineLocatePoint()“ – liefert einen relativen Wert von 0 bis 1 der Position eines Punktes auf einem Polygonzug ausgehend von dessen Startpunkt
2. „ST\_Length()“ – liefert die absolute Länge eines Polygonzugs

In der experimentellen Anwendung wurde festgestellt, dass nicht alle Knoten zuverlässig zugeordnet werden konnten. Abhilfe hat die Funktion „ST\_Buffer()“ geschaffen, welche den Polygonzug radial expandiert und ihn somit zu einer Fläche macht. Als Parameter für die Expansion wurde 1 Millimeter gewählt. Die Ursache, warum dies notwendig war, wurde nicht genau eruiert, denkbar ist, dass es sich bei diesem Problem um Rundungsfehler gehandelt hat.

Durch die Klausel „ORDER BY s.id, pos“ wird die Tabelle so erstellt, dass die Tabelle erstens nach Straßenzügen geordnet ist und zweitens dass die Knoten aufsteigend von ihrer Entfernung zum Anfangspunkt des Straßenzugs aufgelistet werden. Diese Sortierung ermöglicht es, dass die Menge der Subgraphen einfach in eine Adjazenzmatrix umgeformt werden kann.

Der aufmerksamen Leserin und dem wachen Leser wird an dieser Stelle aufgefallen sein, dass die Zuordnung von Knoten zum Straßenzug zweimal passiert, einmal beim Erstellen der Knoten und einmal beim Erstellen des Subgraphen. Das ist in der Tat eine gewisse Redundanz, welche weiter optimiert werden könnte. Diese Optimierung würde erfordern, dass bei der Knotenerstellung bereits die Referenz auf den Straßenzug „street\_id“ mit gespeichert würde. Dieses Vorgehen birgt allerdings ein Problem in sich: Bei Kreuzungsknoten müssten mindestens zwei Straßenzüge zugeordnet werden oder alternativ dazu müssten die Kreuzungsknoten entsprechend dupliziert werden.

#### 4.6.5 Erstellung der Adjazenzmatrix

Die Erstellung der Adjazenzmatrix erfolgt nun durch eine Auswertung der Tabelle „subgraphs“ in ihrer geordneten Reihenfolge. Der in Python implementierte Algorithmus erkennt auf Grund einer geänderten „street\_id“, dass ein neuer Subgraph beginnt. Alle Kanten werden dann mit ihrer entsprechenden Distanz in der Adjazenzmatrix eingefügt. Für die Berechnung der Distanz wird die Differenz der Knotenposition zu der Position des Vorgängerknotens herangezogen.

Im vorliegenden Fall von Straßennetzen handelt es sich zumeist um Graphen, die eine geringe Dichte aufweisen, was in einer dünn besetzten Adjazenzmatrix resultiert. Aus diesem Grund wird die Adjazenzmatrix in Form einer „Sparse Matrix“ im Format „Compressed Sparse Row (kurz CSR)“ gespeichert.

Im CSR-Format wird eine dünn besetzte Matrix durch drei Vektoren repräsentiert. Der Vektor *values* ist eine Auflistung aller Matrixeinträge  $a_{ij} \neq 0$  beginnend mit der ersten Zeile von links nach rechts. Der Vektor *colidx* speichert in derselben Reihenfolge den Spaltenindex  $j$  aller  $a_{ij} \neq 0$ . Der Vektor *rowidx* schließlich speichert

für jede Zeile  $i$  einen Index  $p$ , sodass  $p$  die Position darstellt, an der die  $i$ -te Zeile in den Vektoren  $values_p$  und  $colidx_p$  beginnt. Zusätzlich fasst das letzte Element von  $rowidx$  einen Index, welcher auf das fiktive Element nach dem Ende von  $values$  und  $colidx$  zeigt, dadurch ist auch eine Angabe, wie viele Elemente in  $values$  und  $colidx$  gespeichert sind, mitcodiert [65, S. 92 f.].

Als Beispiel wird folgende Matrix gezeigt:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 0 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

Die Matrix  $A$  wird durch die Vektoren  $values$ ,  $colidx$ ,  $rowidx$  im CSR-Format repräsentiert:

$$\begin{aligned} values &= (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12) \\ colidx &= (1 \ 4 \ 1 \ 2 \ 4 \ 1 \ 3 \ 4 \ 5 \ 3 \ 4 \ 5) \\ rowidx &= (1 \ 3 \ 6 \ 10 \ 12 \ 13) \end{aligned}$$

Soll nun das Element  $a_{ij}$  gefunden werden, wird zunächst  $p = rowidx_i$  ermittelt. Für den Fall  $colidx_p = j$  gilt:  $a_{ij} = values_p$ . Sollte das nicht der Fall sein, wird  $p$  so lange um 1 inkrementiert, bis entweder  $colidx_p = j$  eintritt oder  $p = rowidx_{i+1}$ . Im zuletzt genannten Fall wurde kein Eintrag gefunden und es gilt  $a_{ij} = 0$ .

Im Zusammenhang mit dem CSR-Format gilt es implementierungsabhängige Eigenschaften zu berücksichtigen. Das in dieser Arbeit verwendete Python-Paket „scipy“ unterscheidet, ob Matrixelemente explizit oder implizit 0 sind. Gerade im Zusammenhang mit einer Adjazenzmatrix wird hier folgendes Problem sichtbar: Wenn jedes Element eine Kantendistanz darstellt, dann müssten korrekterweise alle Elemente, die keine Kanten sind, den Wert  $\infty$  annehmen. Aber um die Einträge in der dünn besetzten Matrix gering zu halten, ist es erwünscht, dass nicht existente Kanten durch die implizite 0 definiert sind. Hingegen sollen Kanten, die eine Kantendistanz von 0 besitzen, durch eine explizite 0 dargestellt werden. Aus pragmatischen Gründen wird in dieser Arbeit für diesen Fall anstelle der expliziten 0 ein sehr kleiner Wert von  $10^{-6}m = 1\mu m$  verwendet, um ein eindeutiges Unterscheidungsmerkmal zwischen keiner Kante und einer Kante mit einer Kantendistanz von 0 zu erreichen, ohne auf die Unterscheidung von expliziter und impliziter 0 angewiesen zu sein.

Dieses Vorgehen kann als eine Art Vorsichtsmaßnahme gesehen werden und könnte mit entsprechender Sorgfalt in der Umsetzung des Algorithmus auch weggelassen werden. Da aber auch mit Paketen von Drittanbietern gearbeitet werden soll, lassen wir diese Vorsichtsmaßnahme zunächst aktiv.

Die Verwendung des CSR-Formates bringt wesentliche Vorteile mit sich. Zum einen benötigt es, im Falle des Graphen mit geringer Dichte, weniger Speicherplatz und zum anderen ist die Operation des Findens der angrenzenden Kanten eines Knotens effizienter implementierbar. Im Falle einer herkömmlichen Matrix müssten  $|E|$  Elemente untersucht werden. In der CSR-Darstellung werden dagegen alle Elemente, die implizit 0 sind, einfach ausgelassen und es müssen nur  $\deg(v)$  Elemente betrachtet werden. Der modifizierte Dijkstra-Algorithmus nach [52, S. 5], welcher etwas später vorgestellt wird, nutzt diese Eigenschaften zum Beispiel in der Funktion „neighbors()“ aus.

#### 4.6.6 Eckdaten der Testdatensätze

Zur Evaluierung der GOA werden in dieser Arbeit drei Testdatensätze verwendet, welche unterschiedliche Regionen in Österreich repräsentieren.

Der Testdatensatz „Neukirchen“ stellt die Heimatgemeinde des Autors dar. Dieser Datensatz wurde vor allem für die Entwicklung verwendet, weil gewisse Eigenheiten, wie Überführungen usw., dem Autor bekannt sind. Der Ort Neukirchen an der Vöckla zeichnet sich dadurch aus, dass er ein Ortszentrum und mehrere Ortschaften besitzt, welche sich über eine relativ große Fläche von  $24km^2$  verteilen. Es gibt oftmals Strukturen von einzelnen Häuseransammlungen, die von Feldern umgeben sind, sowie einzelne Häuser, die sich etwas abgeschnitten in einem Wald oder auf einem Hügel befinden.

Die Gemeinde Purbach ist eine Gemeinde, welche sich am Neusiedlersee befindet. Dieser Testdatensatz wurde gewählt, weil hier bereits ein Bedarfsverkehrsprojekt umgesetzt worden ist, welches mit einer händischen Anonymisierung gearbeitet hat. Die Gemeinden rund um den Neusiedlersee zeichnen sich durch kompakte Ortszentren aus, die vom Erscheinungsbild der sogenannten Streckhöfe geprägt sind. Streckhöfe sind lang gezogene Bauten, die dicht an dicht stehen, eine schmale Hausfront hin zur Straße besitzen und senkrecht zur Straße ihre längste Ausdehnung haben. Zusätzlich zum Ortszentrum gibt es vereinzelte Gebäude, die sich etwas in der Peripherie befinden.

Der dritte Testdatensatz entspricht dem 15. Wiener Gemeindebezirk „Rudolfsheim-Fünfhaus“. Dieser Testdatensatz zeichnet sich durch Wohnhäuser aus, die in

einem Raster angeordnet sind, welches annähernd einem „Manhattan“-Raster entspricht. Des Weiteren beinhaltet dieser Datensatz unter anderem Strukturen wie eine Kleingartensiedlung, Parks, einen Friedhof und eine Kaserne.

Betreffend die Datenqualität der OSM-Daten muss an dieser Stelle angemerkt werden, dass für die Datensätze „Neukirchen“ und „Purbach“ die Hausnummern aller bewohnten Gebäude vorhanden waren und sich zur Identifikation geeignet haben. Im Falle von „Wien 15ter“ wurde für die Ermittlung der Gebäude auf das alternative Verfahren über die Gebäudegrundfläche  $\geq 10m^2$  zurückgegriffen, zum einen da die Gebäudeadressen nicht komplett vorhanden waren und zum anderen damit auch die Kleingartensiedlung, welche teilweise auch durchgehend bewohnt ist, abgedeckt wird.

	Neukirchen	Purbach	Wien 15ter
Anz. Knoten	1956	1280	3880
Anz. Kanten	2151	1473	4209
Graph Dichte [ppm]	1125	1799	559
Anz. Geoobjekte	792	622	3158
Anz. Kreuzungen ( $deg > 2$ )	746	406	528
Anz. unechter Kreuzungen ( $deg = 2$ )	418	252	194
durchschnittlicher Grad von Kreuzungen	3.04	3.11	3.32
Straßennetz [km]	134.13	161.89	77.42

Tab. 2: Eckdaten der Testdatensätze

Folgendes kann an diesen Eckdaten beobachtet werden:

- Die Graphen sind nicht besonders groß, die Anzahl der Knoten  $n = |V|$  beträgt nur einige tausend. State-of-the-Art-Graph-Partitionierungsalgorithmen, welche zum Beispiel an der 10. DIMACS Challenge [7] teilgenommen haben, versuchen sich an Graphen, deren Knotenzahl bis zu mehreren 10 Millionen beträgt. Darüber hinaus gibt es Forschungen und Anwendungen, die mit Graphen im Bereich von einer Milliarde Knoten operieren [42].
- Die Anzahl der Kanten ist relativ gesehen klein und beträgt einige tausend. Die Dichte der Graphen bewegt sich im Promillebereich. Diese Zahlen geben einen Hinweis darauf, was dem menschlichen Auge beim Betrachten eines Straßengraphen, dessen Knoten ihre räumlichen Koordinaten behalten haben, intuitiv auffällt, nämlich dass sich potentielle Cluster bei der Betrachtung durch ihre räumliche Nähe ausmachen lassen. Anders gesagt: Die Struktur des Graphen

ist offensichtlich. Wäre dies nicht der Fall, könnten für die weitere Untersuchung der Graphen Verfahren, die der „Spectral Graph Theory“ zugeordnet sind, nützlich sein [69]. Für den vorliegenden Anwendungsfall scheint das jedoch über das Ziel hinauszugehen.

- Der Graph setzt sich nicht nur aus den Geoobjekten und echten Kreuzungen zusammen, sondern auch aus „unechten“ Kreuzungen, welche den Grad  $\deg(v) = 2$  aufweisen. Diese Knoten sind „Überbleibsel“, welche auf eine Stückelung von Straßenzügen in den Ausgangsdaten zurückzuführen sind. Durch einen zusätzlichen Bereinigungsschritt in der Phase der Grapherstellung könnten diese Kreuzungsknoten mit Gewicht  $w(v) = 0$  und Grad  $\deg(v) = 2$  ohne Informationsverlust eliminiert werden. Für die Umsetzung des GOA-Algorithmus wirken sich diese Knoten, abgesehen von geringfügigen Geschwindigkeitseinbußen, nicht störend aus.
- Echte Kreuzungsknoten haben in den hier verwendeten Testdatensätzen tendenziell einen Grad  $\deg(v) = 3$ . Allerdings kommen im Datensatz „Wien 15ter“ auf Grund der Ähnlichkeit zum Manhattan-Raster auch Kreuzungsknoten mit dem Grad  $\deg(v) = 4$  vermehrt vor. Spezielle Kreuzungen wie Kreisverkehre werden in den Datensätzen durch mehrere Knoten und Streckenabschnitte modelliert.

### Definition 13 (Dichte eines Graphen)

*Die Dichte eines ungerichteten, einfachen Graphen definiert sich wie folgt [77]:*

$$D = \frac{2|E|}{|V|(|V| - 1)}$$

Eine weitere Beobachtung ergibt sich aus der Aufgabenstellung: Da die Aggregation der Knoten die Eigenschaft der  $a$ -Anonymität erfüllen muss und die gesetzlich vorgeschriebene Aggregation wenige Haushalte erfordert, werden die Graphen in viele relativ kleine, ungefähr  $a$ -große Teile zerteilt.

## 5 GOA Überblick

Als Vorüberlegung soll zunächst folgendes Gedankenexperiment versucht werden: Die Aggregation im Falle einer 1-Anonymität ist eine triviale Lösung, in der der gesamte Graph in Teilgraphen zerfällt, die jeweils einen Knoten mit dem Gewicht  $w(v) = 1$  beinhalten. Im Falle der 2-Anonymität wäre eine mögliche Vorgehensweise, immer paarweise Knoten – einen Knoten und dessen nächsten Nachbarn – für die

Teilgraphen auszuwählen. Aber bereits in diesem Fall wird ersichtlich, dass erstens durch die Auswahl des ersten Paars die weitere Ausbreitung der Lösung determiniert wird und zweitens mit Residuen zu rechnen ist. Im Falle einer  $a$ -Anonymität könnten immer ein Knoten und dessen  $k$ -nächster Nachbar für  $k = a$  ausgewählt werden, wobei der 1-nächste Nachbar per Definition der ausgewählte Knoten selbst ist. Auch hier würden sich, abhängig von der Auswahl der Knoten, verschiedene Lösungsmöglichkeiten ergeben.

Das bereits vorgestellte „Similarity-Partitioning“ erstellt einen Umkreis um einen zentralen Punkt. Für die  $k$ -nächsten Nachbarn im Straßengraphen stellt sich die Frage, wie so ein Zentrum definiert werden kann. Eine Möglichkeit ist die Verwendung des zentralen Knotens eines Graphen.

Der zentrale Knoten eines Graphen lässt sich mit Hilfe folgender Definitionen bestimmen [78]:

#### **Definition 14 (Exzentrizität eines Knotens)**

*Die Exzentrizität  $\epsilon(v)$  eines Knotens entspricht dem Abstand des maximal entfernten Knoten*

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

#### **Definition 15 (Radius eines Graphen)**

*Der Radius  $r$  eines Graphen ist definiert durch die minimale Exzentrizität aller Knoten*

$$r = \min_{v \in V}(\epsilon(v))$$

#### **Definition 16 (Zentraler Knoten eines Graphen)**

*Der zentrale Knoten eines Graphen ist derjenige, dessen Exzentrizität dem Radius entspricht, oder anders formuliert der Knoten mit der minimalen Exzentrizität.*

Als zentrale Knoten kommen nur Knoten mit einem Gewicht von  $w(v) = 1$  infrage. Für den Fall der  $k$ -NN im Graphen muss festgestellt werden, dass ein zufällig ausgewählter Knoten nicht zwingend auch dem zentralen Knoten des durch die  $k$ -nächsten Knoten aufgespannten Teilgraphen entspricht. Das ist immer dann der Fall, wenn mehrere verschiedene Knoten denselben  $kNN$ -Teilgraphen aufspannen können.

### **Definition 17 (Knoten mit Gewicht ( $w = 1$ )**

Die Funktion  $V_G(G)$  liefert alle Knoten eines GSD- oder Teilgraphen oder einer Menge von Graphen, deren Gewicht 1 ist; damit sind alle Knoten gemeint, die Geobjekten entsprechen.

### **Definition 18 (Knoten mit Gewicht ( $w = 0$ )**

Die Funktion  $V_K(G)$  liefert alle Knoten eines GSD- oder Teilgraphen oder einer Menge von Graphen, deren Gewicht 0 ist; damit sind alle Knoten gemeint, die Kreuzungsknoten entsprechen.

### **Definition 19 (Teilgraph)**

Ein Teilgraph  $TG(V_T, E_T, w, d)$  ist ein Untergraph eines GSD-Graphen  $G(V, E, w, d)$ , sodass für die Knoten  $V_T \subseteq V$  und für die Kanten  $E_T \subseteq E$  gilt. Die Menge an Kanten  $E_T$  entspricht den Straßenabschnitten, die zur Verbindung der Knoten notwendig sind.

### **Definition 20 ( $kNN$ -Teilgraph)**

Dies ist ein Teilgraph, der durch den Knoten  $v$  und dessen  $k$ -nächsten Nachbarn aus  $V_G(G)$  bestimmt wird. Die Menge an Knoten  $V_T = V_G(TG) \cup V_K(TG)$  entspricht den  $k$ -nächsten Nachbarknoten  $V_G(TG)$  des Knoten  $v$  und den entsprechenden Kreuzungsknoten  $V_K(TG)$ .

**Lemma 1** Wenn ein  $kNN$ -Teilgraph  $TG$  von exakt einem Knoten  $v_i \in V_G(TG)$  bestimmt wird, dann ist  $v_i$  der zentrale Knoten von  $TG$ .

**Beweis:** Angenommen ein weiterer Knoten  $v_j \in V_G(TG)$  würde den  $kNN$ -Teilgraphen bestimmen, dann wäre  $TG$  durch  $v_i$  und  $v_j$  bestimmt, was einen Widerspruch zur Bestimmtheit durch exakt einen Knoten verursacht. Daher kann es auch keinen Knoten mit  $\epsilon(v_j) < \epsilon(v_i)$  geben.  $\square$

Für den Fall, dass  $TG$  durch mehrere Knoten bestimmt wird, genügt es, sich auf den zentralen Knoten  $v$  von  $TG$  zu beschränken, da dieser mit den geringstmöglichen Kosten  $cost(TG)$  behaftet ist.

### **Definition 21 (Kosten eines Teilgraphen)**

Die Kosten  $cost(TG, v_c)$  eines Teilgraphen werden durch die Summe der Distanzen des Knoten  $v_c \in V_G(TG)$  zu den übrigen Knoten aus  $V_g \setminus \{v_c\}$  ermittelt.

$$cost(TG, v_c) = \sum_{v_i \in V_G(TG) \setminus \{v_c\}} d(v_c, v_i)$$

Es gilt  $w(v_i) = 1, \forall v_i \in V_g(TG)$ . Die Schreibweise mit einem Parameter  $cost(TG)$  berechnet die Kosten in Bezug auf den zentralen Knoten von  $TG$ .

Die Definition von  $cost(TG)$  kann auch als das verwendete Kompaktheitsmaß verstanden werden, ein Teilgraph ist umso kompakter, je geringer die Kosten sind. Durch die Summierung der einfachen Distanzen ist dieses Vorgehen analog dem  $k$ -median-Verfahren in der Standortplanung zu verstehen.

Zur Implementierung eines GOA-Algorithmus wird das folgende mehrstufige Verfahren skizziert:

1. Für jeden Knoten  $v_i \in V_G(G_{GSD})$  wird der durch ihn bestimmte ( $a$ -anonyme) NN Teilgraph  $TG_{v_i}$  berechnet.
2. Ermittlung der Menge aller eindeutig bestimmten Teilgraphen  $TG^*$ .  $TG^*$  ist die Vereinigung aller  $\{TG_{v_i}\}$ , für die gilt, dass  $v_i$  dem zentralen Knoten von  $TG_{v_i}$  entspricht.
3. Optimierung einer Auswahl von sich nicht überschneidenden Teilgraphen  $TG_{opt}^* \subseteq TG^*$  durch Minimierung einer Kostenfunktion  $c_{opt}(TG_{opt}^*)$ , unter der Berücksichtigung der Disjunktheit der Knoten für alle Teilgraphen  $V_G(TG_{v_i}) \cap V_G(TG_{v_j}) = \emptyset, \forall v_i \neq v_j$ .
4. Zuweisung der Residuen  $v_r \in \{V_G(G_{GSD}) \setminus V_G(TG_{opt}^*)\}$  zu dem nächstgelegenen zentralen Knoten  $v_i$  eines Teilgraphen  $TG_{v_i} \in TG_{opt}^*$ . Durch die Zuweisung von  $v_r$  entsteht eine Menge an neuen Teilgraphen  $TG_{res}^*$ .
5. Neuberechnung der zentralen Knoten für jeden Teilgraphen mit zugeordneten Residuen für alle  $TG \in TG_{res}^*$ .
6. „Stehlen“ von Knoten eines Nachbar-Teilgraphen. Wenn die Distanz  $d(v_s, v_i)$  des Knoten zum zentralen Knoten vom Teilgraphen  $TG_{v_i}$  größer ist als die Distanz  $d(v_s, v_j)$  zu dem zentralen Knoten eines anderen Teilgraphen  $TG_{v_j}$ , dann kann der Teilgraph  $TG_{v_j}$  den Knoten  $v_s \in V_G(TG_{v_i})$  „stehlen“. Die Voraussetzung ist, dass die Anzahl der Knoten des „bestohlenen“ Teilgraphen  $|V_G(TG_{v_i})| > a$  größer als die geforderte  $a$ -Anonymität ist. Die Menge an Teilgraphen nach diesem Schritt wird mit  $TG_{fin}^*$  bezeichnet.
7. Neuberechnung des minsum-Zentrums  $v_m$  für jeden Teilgraphen  $TG \in TG_{fin}^*$ . Anm.: In diesem Schritt wird abschließend anstelle des zentralen Knotens der Knoten mit der minimalen Summe der Distanzen verwendet, entsprechend dem  $k$ -median-Verfahren.

8. Vereinigung der Basisgebiete in der Ebene, welche den Knoten  $V_G(TG_{v_i})$  eines Teilgraphen  $TG_{v_i} \in TG_{fin}^*$  zugeordnet sind, zu Territorien mit dem Zentrum der zugeordneten Koordinate des jeweiligen minsum-Knotens  $v_m$ .

Anm.: Die Schritte 5. und 6. sind als optional zu betrachten.

### **Definition 22 (Kostenfunktion $c_{opt}()$ )**

$c_{opt}(TG_{opt}^*)$  ist eine Linearkombination der Summe der Kosten der Teilgraphen und der Bewertung der Anzahl der nicht zugewiesenen Residualknoten  $V_R = V_G(G_{GSD}) \setminus V_G(TG_{opt}^*)$  multipliziert mit einem Faktor  $r_{penalty}$

$$c_{opt}(TG_{opt}^*) = \sum_{TG_i \in TG_{opt}^*} cost(TG_i) + |V_R| \cdot r_{penalty}$$

Während die Erstellung des GSD-Graphen das ursprüngliche Problem der Aggregation von Geoobjekten von der Ebene in einen Graphen transferiert, wird mit dem letzten genannten Punkt das Ergebnis der Aggregation wieder in die Ebene transferiert.

## **6 GOA Berechnung**

Im folgenden Kapitel wird erläutert, welche einzelnen Schritte für die Berechnung einer Geoobjekte-Aggregation notwendig sind. Für die Implementierung ist anzumerken, dass es für viele Schritte genügt, anstelle eines kompletten Teilgraphen den bestimmenden Knoten  $v_i$ , die Knoten  $V_G$  mit einem Gewicht  $w(v) = 1$ , die Kosten des Teilgraphen  $c = cost(TG, v_i)$  und die Exzentrizität  $e = \epsilon(v_i, V_G)$  in einem Tupel  $(v_i, V_G, c, e)$  zu speichern. Die zusätzlichen Informationen, die der Teilgraph in sich trägt, entsprechen den Kreuzungsknoten  $V_K$  mit Gewicht  $w(v) = 0$  sowie den betreffenden Straßenabschnitts-Kanten  $E_T$ . Da Kreuzungsknoten und Straßenabschnitts-Kanten nur für die Berechnung der Distanzen  $d(v_i, v_j), v_i, v_j \in V_G$  benötigt werden, genügt es, wenn diese Informationen einmalig im GSD-Graphen  $G_{GSD}$  gespeichert sind.

### **6.1 Bestimmen der ( $a$ -anonymen) NN Teilgraphen**

Zur Bestimmung der Teilgraphen wird vorgeschlagen, auf den Algorithmus „Geodesic  $k$  nearest labeled neighbors“ (kurz geodesic kNN) zurückzugreifen, welcher im Kapitel siehe Abschnitt 7.1 auf Seite 52 im Detail vorgestellt wird. Bei diesem Algorithmus handelt es sich um eine Abwandlung des bekannten Dijkstra-Algorithmus,

welcher den kürzesten Pfad zwischen zwei Knoten in einem Graphen bestimmt. Der Algorithmus geodesic kNN findet für alle Knoten  $V$  die  $k$ -nächsten Nachbarn aus der Menge der Knoten  $V_G$  mit Gewicht  $w(v) = 1$ . Es werden also nur Knoten als nächste Nachbarn betrachtet, die Geoobjekte repräsentieren. Kreuzungsknoten  $V_K$  werden für die Berechnung der Distanzen berücksichtigt, sie zählen jedoch nicht zu den Kandidaten für die nächsten Nachbarn. Als Ergebnis liefert dieser Algorithmus für jeden Knoten  $v_i \in V = V_G \cup V_K$  eine Liste  $kNN(v_i)$  von Tupeln  $(d(v_i, v_n), v_n)$ , wobei jedes  $v_n$  für einen  $k$ -nächsten Nachbarn steht und der Abstand  $d(v_i, v_n)$  der Abstand vom Knoten  $v_i$  zu dem jeweiligen  $k$ -nächsten Nachbarknoten entspricht. Diese Liste ist gemäß dem Abstand in einer aufsteigenden Reihenfolge geordnet. Per Definition ist der 1-nächste Nachbar eines Knoten der Knoten selbst und die Distanz  $d(v_i, v_i) = 0$ .

Wie bereits erwähnt, werden in der Implementierung nicht die kompletten Teilgraphen gespeichert, sondern jeweils nur der bestimmende Knoten  $v_i$ , die Menge der nächsten Nachbarn  $V_G$ , die Kosten des Teilgraphen  $c$  und die Exzentrizität  $e$  des bestimmenden Knotens im Teilgraphen. Da die Listen  $kNN(v_i)$  die Knoten gemäß ihrem Abstand in aufsteigender Reihenfolge beinhalten, lassen sich die Menge der Nachbarknoten, die Kosten des Teilgraphen und die Exzentrizität des bestimmenden Knotens auf direktem Weg berechnen.

$$\begin{aligned} V_G(TG_{v_i}) &= \bigcup_{i=1}^k \{v(kNN(v_i)[i])\} \\ cost(TG_{v_i}) &= \sum_{i=1}^k d(kNN(v_i)[i]) \\ \epsilon(v_i, TG_{v_i}) &= d(kNN(v_i)[k]) \end{aligned}$$

Anm.: Für die Exzentrizität wird einfach das letzte Element  $k$  in der Liste  $kNN(v_i)$  ausgewählt, da dies dem Knoten mit dem weitesten Abstand zu  $v_i$  im Teilgraphen  $TG(v_i)$  entspricht.

## 6.2 Vorauswahl der eindeutig bestimmten Teilgraphen

Zur Auswahl der eindeutig bestimmten Teilgraphen werden nur diejenigen Teilgraphen  $TG_{v_i}$  verwendet, für die  $v_i$  dem zentralen Knoten von  $TG_{v_i}$  entspricht. Für die Umsetzung wird in der Menge der  $\{TG_{v_i}\}, \forall v_i \in V_G(G_{GSD})$  danach gesucht, ob die Knotenmenge  $V_G(TG_{v_i})$  mehrmals vorkommt. Kommt diese nur einmal vor, dann ist  $v_i$  der zentrale Knoten von  $TG_{v_i}$  (siehe Lemma 1 auf Seite 39). Kommt diese Knotenmenge mehrmals vor, so wird nach dem Knoten mit der geringsten Exzentrizität

gesucht, welcher dadurch der zentrale Knoten ist. Folgender Pseudecode versucht mit Hilfe eines Hashtables diese Aufgabe umzusetzen. In der Programmiersprache Python ist zum Beispiel eine unkomplizierte Implementierung mit dem Datentyp „dict()“ als Hashtable möglich, wobei der verwendete Schlüssel die Eigenschaft besitzen muss, „hashbar“ zu sein. In Python ist der Datentyp „frozenset()“ hashbar, während der Datentyp „set()“ es nicht ist [64].

---

**Algorithm 1:** Ermittlung der eindeutig bestimmten Teilgraphen

---

**Data:** Menge aller Teilgraphen  $\{TG_{v_i}\}, \forall v_i \in V_G(G_{GSD})$

**Result:** Menge aller eindeutig bestimmten Teilgraphen

$$TG^* = \{TG_{v_j}\} \subseteq \{TG_{v_i}\}, v_j \text{ ist der zentrale Knoten von } TG_{v_j}$$

```

begin
     $H \leftarrow \text{HashTable}()$ 
    forall  $TG_{v_k} \in \{TG_{v_i}\}$  do
        if  $H$  contains  $V_G(TG_{v_k})$  then
             $TG_{v_o} \leftarrow \text{find}(H, V_g(TG))$ 
            if  $\epsilon(v_k, TG_{v_k}) < \epsilon(v_o, TG_{v_o})$  then
                 $\text{update}(H, V_G(TG_{v_k}), TG_{v_k})$ 
            else
                 $\text{insert}(H, V_G(TG_{v_k}), TG_{v_k})$ 

```

---

Durch diese Vorauswahl erhält man die Menge  $TG^*$  der Elemente, welche nun eindeutig durch exakt einen zentralen Knoten  $v_i$  bestimmt sind.

### 6.3 Optimierte Auswahl der Teilgraphen

Als nächster Schritt gilt es die Untermenge  $TG_{opt}^* \subseteq TG^*$  zu finden, sodass die Knoten  $V_G(TG^*)$  so gut wie möglich alle Knoten  $V_G(G_{GSD})$  abdecken. Für die Auswahl muss gelten, dass die Knoten der Teilgraphen disjunkt sind  $V_G(TG_{v_i}) \cap V_G(TG_{v_j}) = \emptyset, \forall v_i \neq v_j$ .

Bildhaft lässt sich dieses Vorgehen am einfachsten anhand des „Similarity-Partitioning“ erklären. Dort gibt es die Menge  $KS^*$  an Kreisscheiben. Aus diesen muss eine Auswahl getroffen und in der Ebene platziert werden, sodass sie sich nicht überschneiden. Ähnliches passiert auch im Fall der GOA, hier wird aus der Menge an Teilgraphen  $TG^*$  eine Auswahl getroffen, sodass sich die Knotenmengen  $V_G(TG_{v_i})$  der ausgewählten Teilgraphen nicht überschneiden und die Knoten der Gesamtauswahl  $V_G(TG_{opt}^*)$  möglichst viele Knoten aus dem Ausgangsgraphen  $V_G(G_{GSD})$  ab-

decken. Während das „Similarity-Partitioning“ eine Zufallsauswahl verwendet, wird an dieser Stelle versucht, mit Hilfe eines einfachen genetischen Algorithmus die Auswahl von  $TG_{opt}^* \subseteq TG^*$  zu optimieren. Bevor dieser skizziert wird, soll jedoch darauf hingewiesen werden, dass dieses Problem im Wesentlichen dem Problem „Exact Cover“ entspricht. Beim „Exact Cover“-Problem müssen aus einer Menge Untermengen ausgewählt werden, sodass alle Elemente exakt einmal abgedeckt sind. Folgende Definition entstammt [46, S. 95]:

### **Definition 23 (Exact Cover)**

*INPUT: family  $S_j$  of subsets of a set  $\{u_i, i = 1, 2, \dots, t\}$*

*PROPERTY: There is a subfamily  $T_h \subseteq S_j$  such that the sets  $T_h$  are disjoint and  $\bigcup T_h = \bigcup S_j = \{u_i, i = 1, 2, \dots, t\}$ .*

Karp [46] zeigt in „Reducibility Among Combinatorial Problems“, dass dieses Problem NP-vollständig ist. Zur Lösung des „Exact Cover“-Problems existieren exakte Algorithmen, der prominenteste dürfte vermutlich der Backtracking-Algorithmus „DLX“ von Donald Knuth sein [47], welcher durch eine optimierte Datenstruktur namens „Dancing Links“ versucht, möglichst schnell Untermengen zur Gesamtmenge hinzuzufügen oder zu entfernen. Der „DLX“-Algorithmus ist bei der Auswahl von Teilgraphen jedoch nicht ohne Weiteres verwendbar, da nicht garantiert werden kann, dass es überhaupt Lösungen gibt, welche alle Knoten aus  $V_G(G_{GSD})$  abdecken.

Daher wird im Folgenden versucht, einen einfachen genetischen Algorithmus umzusetzen (siehe Abschnitt 7.2 auf Seite 57). Die Individuen sowie der Lösungsvektor  $s$  werden durch einen Vektor mit  $|TG^*|$  booleschen Elementen repräsentiert, für jedes Element  $s_i = 1$  wird der entsprechende Teilgraph ausgewählt. Ein Element aus  $s$  wird in Anlehnung an die Biologie auch „Locus“ genannt.

### **Definition 24 (Lösungsvektor der GA)**

$$TG^* = \{TG_1, TG_2, TG_3, \dots, TG_n\}$$

$$s = (1 \ 0 \ 0 \ \dots \ 1 \ 0)$$

$$TG_{sel}^* = \bigcup_{i=1}^{|TG^*|} \begin{cases} \{TG_i\} & \text{wenn } s_i = 1 \\ \emptyset & \text{wenn } s_i = 0 \end{cases}$$

### Definition 25 (Crossover-Operator)

*a, b und c sind Lösungsvektoren nach obiger Definition. c ist die Rekombination X() der Lösungsvektoren a und b nach folgender Vorschrift:*

$$c = X(a, b), c_i = \begin{cases} a_i & \text{wenn } i \text{ gerade} \\ b_i & \text{wenn } i \text{ ungerade} \end{cases}$$

Der Hauptgrund, aus dem dieser Crossover-Operator  $X()$  gegenüber einer Single-point-Variante gewählt worden ist, ist, dass die einzelnen Positionen eines Genoms, wegen der Art, wie die Teilgraphen aus den ursprünglichen Daten ermittelt werden, eine gewisse Lokalität besitzen können. Bildlich gesprochen soll vermieden werden, dass die linke Seite der Teilgraphenauswahl durch  $a$  und die rechte Seite durch  $b$  bestimmt wird. In der Literatur gibt es den sogenannten „Uniform Crossover“, welcher für jede Position zufällig entweder auf  $a$  oder  $b$  zurückgreift [17, S. 84]. Diese Zufallsauswahl wird von dem hier verwendeten Crossover-Operator nicht angewandt, damit verhält sich  $X()$  im Wesentlichen wie ein Singlepoint-Crossover, mit dem Unterschied, dass in dem vorliegenden Fall keine Zufallsposition bestimmt wird, sondern  $a$  und  $b$  immer zu gleichen Teilen in die Rekombination einfließen.

### Definition 26 (Mutations-Operator)

*Der Mutationsoperator  $M()$  und die Mutationsrate  $m$  sind wie folgt definiert:*

$$0 \leq m \leq |c|$$

$$c_{mut} = M(c, m), c_{mut_i} = \begin{cases} not(c_i) & \text{wenn } rand() < \frac{m}{|c|}, 0 \leq rand() < 1 \\ c_i & \text{ansonsten} \end{cases}$$

Damit entspricht die Mutationsrate  $m$  der Anzahl der durchschnittlichen Mutationen in einem Genom pro Anwendung des Mutations-Operators. Für den Fall, dass  $rand() < \frac{m}{|c|}$  zutrifft, wird der Locus an der betreffenden Stelle invertiert.

Die Auswahl der Individuen für die Fortpfianzung erfolgt über eine einfache „Tournament-Selection“ [17, S. 82 f.] und in jeder Runde wird durch „Elitism“ [17, S. 25] die beste Lösung in die nächste Runde mit übernommen. Das Zusammenspiel dieser Operatoren, gemeinsam mit der Kostenfunktion (siehe Definition 22 auf Seite 41), entspricht einem einfachen genetischen Algorithmus (siehe Algorithmus 5 auf Seite 58), mit einer zusätzlichen Forderung: Die Knotenmengen der Teilgraphen müssen disjunkt sein.

Zu diesem Zweck muss für jede Lösung, die durch Crossover und Mutation entstanden ist, in einem Bereinigungsschritt (siehe Algorithmus 2 auf Seite 47) sichergestellt werden, dass die Knotenmengen der ausgewählten Teilgraphen  $TG_{sel}^*$  sich nicht

überschneiden. Darüber hinaus muss auch sichergestellt sein, dass die durch Crossover und Mutation entstandenen Lösungen zumindest die Chance haben, bessere Lösungen zu generieren als ihre Vorfahren.

Dazu eine Anekdote, welche sich während der experimentellen Umsetzung des GOA-Algorithmus zugetragen hat: Die Implementierung hatte einen Fehler, der lange Zeit nicht auffiel. Der Fehler beruhte darauf, dass im Schritt der „Tournament-Selection“ die Reihenfolge der Population randomisiert wurde und gleichzeitig zwei Objektreferenzen auf die Population für die „Mutter“- und „Vater“-Seite existierten. Die Randomisierung änderte die Reihenfolge für beide Seiten dadurch in der exakt gleichen Weise, die Konsequenz daraus war, dass de facto kein Crossover mehr durchgeführt wurde, da für den Operator  $X(a, a)$  beide Parameter ident waren. Das Verblüffende war, dass trotzdem der Algorithmus brauchbare Ergebnisse lieferte. Wie kann das sein? Die Antwort ist, dass durch die Mutation, in Kombination mit dem Elitism, der genetische Algorithmus zu einem „Hill-Climbing“ degenerierte, aber für die entsprechenden Ausgangsdaten trotzdem noch akzeptabel funktionierte [75, S. 31 f.]. Nachdem das Problem identifiziert worden war, wurde anstelle einer Objektreferenz eine Objektkopie verwendet und der Crossover funktionierte wieder. Aber zur großen Überraschung war keine Verbesserung des Ergebnisses erkennbar. Der Grund dafür war, dass der Bereinigungsschritt, welcher die Disjunktheit der Knotenmengen garantieren muss, immer schlechtere Lösungen im Vergleich zu den Vorfahren hervorbrachte, indem sich überschneidende Knotenmengen ausgelassen wurden. Nachdem der Bereinigungsschritt so überarbeitet worden war, dass dieser versuchte, im Anschluss noch „Leerstellen“ mit entsprechenden Teilgraphen aufzufüllen, konvergierte der Algorithmus wesentlich schneller und zuverlässiger.

Nach einer definierten Anzahl von Runden mit einer bestimmten Populationsgröße endet der genetische Algorithmus, der beste Lösungsvektor bestimmt die Auswahl der Teilgraphen  $TG_{opt}^*$ .

## 6.4 Zuordnung der Residuen

Da die Knotenmenge  $V_G(TG_{opt}^*)$  möglicherweise nicht die gesamte Knotenmenge von  $V_G(G_{GSD})$  abdeckt, kann es notwendig sein, dass die Residuen  $v_r \in \{V_G(G_{GSD}) \setminus V_G(TG_{opt}^*)\}$  zugeordnet werden müssen. Für die Repräsentation der Zuordnung der Residuen wird ein  $|V_G(G_{GSD})|$  Elemente großer Residuenvektor verwendet. Jedes Element bestimmt die Zuordnung eines Knoten  $v_i \in V_G(G_{GSD})$  zu einem Zentrum. Per Definition wird der Wert  $-1$  verwendet, wenn ein Knoten nicht zugeordnet werden soll, weil dieser bereits in  $V_G(TG_{opt}^*)$  enthalten ist. Die

---

**Algorithm 2:** Bereinigungsschritt

---

**Data:** Lösungsvektor  $s$ , Menge der eindeutig bestimmten Teilgraphen  $TG^*$

**Result:** Bereinigter Lösungsvektor  $t$ , sodass Knotenmengen disjunkt sind

$V_G(TG_i) \cap V_G(TG_j) = \emptyset, \forall v_i \neq v_j$ . Anschließend Hinzufügen von weiteren Teilgraphen, sofern deren Knotenmengen disjunkt zur Lösung sind.

**begin**

```
N ← ∅  
rnd ← (1, ⋯, |s|)  
t ← (0, ⋯, 0)  
shuffle(rnd)  
forall  $i \in rnd$  do  
    if  $s_i = 1$  then  
        if  $V_G(TG_i) \cap N = \emptyset$  then  
             $t_i \leftarrow 1$   
             $N \leftarrow N \cup V_G(TG_i)$   
forall  $i \in rnd$  do  
    if  $V_G(TG_i) \cap N = \emptyset$  then  
         $t_i \leftarrow 1$   
         $N \leftarrow N \cup V_G(TG_i)$ 
```

---

Zuordnung wird nur zu einem bestimmenden zentralen Knoten der Teilmengen aus  $TG_{opt}^*$  vorgenommen. Zum Beispiel kann ein Residuenvektor wie folgt aussehen:

$$r = (-1 \ -1 \ \cdots \ v_{c_1} \ -1 \ v_{c_2} \ -1 \ \cdots \ v_{c_i})$$

Für die Zuordnung eines Residuums  $v_r$  wird in der Liste von dessen nächsten Nachbarn  $kNN(v_r)$  danach gesucht, ob ein zentraler Knoten  $v_i$  existiert, der einen Teilgraphen bestimmt; ist dies der Fall, kann  $v_r$  dem bestimmenden Knoten  $v_i$  zugeordnet werden. Für die Umsetzung ist es wichtig anzumerken, dass für die Suche des zentralen Knotens über die Liste  $kNN(v_r)$  diese mindestens so viele Element besitzen muss, dass mit Sicherheit mindestens ein zentraler Knoten darin enthalten ist. In der Praxis hat sich gezeigt, dass diese Zuordnung funktioniert, sobald die Auswahl der Teilgraphen durch den genetischen Algorithmus eine gewisse Qualität hat und der Suchbereich der nächsten Nachbarn nicht zu knapp bemessen ist.

## 6.5 Neuberechnung der zentralen Knoten

Durch die Zuordnung der Residuen werden die Knotenmengen der bestehenden Teilgraphen erweitert, was dazu führen kann, dass sich die Position des zentralen Knotens ändert. Daher muss der zentrale Knoten von allen Teilgraphen, denen Residuen zugeordnet worden sind, neu berechnet werden. Für die Berechnung wird ebenfalls auf den Algorithmus geodesic kNN zurückgegriffen.

Für jeden Teilgraphen werden neben der Adjazenzmatrix von  $G_{GSD}$  eine Knotenmenge bestehend aus ausschließlich den Knoten  $V_G(TG_{v_i}), TG_{v_i} \in TG_{res}^*$  und die Anzahl der Knoten  $k = |V_G(TG_{v_i})|$  übergeben. Bildlich lässt sich das wie folgt vorstellen. Mit der Adjazenzmatrix wird die Information über das gesamte Straßennetz übergeben. Die ausgewählten Knoten entsprechen denjenigen, die für die Suche nach nächsten Nachbarknoten als Kandidaten zur Verfügung stehen. Mit der Anzahl  $k$  wird festgelegt, dass für jeden Knoten  $k$ -nächste Nachbarn gesucht werden.

Da es insgesamt  $k$  Kandidaten gibt und nach den  $k$  nächsten Nachbarn gesucht wird, liefert der Algorithmus für alle  $kNN(v_j), v_j \in V_G(TG_{v_i})$  immer eine nach Distanz aufsteigend geordnete Liste mit  $k$  Elementen mit Tupeln der Form  $(d(v_j, v_k), v_k), v_k \in V_G(TG_{v_i})$ . Weil die Exzentrizität der Distanz zum weitest entfernten Knoten entspricht, lässt sich diese einfach durch die Betrachtung des letzten Elements der entsprechenden Liste ermitteln. Der Knoten mit der geringsten Exzentrizität wird als neues Zentrum des jeweiligen Teilgraphen bestimmt.

Dieser beschriebene Vorgang muss im ungünstigsten Fall für alle  $TG_{v_i} \in TG_{res}^*$  wiederholt werden, was dazu führt, dass  $|TG_{res}^*|$  Aufrufe des Algorithmus geodesic kNN notwendig sind. Um die Laufzeit des Gesamタルgorithmus dadurch nicht allzu sehr in die Höhe zu treiben, wurde die Erweiterung „earlyexit“ für den Algorithmus geodesic kNN entwickelt (siehe Abschnitt 7.1 auf Seite 52).

## 6.6 „Stehlen“ von Knoten

Dieser Schritt ist als eine Verbesserung der bestehenden Lösung zu verstehen. Der Grund, aus dem dieser Schritt vorgeschlagen wird, ist, dass die optimierte Auswahl der Teilgraphen durch den genetischen Algorithmus kompakte Teilgraphen favorisiert. Dadurch kann es bei der Zuordnung der Residuen vorkommen, dass peripherie Knoten einem Teilgraphen zugeordnet werden und durch die Neuberechnung des zentralen Knotens zunächst zentral gelegene Konten an die Peripherie rücken. Solche Knoten können dann unter Umständen bei einem anderen Teilgraphen besser aufgehoben sein.

Es wird der Frage nachgegangen, ob es für einen Knoten  $v_s$ , der einem Teilgra-

phen  $TG_{v_i}$  mit dem zentralen Knoten  $v_i$  zugeordnet ist, einen anderen Teilgraphen  $TG_{v_j} \in \{TG_{res}^* \setminus \{TG_{v_i}\}\}$  gibt, sodass der Abstand zu dessen zentralem Knoten  $d(v_s, v_j)$  kleiner ist als der Abstand zum aktuellen Zentrum  $d(v_s, v_i)$ .

Unter der Voraussetzung, dass der Teilgraph  $TG_{v_i}$  die Anonymitätsbedingung übererfüllt  $|V_G(TG_{v_i})| > a$ , kann dieser Knoten vom Teilgraphen  $TG_{v_j}$  übernommen werden.

In der Implementierung wird mit Hilfe der Liste der nächsten Nachbarn  $kNN(v_s)$  ähnlich wie im Schritt der Zuordnung der Residuen (siehe Abschnitt 6.4 auf Seite 46) nach den nächsten zentralen Knoten gesucht.

## 6.7 Berechnung der „minsum“-Zentren

Dies ist der letzte Schritt, der auf den Teilgraphen operiert. Während versucht wird, in dem vorherigen Schritt der Definition des zentralen Knotens entsprechend die Zentren der Teilgraphen zu bestimmen, wird in diesem Schritt das „minsum“-Zentrum – der Knoten, dessen Summe der Distanzen zu den anderen Knoten des Teilgraphen minimal ist – verwendet, damit die mittlere Abweichung eines Knotens zu seinem Zentrum minimiert wird.

Die Berechnung des „minsum“-Zentrums erfolgt im Wesentlichen analog zur Berechnung der zentralen Knoten (siehe Abschnitt 6.5 auf Seite 48), mit dem Unterschied, dass das Zentrum nicht durch die minimale Exzentrizität, sondern nach dem Minimum der Summe der Distanzen ausgewählt wird.

Die Menge der Teilgraphen mit den „minsum“-Zentren wird  $TG_{fin}^*$  genannt.

## 6.8 Erstellung der Territorien in der Ebene

Abschließend muss das Ergebnis der Teilgraphen  $TG_{fin}^*$  wieder in die Ebene transferiert werden. Jedem Knoten  $v \in V_G(TG_{fin}^*)$  ist in der Ebene eine Koordinate und somit eine entsprechende Voronoi-Region zugeordnet. Ein Territorium ist die Vereinigung aller Voronoi-Regionen, die jeweils einem Teilgraphen zugeordnet sind.

Die Umsetzung erfolgt wiederum mit Hilfe der PostGIS-Erweiterung als SQL-Anfrage. Für die Erstellung der Voronoi-Regionen wird eine Hilfstabelle „voronoi-helper“ verwendet. Mit Hilfe der Funktion „ST\_Collect()“ werden alle Koordinaten der Basisregionen als eine sogenannte „geometry collection“ abgebildet. Anschließend werden mit „ST\_VoronoiPolygons()“ die Voronoi-Regionen erstellt. Die set-returning-Funktion „ST\_Dump()“ erzeugt aus der Menge der Voronoi-Regionen eine Menge an Zeilen mit jeweils einer Voronoi-Region.

```
1 INSERT INTO voronoihelper SELECT (ST_Dump(ST_VoronoiPolygons(
    ST_Collect( ARRAY( SELECT point FROM baseregions))))).geom;
```

Die Tabelle „baseregions“ enthält neben einer Knoten-ID die Koordinate des Zentrums, die Zuordnung zu einem Teilgraphen und die Voronoi-Region. Dabei wird die Voronoi-Region nachträglich aus der Tabelle „voronoihelper“ übernommen. Dazu wird mit Hilfe des Nearest-Neighbour-Operators die Region gesucht, die der jeweiligen Zentrumskoordinate am nächsten ist und entsprechend die Spalte „voronoi“ aktualisiert.

```
1 UPDATE baseregions SET voronoi=(SELECT voronoi FROM voronoihelper
    ORDER BY voronoi <-> point LIMIT 1);
```

Die Territorien werden im Anschluss aus der Vereinigung der Voronoi-Regionen der Basisregionen, die demselben Teilgraphen über die Spalte „cluster\_id“ zugeordnet sind, erstellt. Die Vereinigung von den Voronoi-Regionen erfolgt mit der Funktion „ST\_Union()“, welche Geometrien vereinigt.

```
1 INSERT INTO territories (select cluster_id,NULL,ST_Union(b.voronoi)
    from baseregions as b group by cluster_id);
```

Damit ist das Verfahren der GOA abgeschlossen. Zur Implementierung ist anzumerken, dass zwei verschiedene Programmiersprachen verwendet werden. Immer wenn die Daten in der Ebene vorliegen, werden mit Hilfe von PostGIS die Daten manipuliert. Sobald die Daten als Graph zu bearbeiten sind, wird der Algorithmus in Python implementiert.

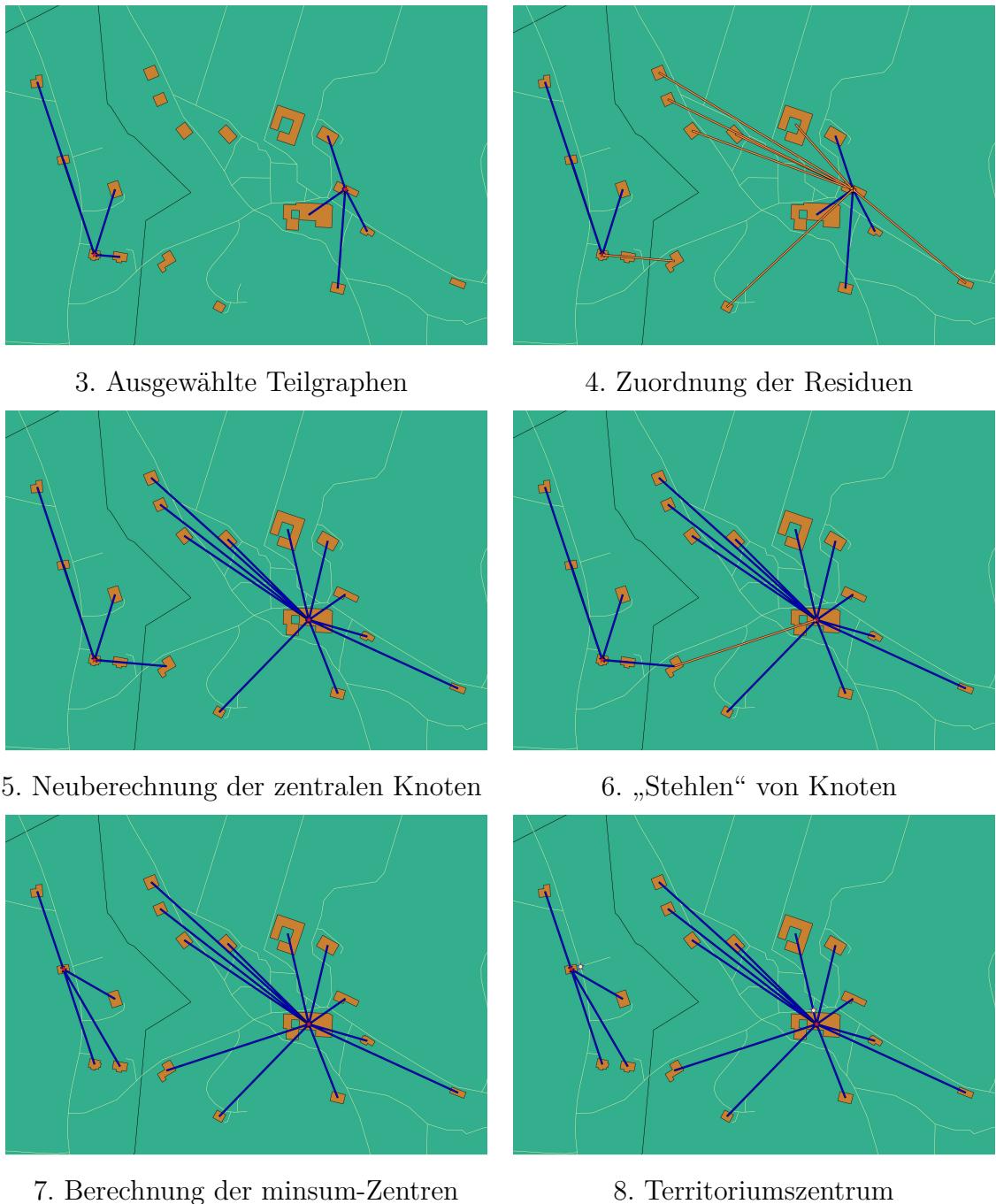


Abb. 14: GOA-Verfahren ab Teilgraphauswahl

## 7 Verwendete Algorithmen

### 7.1 Bestimmen der $k$ nächsten Nachbarn

Der Algorithmus, welcher zum Auffinden der  $k$  nächsten Nachbarn im Graphen verwendet wird, hat seinen Ursprung in einem Artikel mit dem Titel „Minimax-optimal semi-supervised regression on unknown manifolds“ [52]. Diese Arbeit beschäftigt sich, im Kontext des „Semi-Supervised Learning“, damit zu zeigen, dass unter gewissen Umständen ungekennzeichnete Daten, zusätzlich zu den gekennzeichneten Trainingsdaten, helfen können, bessere Regressionsfunktionen zu finden.

Obwohl die vorliegende Arbeit sich in einem anderen Fachbereich bewegt, ist der dort vorgeschlagene Algorithmus „geodesic kNN“ interessant, da er zwischen gekennzeichneten und ungekennzeichneten Daten unterscheidet (im Englischen „labeled data“ und „unlabeled data“).

Der GSD-Graph besteht ebenfalls aus zwei Klassen von Knoten: aus Kreuzungsknoten mit dem Gewicht 0 und Knoten, welche den projizierten Geoobjekten entsprechen, mit dem Gewicht 1. Damit lassen sich die Kreuzungsknoten direkt auf die ungekennzeichneten Daten umlegen, während die Knoten der projizierten Geoobjekte den gekennzeichneten Daten entsprechen.

Der Algorithmus „geodesic kNN“ basiert im Kern auf dem bekannten Dijkstr算法. Dieser ist zum Auffinden des kürzesten Weges zwischen einem Ausgangsknoten und allen übrigen Knoten in einem Graphen mit positiven Kantengewichten geeignet. Da von einem Knoten ausgegangen wird, wird das Problem im Englischen auch „single-source shortest path problem“ genannt [4, S. 95]. Der Algorithmus „geodesic kNN“ erweitert dieses Vorgehen, indem bei allen gekennzeichneten Knoten quasi gleichzeitig nach deren  $k$ -nächsten Nachbarknoten gesucht wird, wobei Nachbarknoten nur aus der Menge der gekennzeichneten Knoten  $\mathcal{L}$  entstammen können.

Für die Implementation der GOA erwies sich dieser Algorithmus als vielseitig einsetzbar, was ihn zu einer Art „Schweizer Taschenmesser“ für diese Arbeit macht. Folgende Problemstellungen können durch diesen Algorithmus abgedeckt werden:

- Das Auffinden aller  $k$  nächsten Nachbarn im GSD-Graphen, für jeden Knoten, der einem projizierten Geoobjekt entspricht  $V_G(G_{GSD})$ , was im Wesentlichen der Aufgabe gleichkommt, für die er ursprünglich vorgesehen war.
- Das Bestimmen des zentralen Knotens oder des „minsum“-Knotens eines Teilgraphen von  $TG \subseteq G_{GSD}$ . Dazu werden alle Knotengewichte, die nicht im

Teilgraphen vorkommen, auf 0 gesetzt  $w(v_k) = 0, v_k \in \{V(G_{GSD}) \setminus V_G(TG)\}$ . Die Exzentrizität des Knotens  $v, v \in V_G(TG)$  im Teilgraphen kann dann über das letzte Element der Liste  $kNN[v]$  bestimmt werden bzw. der „minsum“-Knoten kann durch Summierung der Distanzen in  $kNN[v]$  ermittelt werden. Für die Implementierung vorteilhaft ist, dass die Knotengewichte als ein Vektor repräsentiert werden und somit unabhängig von der Adjazenzmatrix des Gesamtgraphen manipuliert werden können. Die Adjazenzmatrix von  $G_{GSD}$  wird genau einmal im gesamten Verfahren erstellt und muss nicht weiter angepasst werden.

- Das Finden der Distanz des kürzesten Weges zwischen zwei Knoten  $v_1$  und  $v_2$  kann ebenfalls mit diesem Algorithmus umgesetzt werden, indem alle Knotengewichte, welche nicht zu  $v_1$  oder  $v_2$  gehören, auf 0 gesetzt werden  $w(v_k) = 0, v_k \in \{V(G_{GSD}) \setminus \{v_1, v_2\}\}$  und  $k = 2$  gewählt wird. Damit verhält sich der Algorithmus ähnlich wie ein angepasster Dijkstra-Algorithmus, der für die Suche nach dem kürzesten Pfad zwischen zwei Knoten von beiden Seiten gleichzeitig zu suchen beginnt. Diese Variante des Dijkstra-Algorithmus wird auch „Bidirektonaler Dijkstra-Algorithmus“ genannt [4, S. 112 f.].

### 7.1.1 Dijkstra-Algorithmus

Bevor nun im Detail auf den Algorithmus „geodesic kNN“ eingegangen wird, soll an dieser Stelle zum Einstieg der Dijkstra-Algorithmus vorgestellt werden, damit eine einfache Gegenüberstellung möglich wird [22, 38, 4].

Beginnend bei einem Startknoten wird zu allen anderen Knoten der kürzeste Weg gesucht. Bildhaft gesprochen ist die Idee des Algorithmus ausgehend vom Startknoten, dessen Nachbarknoten zu besuchen und von den Nachbarknoten weiter zu den Nachbarn der Nachbarknoten und so weiter. Dabei ist die Reihenfolge der Besuche immer durch die kürzeste Distanz bestimmt.

Die Knoten können während der Laufzeit einen von drei Zuständen annehmen: „unbesucht“, „besucht“ oder „abgeschlossen“. Der Algorithmus endet, wenn alle Knoten „abgeschlossen“ sind. In der Menge  $W$  befinden sich alle Knoten, die „besucht“ sind. Gestartet wird, indem der Menge  $W$  der Knoten  $v_s$  hinzugefügt wird. Danach wird immer aus der Menge  $W$  derjenige Knoten  $v$  betrachtet, für den gilt, dass  $d(v_s, v)$  ist minimal,  $\forall v \in W$ .

Da nur positive Distanzen erlaubt sind, wurde der kürzeste Weg  $d(v_s, v)$  gefunden und der Zustand von  $v$  ist „abgeschlossen“, somit muss  $v$  aus der Menge  $W$  entfernt werden.

In jedem Durchgang werden auch die Nachbarknoten  $w_i \in \text{neighbours}(v)$  betrachtet. Wenn ein Nachbarknoten  $w_i$  noch nicht besucht wurde, dann wird dieser in die Menge  $W$  mit aufgenommen. Des Weiteren wird überprüft, ob die zuvor gespeicherte Distanz von  $d(v_s, w_i)$  länger ist als der aktuelle Pfad, welcher über  $v$  führt. Wenn das der Fall ist, dann wird auch entsprechend der Vorgängerknoten  $\text{pred}(w_i) = v$  gesetzt.

---

**Algorithm 3:** Dijkstra-Algorithmus

---

**Data:** Ein ungerichteter, kantengewichteter Graph  $G = (V, E, w)$  und ein Startknoten  $v_s$ .  
**Result:** Für jeden Knoten  $v_i \in \{V \setminus \{v_s\}\}$  die Distanz  $\text{dist}[v_i] = d(v_s, v_i)$  sowie der Vorgängerknoten  $\text{pred}[v_i]$ , welcher angibt, über welchen Knoten der Pfad von  $v_s$  nach  $v_i$  zuletzt geführt hat.

```

begin
     $W \leftarrow \{v_s\}$ 
     $\text{dist}[v_s] \leftarrow 0$ 
     $\text{dist}[v_i] \leftarrow \infty, \forall v_i \in \{V \setminus \{v_s\}\}$ 
    while  $W \neq \emptyset$  do
         $v \leftarrow \text{pick-minimal}(W, \text{dist})$ 
         $W \leftarrow \{W \setminus \{v\}\}$ 
        forall  $w \in \text{neighbors}(v)$  do
            if  $\text{dist}[w] = \infty$  then
                 $W \leftarrow \{W \cup \{w\}\}$ 
            if  $\text{dist}[w] > \text{dist}[v] + d(w, v)$  then
                 $\text{dist}[w] = \text{dist}[v] + d(w, v)$ 
                 $\text{pred}[w] = v$ 

```

---

Die formale Beschreibung des Problems von Dijkstra selbst lautet: „Construct the tree of minimum total length between  $n$  nodes. (A tree is a graph with one and only one path between every two nodes.)“ [22, S. 1].

Nachdem alle Elemente aus  $W$  bearbeitet worden sind, kann der kürzeste Pfad zwischen dem Startknoten  $v_s$  und einem anderen Knoten  $v_o$  dadurch ermittelt werden, dass ausgehend von  $v_o$  die Referenzen der Vorgängerknoten  $\text{pred}[]$  herausgefunden werden, bis  $v_s$  erreicht wird. Der Pfad wird demnach rückwärts von  $v_o$  bis  $v_s$  durchlaufen.

Die Laufzeit des Dijkstra-Algorithmus ist mit  $\mathcal{O}(|V|^2)$  beschränkt, da alle Knoten

besucht werden müssen und dabei jedes Mal die Menge  $W$  nach dem Knoten mit der minimalen Distanz zum Startknoten durchsucht werden muss. Diese Laufzeit gilt, sofern ein naiver Ansatz zum Durchsuchen von  $W$  verwendet wird. Durch die Verwendung einer geeigneten Datenstruktur wie des „Fibonacci-Heaps“ lässt sich eine Schranke von  $\mathcal{O}(|E| + |V|\log|V|)$  erreichen [25]. Solch eine Datenstruktur wird auch in der Implementierung des Algorithmus „geodesic kNN“ verwendet.

### 7.1.2 „geodesic kNN“-Algorithmus

Dieser Algorithmus erwartet als Parameter den Graphen  $G$ , eine Menge an gekennzeichneten Knoten  $\mathcal{L} \subseteq V$  und die Anzahl der  $k$ -nächsten Nachbarn.

Die wesentlichen Unterschiede zwischen dem Dijkstra- und dem „geodesic kNN“-Algorithmus bestehen darin, dass nicht mit einem, sondern mit  $|\mathcal{L}|$  Startknoten begonnen wird. Des Weiteren wird auf die Konstruktion des Baums mit minimaler Gesamtlänge verzichtet und an dessen Stelle werden  $|V|$  Listen  $kNN[v]$  für jeden Knoten  $v$  mit dessen  $k$  nächsten Nachbarn aus der Menge  $\mathcal{L}$  und deren Distanz zu  $v$  erstellt.

Ansonsten verhält sich dieser Algorithmus ähnlich der Dijkstra-Variante unter Verwendung des Heaps. Zunächst werden alle Startknoten aus  $\mathcal{L}$  in die Priorityqueue  $Q$  eingetragen. Die Elemente in der Priorityqueue enthalten neben dem Startknoten  $v_s$  die „besuchten“ Knoten  $v_0$  und die Distanz  $d(v_s, v_0)$  als Tupel.

In jeder Runde wird das Element  $v_0$  mit der minimalen Distanz aus  $Q$  entnommen, damit kann die Distanz  $d(v_s, v_0)$  – unter der Voraussetzung von positiven Kantengewichten – auch nicht mehr unterboten werden. Die Zuweisung eines nächsten Nachbarknotens erfolgt, analog zum Finden des Pfades beim Dijkstra-Algorithmus, rückwärts und dem „besuchten“ Knoten  $v_0$  wird ein Startknoten aus  $v_s \in \mathcal{L}$  zugeordnet, sofern noch weniger als  $k$  nächste Nachbarn in der Liste  $kNN[v_0]$  eingetragen sind. Dieses Vorgehen erklärt auch, dass nur Knoten  $v_s \in \mathcal{L}$  als nächste Nachbarn gelten.

Ausgehend von  $v_0$  werden dann dessen Nachbarknoten  $v$  gemeinsam mit dem aktuellen Startknoten  $v_s$  in die Priorityqueue eingetragen, sofern die Liste  $kNN[v]$  nicht bereits  $k$  Elemente enthält.

Da ein Knoten  $v_i$  von mehreren Startknoten aus besucht werden kann, kommt den Mengen  $S_{v_i}$  noch eine besondere Bedeutung zu. Mit Hilfe von  $S_{v_i}$  wird mitprotokolliert, von welchen Startknoten aus ein Knoten  $v_i$  bereits besucht wurde. Im Falle, dass bereits ein Besuch von einem bestimmten Startknoten erfolgt ist, wird eine weitere Eintragung in die Priorityqueue unterbunden.

---

**Algorithm 4:** Geodesic  $k$  nearest labeled neighbors **with early exit**


---

**Data:** An undirected weighted graph  $G = (V, E, w)$  and a set of labeled vertices  $\mathcal{L} \subseteq V$ .

**Result:** For every  $v \in V$  a list  $kNN[v]$  with the  $k$  nearest labeled vertices to  $v$  and their distances. For  $v \notin \mathcal{L}$  the list  $kNN[v]$  is allowed to be incomplete.

**begin**

$Q \leftarrow PriorityQueue()$

$l \leftarrow 0$

**forall**  $v \in V$  **do**

$kNN[v] \leftarrow EmptyList()$

$S_v \leftarrow \emptyset$

**if**  $v \in \mathcal{L}$  **then**

$\lfloor$  insert( $Q, (v, v)$ , priority = 0)  $\rfloor$

**while**  $Q \neq \emptyset$  **and**  $l < |\mathcal{L}| \cdot k$  **do**

$(seed, v_0, dist) \leftarrow \text{pop-minimum}(Q)$

$S_{v_0} \leftarrow S_{v_0} \cup \{seed\}$

**if**  $length(kNN[v_0]) < k$  **then**

append ( $dist, seed$ ) to  $kNN[v_0]$

**if**  $v_0 \in \mathcal{L}$  **then**

$\lfloor$   $l \leftarrow l + 1$   $\rfloor$

**forall**  $v \in neighbors(v_0)$  **do**

**if**  $length(kNN[v]) < k$  **and**  $seed \notin S_v$  **then**

$\lfloor$  decrease-or-insert( $Q, (seed, v)$ , priority =  $dist + w(v_0, v)$ )  $\rfloor$

---

Die Laufzeit des Algorithmus „geodesic kNN“ beträgt  $\mathcal{O}(k|E| + N_p \log(|V|))$ . Dabei steht  $N_p$  für die Anzahl der pop-minimum-Operations auf dem Fibonacci-Heap, es gilt  $N_p = \min(|\mathcal{L}| |V|, k|E|)$ . Die Herleitung dazu findet sich in dem Supplementary zu „Minimax-optimal semi-supervised regression on unknown manifolds“ [53].

In der Praxis ist, für die in dieser Arbeit verwendeten Testdatensätze, mit einer Laufzeit im unteren einstelligen Sekundenbereich zu rechnen (siehe Abschnitt A.5.2 auf Seite 80).

Die grünen Zeilen des Pseudocodes entsprechen einer Modifikation des „geodesic kNN“-Algorithmus, welche für die vorliegende Arbeit entwickelt worden ist.

Die Intention ist, dass nur die nächsten Nachbarn von Geoobjekten von Interesse sind und auf Informationen über die nächsten Nachbarn von Kreuzungsknoten

verzichtet werden kann. (Zur Erinnerung: Kreuzungsknoten entsprechen den ungekennzeichneten Knoten.)

Im Rahmen dieser Arbeit ist es daher zulässig, den Algorithmus zu beenden, sobald für alle  $v \in \mathcal{L}$  die Liste  $kNN[v]$  mit jeweils  $k$  Elementen bestückt ist, daher auch der Name „early exit“. Diese Modifikation bringt in der Praxis – vor allem dann, wenn die Menge  $\mathcal{L}$  klein ist – wesentliche Geschwindigkeitsvorteile. Gerade wenn in einem einzelnen Teilgraphen lokal nach dem zentralen Knoten oder dem „minsum“-Knoten gesucht wird, ist die Menge  $\mathcal{L} = V_G(TG)$  nur etwa so mächtig wie die geforderte  $a$ -Anonymität plus etwaigen Residuen und „gestohlenen“ Knoten  $|\mathcal{L}| \approx a$ .

## 7.2 Genetische Algorithmen

„Genetische Algorithmen“ (kurz GAs) gehören zur großen Gruppe der „evolutionären Algorithmen“, welche globale Optimierungsverfahren darstellen, die von der Evolution natürlicher Organismen inspiriert sind [76].

Die Art der Repräsentation der Lösungen (und damit auch des Suchraums) als binäre Strings bzw. Strings von anderen elementaren Datentypen ist ein zentrales Unterscheidungsmerkmal von GAs zu anderen „evolutionären Algorithmen“ [74, S. 141].

Populär wurden GAs durch die Arbeit von John Holland, seinen KollegInnen und StudentInnen in den 70er-Jahren. Ihre Motivation war zum einen, adaptive Prozesse in der Natur abstrahieren und erklären zu können, und zum anderen, künstliche Softwaresysteme zu entwickeln, welche die aus der Natur entliehenen Mechanismen beibehalten [29, S. 1]. John Hollands Hauptwerk in diesem Zusammenhang trägt den Titel „Adaptation in Natural and Artificial Systems“ [39].

„Evolutionäre Algorithmen“ zeichnen sich unter anderem dadurch aus, dass nicht nach einer einzigen, sondern nach mehreren Optimierungslösungen gleichzeitig gesucht wird. In der Analogie zur Biologie wird deshalb von einer „Population“ gesprochen. Die einzelnen Lösungen (genannt Individuen) bringen ihr genetisches Material in den Genpool der Population ein. Die Population durchläuft mehrere Generationen. Individuen einer Generation können zu einem oder mehreren Nachfolgern der nächsten Generation rekombinieren, wobei in diesem Prozess auch Mutationen an den Nachfolgern auftreten können. In Anlehnung an die Biologie wird dabei von Eltern und Kindern oder im Englischen von „parents“ und „offsprings“ oder „children“ gesprochen. Jedes Individuum ist dabei mehr oder weniger gut an seine „Umwelt“, die gewünschten Randbedingungen, angepasst.

Die „Güte“ einer Lösung wird dabei mit einer Fitnessfunktion bewertet, welche aussagt, wie gut ein Individuum an seine „Umwelt“ angepasst ist [13, S. 96].

Eine weitere Eigenschaft von „evolutionären Algorithmen“ ist, dass Änderungen an der Population probabilistisch sind, wobei jedoch mit den Auswahlschritten dieser Zufall über die Generationen gelenkt wird. Des Weiteren werden Lösungen nicht durch Parameter, sondern in der codierten Form des Genoms dargestellt [30, S. 8].

Ein einfacher, generell gehaltener Algorithmus als Beispiel für „evolutionäre Algorithmen“ und somit auch für GAs ist in [12, S. 90 f.] zu finden.

---

**Algorithm 5:** Evolutionary computation algorithm

---

```
begin
    Initialize the population with random individuals
    Evaluate each individual
repeat
    Select parents
    Recombine pairs of parents
    Mutate the resulting offspring
    Evaluate new individuals
    Select individuals for the next generation
until a termination condition is satisfied
```

---

Die Evaluation eines Individuums bzw. einer Lösung erfolgt mit der sogenannten Fitnessfunktion. Wobei Zielfunktion die angemessenere Bezeichnung darstellt würden, wie die englische Literatur mit der Verwendung des Begriffs „objective function“ klarstellt.

Die Auswahl der Eltern-Paare kann auf unterschiedliche Weise erfolgen, folgende zwei Verfahren sind dabei gebräuchlich [74, S. 124 ff.]:

- „Fitness Proportionate Selection“, diese wird auch „Roulette Wheel Selection“ genannt. In diesem Verfahren wird die Gesamtfitness  $f_{total} = \sum f_i$  der Population ermittelt und die Auswahl der Individuen erfolgt mit einer Wahrscheinlichkeit von  $\frac{f_i}{f_{total}}$ . Wird das Bild des Rouletterads verwendet, so kann ein Individuum auf Grund seiner Fitness mehr oder weniger Positionen einnehmen, wobei die Wahrscheinlichkeit der Auswahl umso höher steigt, je höher der Anteil der eingenommenen Position ist.

- „Tournament Selection“, in diesem Verfahren werden zufällig zwei Individuen aus der Population entnommen, welche gegeneinander in einem „Turnier“ antreten müssen. Es gewinnt das Individuum mit der höheren Fitness  $f_i$ . Dieses Verfahren hat zur Konsequenz, dass das Individuum mit der höchsten Fitness garantiert ausgewählt wird. Das Individuum mit der geringsten Fitness kann nur im Falle, dass es gegen sich selbst antritt, gewinnen.

Die Rekombination der Eltern-Paare zu einem oder mehreren Kindern erfolgt durch den sogenannten Crossover-Operator, welcher Teile des Genoms des einen und des anderen Elternteils zu einem neuen Genom kombiniert.

- Der Singlepoint-Crossover-Operator entspricht der einfachsten Form der Rekombination. Es wird eine zufällige Position gewählt und beide Eltern-Genome werden an dieser Stelle geteilt. Die Rekombination besteht aus dem ersten Teil des einen und dem zweiten Teil des anderen Elternteils [17, vgl. S. 25 f.].
- Der  $n$ -Point- oder Multi-Point-Crossover-Operator ist eine Weiterentwicklung des zuvor genannten Verfahrens und die Genome werden an  $n$  Positionen mehrfach geteilt. Im Anschluss werden für die Genome der neuen Generation abwechselnd von der einen und der anderen Elternseite Fragmente von deren Genomen zusammengefügt [74, vgl. S. 148].
- Der Uniform-Crossover-Operator entscheidet für jede Position zufällig, ob das Gen von dem einen oder dem anderen Elternteil in die Rekombination einfließt [75, vgl. S. 17 f.].

Einzelne Loci des Genoms der Kind-Generation können sich durch Mutation auch ändern. Im Falle einer binären Repräsentation wird der boolesche Wert negiert, wobei Mutationen mit einer tendenziell geringen Wahrscheinlichkeit auftreten [17, S. 22 f.].

Die Individuen für die nächste Generation werden im Anschluss anhand ihrer Fitness ausgewählt. Bei der Auswahl kann die aktuell beste Lösung auch in die nächste Generation mit übernommen werden, diese Eigenschaft wird als „Elitism“ bezeichnet [17, S. 25].

In der Praxis lassen sich einfache genetische Algorithmen, eine passende Aufgabenstellung vorausgesetzt, schnell umsetzen. Im Wesentlichen müssen nur eine Fitnessfunktion und eine passende Repräsentation des Genoms gefunden werden. Es gilt jedoch darauf zu achten, dass alle Mechanismen tatsächlich auch ihren Beitrag leisten, sonst kann es passieren, dass ein GA zu einem Hill-Climbing-Algorithmus degeneriert (siehe Abschnitt 6.3 auf Seite 45).

## 8 Evaluierung des GOA-Verfahrens

In diesem Kapitel werden die Ergebnisse der Anwendung des GOA-Verfahrens auf die Testdatensätze interpretiert. Da die Datensätze mehrere hundert oder sogar tausende Geoobjekte beinhalten, ist nach der Anwendung des Verfahrens mit hunderten von Clustern zu rechnen. Im Anhang findet sich dazu eine vereinfachte Darstellung, um einen Eindruck der Einteilung in Territorien auf Gemeinde- bzw. Gemeindebezirksebene zu ermöglichen (siehe Abschnitt A.7 auf Seite 81).

Für die Betrachtung im Detail muss jedoch, auf Grund der hohen Anzahl von Clustern, auf eine komplette Darstellung verzichtet werden. Deshalb werden im Folgenden interessante Ausschnitte ausgewählt, die nur wenige Cluster beinhalten. Neben dieser „visuellen“ Analyse werden anschließend Kennzahlen entwickelt, um eine quantitative Bewertung des Verfahrens zu ermöglichen.

### 8.1 Visuelle Analyse

Bei der Betrachtung der Cluster im Ortszentrum (siehe Abb. 15 auf Seite 61) lässt sich über die Territorien zunächst aussagen, dass wie erwartet, wenn die Häuser dichter stehen, die Territorien kleiner sind und Bereiche mit einer geringeren Häuserdichte größere Territorien ergeben. Die Anzahl der Häuser pro Cluster variiert in diesem spezifischen Ausschnitt von der geforderten  $a$ -Anonymität bis zu  $(2 \cdot a - 1)$  Häusern. Da die Lösung auf der Minimierung der Distanzen zum Zentrum basiert, sind auch Cluster mit neun Häusern (zum Beispiel Cluster links oben) nachvollziehbar.

Etwas auffällig ist das Cluster am linken Rand in der vertikalen Mitte, hier scheinen die beiden H-förmigen Gebäude etwas weiter vom Clusterzentrum entfernt. Bei genauerer Betrachtung des Straßenverlaufs ist jedoch ersichtlich, dass diese beiden Gebäude sich in einer Art Sackgasse befinden, die nur von einer Seite aus erreicht werden kann. Vor Ort lässt sich dieses Ergebnis bestätigen. Es handelt sich bei den Gebäuden um ein Altersheim und ein betreubares Wohnheim, welche mit einem Auto tatsächlich nur von einer Seite erreichbar sind.

Dieser Ausschnitt lässt sich auch in die linke und rechte Hälfte teilen. Auf der linken Seite, mit den größeren Gebäuden, befindet sich der Ortskern mit öffentlichen Gebäuden, Geschäften und Wirtshäusern, auf der rechten Seite eine Wohnsiedlung. Die Cluster, die zwischen Ortskern und der Wohnsiedlung liegen, besitzen ein flächenmäßig größeres Territorium.

In diesem Ausschnitt sind alle Territorien zusammenhängend, wobei es denkbar wäre, dass das Gebäude am rechten Rand in der Mitte, weiter links platziert, dazu

führen könnte, dass das Nachbarterritorium in zwei Teile zerfallen würde.

Die Zuordnung der Geoobjekte zu den Clustern anhand des Straßenverlaufs ist schlüssig. Insgesamt ist das Ergebnis gut nachvollziehbar und intuitiv scheint es, unter den gegebenen Vorgaben, nicht einfach möglich, eine bessere Lösung zu finden.

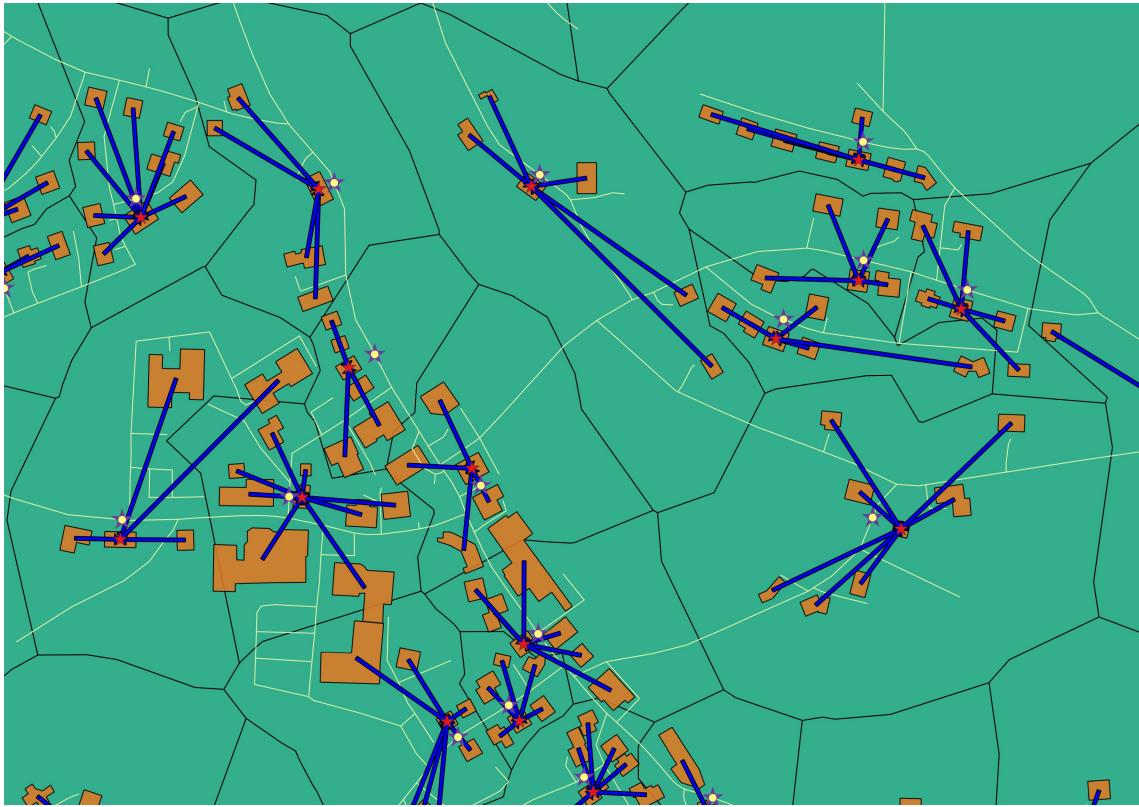


Abb. 15: Cluster im Ortszentrum

Für die Randbereiche an der Gemeindegrenze (siehe Abb. 16 auf Seite 62) lassen sich folgende Beobachtungen festhalten. Das Cluster in der Mitte des Ausschnitts, welches aus fünf Häusern besteht, ist ideal platziert. Die verteilten Gebäude werden unter Zuhilfenahme eines zusätzlichen Gebäudes der unten liegenden Siedlung zu einem Cluster zusammengefasst.

Die Siedlung selbst wird in vier Cluster von unterschiedlichen Größen von 5 bis 8 Gebäuden aufgeteilt. Rechnerisch würde sich in dieser Siedlung ein zusätzliches Cluster ausgehen, wobei ungewiss ist, ob unter der Minimierung der Knotendistanz dieses zu einem besseren Ergebnis führen würde. Die Einteilung der Siedlung wirkt, unter Berücksichtigung des Straßenverlaufs, stimmig.

Besonders auffällig ist das große Gebäude unten in der Mitte, dabei handelt es sich um einen Bahnhof, welcher von der Siedlung durch die Bahnstrecke getrennt ist. Aus diesem Grund ist es nicht möglich, den Bahnhof auf Straßen, die ausschließ-

lich auf dem Gemeindegebiet verlaufen, zu erreichen. Für ein gedachtes Bedarfsverkehrssystem stellt das ein Problem dar, welches in der vorliegenden Arbeit noch nicht behandelt worden ist. Für den wahrscheinlichen Fall, dass eine Person das Bedarfsverkehrssystem benutzt, um von zu Hause zum Bahnhof zu gelangen, würde die Fahrt außerhalb des Gemeindegebiets enden, obwohl der Bahnhof teilweise innerhalb der Administrationsgrenzen der Gemeinde liegt.

Zur Mitigation dieses Problems könnten entweder die Nachbargemeinden mit in die GOA einbezogen werden oder es könnte die Administrationsgrenze der jeweiligen Gemeinde um eine Art „Pufferzone“ erweitert werden.

In diesem Zusammenhang gilt es auch zu hinterfragen, ob es im Sinne des Datenschutzes zulässig wäre, einzelne ausgewählte Gebäude wie zum Beispiel Bahnhöfe, Schulen, Geschäfte usw. unaggregiert zu verwenden. Da diese Gebäude von vielen unterschiedlichen Menschen genutzt werden, ist ein Rückschluss auf eine bestimmte Person ohnehin nur schwer möglich. Diese Fragestellung ist aus einer juristischen Perspektive zu beantworten. Die technische Umsetzung durch eine Klasse von nicht aggregierbaren Geoobjekten ist in diesem Zusammenhang vermutlich das kleinere Problem.

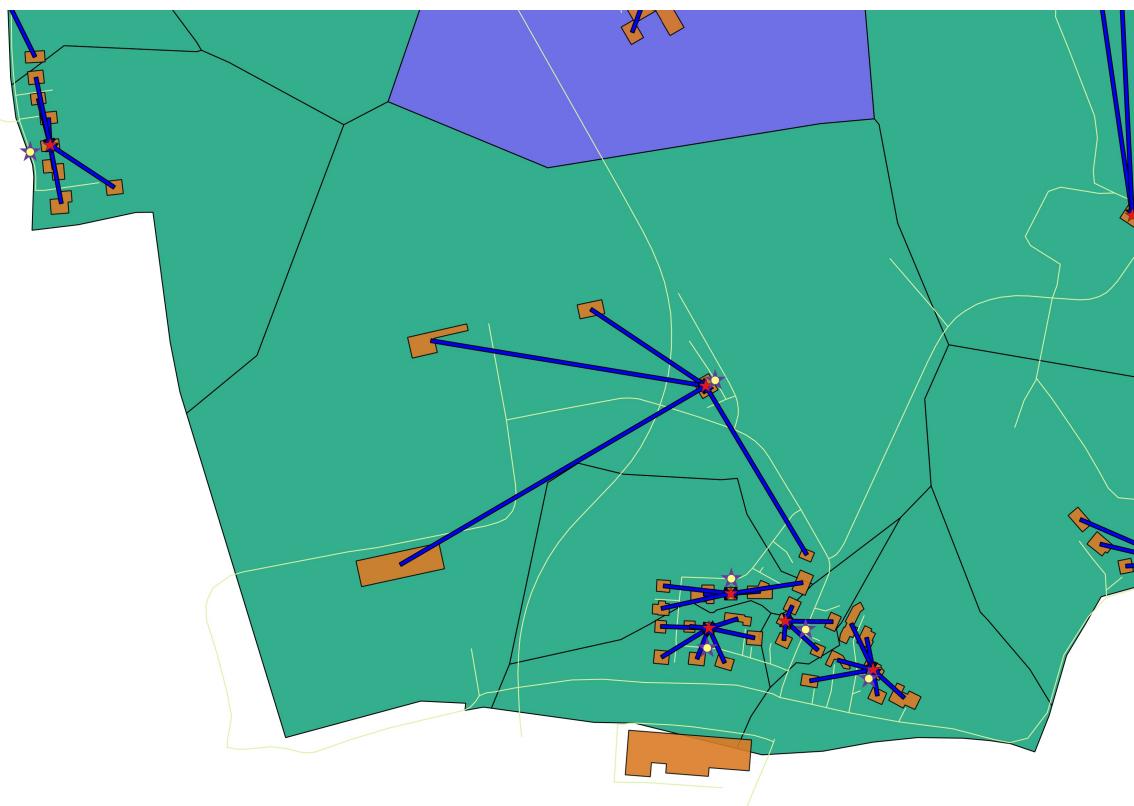


Abb. 16: Cluster an der Gemeindegrenze

Da die Clusterbildung auf Ebene des GSD-Graphen passiert und die Territorien in der Ebene gebildet werden müssen, kann es sein, dass je nach Straßenverlauf ein Territorium in mehrere Teile zerfällt (siehe Abb. 17 auf Seite 64).

Im einfachsten Fall wird durch Geoobjekte von benachbarten Clustern und deren zugeordneten Voronoi-Regionen ein Gebiet beansprucht, welches die Linie zwischen einem Knoten und dessen zugeordnetem Clusterzentrum schneidet. Unter Umständen kann dieses Gebiet auch so gestaltet sein, dass eine Voronoi-Region komplett von dem ihm zugeordneten Territorium abgetrennt wird.

Bei einem entsprechenden Straßenverlauf (zum Beispiel einer S-Kurve) können solche Abtrennungen auch gleichzeitig benachbarte Territorien treffen.

Im Extremfall ist es sogar möglich, dass zwei benachbarte Cluster sich kreuzen bzw. ineinandergreifen, so Gebiete innerhalb eines Territoriums von einem anderen Cluster beansprucht werden und es daher zu einer Trennung des Territoriums kommt (siehe Abb. 25 auf Seite 84).

Neben dem Zusammenspiel von benachbarten Clustern gibt es allerdings noch eine weitere Ursache, warum Territorien nicht zusammenhängend sein müssen. Durch die Beschränkung der Gebiete auf die Administrationsgrenze der Gemeinde bzw. des Gemeindebezirks kann es sein, dass Territorien geteilt werden. In der Grafik ist dies im Falle des rosa gefärbten Territoriums ersichtlich. Ähnlich dem Problem des Bahnhofs an der Gemeindegrenze muss auch für diesen Fall eine Mitigationsstrategie entwickelt werden.

In dem vorliegenden Fall ist auch noch ein „wurmartiges“ Gebäude zu sehen, welches keinem Cluster zugeordnet ist. Dabei handelt es sich um den Kindergarten „Schweizerspende“, welcher sich im nordwestlichen Teil des Auer-Welsbach-Parks im 15. Wiener Gemeindebezirk befindet. Dieser ist ebenfalls, wie das zuvor genannte Beispiel des Bahnhofs, nicht über reguläre Straßen innerhalb des Gemeindebezirks zu erreichen.

Eine weitere beachtenswerte Struktur ist die „Kleingartensiedlung auf der Schmelz“. Dabei handelt es sich um ein Areal, in dem viele kleine Häuser nur durch schmale Fußwege zu erreichen sind. Befahrbare Straßen gibt es nur rund um das Gebiet und in Form von zwei Strecken, welche das Areal in der Mitte durchkreuzen (siehe Abb. 18 auf Seite 64).

Da alle Geoobjekte im GSD-Graphen auf das Straßennetz projiziert werden müssen, liegen viele dieser Punkte dicht an dicht auf den Straßen in einer Linie. Die zugeordneten Voronoi-Regionen ergeben somit längliche Territorien, die normal zum Straßenverlauf stehen. Die gelb-violetten Sterne kennzeichnen in der Grafik die Zentren der Territorien.

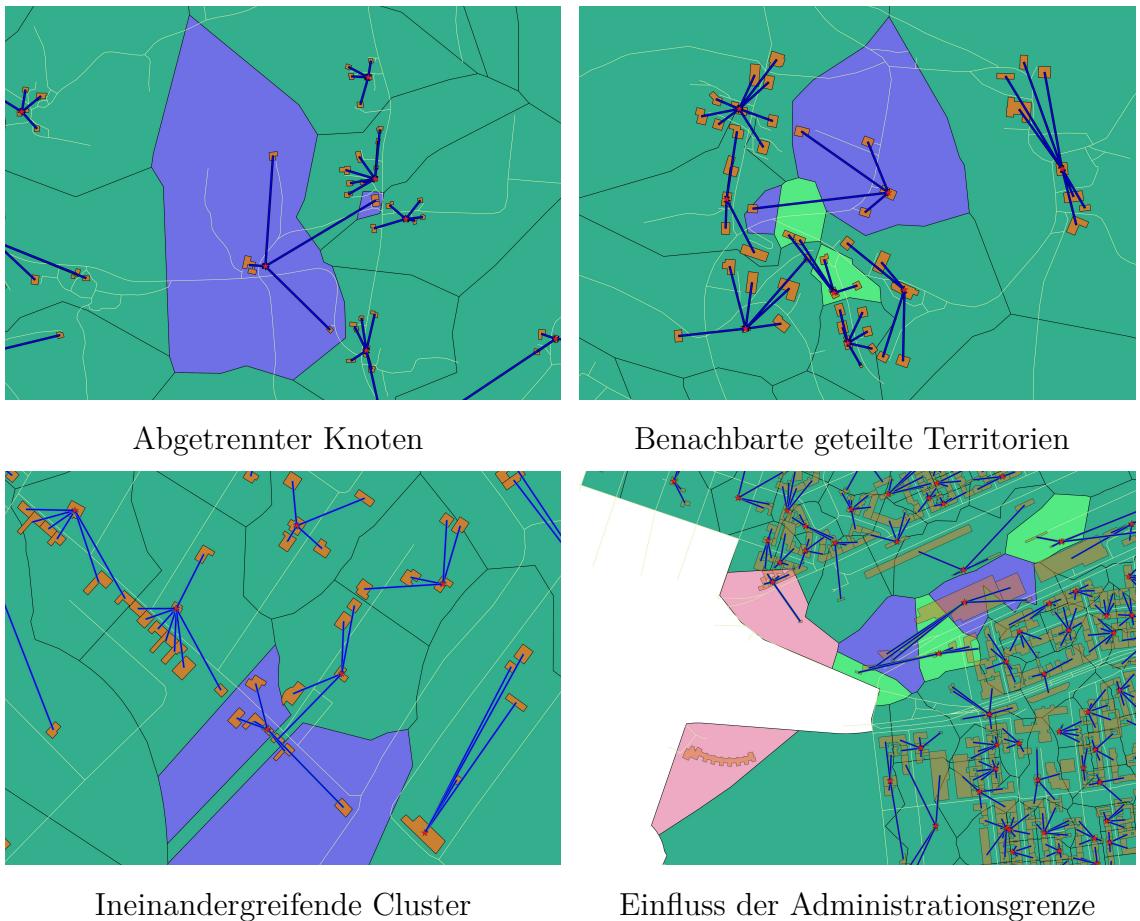


Abb. 17: Nicht zusammenhängende Territorien



Abb. 18: Kleingartensiedlung auf der Schmelz

Es lässt sich somit festhalten, dass die Territorien folgende Eigenschaften aufweisen:

- Territorien müssen nicht konvex sein, da sie sich aus mehreren Voronoi-Regionen zusammensetzen.
- Territorien müssen nicht zwingend zusammenhängend sein.
- Jedem Territorium ist ein Zentrum zugeordnet, welches einem Punkt im Straßennetz entspricht.

Für den Anwendungsfall der Anonymisierung von Fahrgastdaten ist es nicht weiter hinderlich, wenn Territorien nicht zusammenhängend sind, da sich die erhobenen Daten auf das Straßennetz beziehen. Die Territorien sind in dieser Anwendung „nur“ dazu da, eine Koordinate im Bereich eines Territoriums auf einen Punkt auf dem Straßennetz zu transferieren.

Je nach Anwendung kann die Forderung nach zusammenhängenden Territorien notwendig sein oder nicht. Zum Beispiel wird in [73, S. 46] die Verwendung von kNN-Verfahren zur Gebietsaufteilung mit Hilfe des Straßennetzes ausgeschlossen, weil diese keine zusammenhängenden Territorien garantieren können.

## 8.2 Quantitative Betrachtung

Die Interpretation der Ergebnisse erfolgt auf drei verschiedenen Ebenen. Sowohl die sich ergebenden Clustergrößen, die Knotenentferungen zu dem zugeordneten Clusterzentrum als auch die Anzahl der nicht zusammenhängenden Territorien werden in diesem Kapitel betrachtet.

In der Tabelle (siehe Tab. 3 auf Seite 66) findet sich eine Auflistung der entsprechenden Kennzahlen der Anwendung des GOA-Verfahrens auf die drei Testdatensätze. Die Standardabweichung von Clustergröße und Knotenentfernung wurde in dieser Tabelle bewusst ausgelassen, weil, wie später gezeigt wird, die entsprechenden Verteilungen allesamt rechtsschief sind und damit die Standardabweichung für den vorliegenden Fall nicht einfach interpretierbar ist. Anstelle der Standardabweichung wird daher in der Tabelle das jeweilige 50%--, 95%- und 99%-Perzentil angegeben. Des Weiteren wurde die minimale Clustergröße ausgelassen, weil diese im Regelfall durch die geforderte  $a$ -Anonymität determiniert ist und in dieser Arbeit immer  $a = 5$  entspricht. Auch ausgelassen wurde die minimale Knotenentfernung, da diese für den zentralen Knoten immer 0 entspricht. Bei den nicht zusammenhängenden Territorien sind im Falle des Testdatensatzes „Wien 15ter“ in Klammern diejenigen

Territorien angeführt, welche durch Verschneidung mit den administrativen Grenzen des Gemeindebezirkes entstehen.

	Neukirchen	Purbach	Wien 15ter
Anz. Geoobjekte	791	622	3151
Anz. Cluster	121	107	537
Max. Clustergröße	13	10	9
$\varnothing$ Clustergröße	6.54	5.81	5.87
Clustergröße 50%-Perzentil	6.00	5.00	6.00
Clustergröße 95%-Perzentil	10.00	8.00	8.00
Clustergröße 99%-Perzentil	11.60	9.00	9.00
Max. Knotenentfernung [m]	798.75	582.29	399.90
$\varnothing$ Knotenentfernung [m]	101.34	47.37	21.19
Knotenentfernung 50%-Perzentil [m]	69.62	22.82	11.91
Knotenentfernung 95%-Perzentil [m]	335.37	176.30	75.59
Knotenentfernung 99%-Perzentil [m]	604.33	304.53	133.53
nicht zshg. Territorien	8	3	6 (+3)

Tab. 3: Kennzahlen der Aggregation der Testdatensätze

An dieser Stelle ist möglicherweise eine kurze Erinnerung an die Eigenschaften der den Testdaten zugrundeliegenden Gemeinden bzw. des Gemeindebezirks angebracht. Die Gemeinde Neukirchen besteht aus vielen Ortschaften, die über das ganze Gemeindegebiet verteilt sind. Die Gemeinde Purbach hat ein relativ kompaktes Zentrum, welches sich zum Rand der Gemeindegrenze hin ausdünnt. Der 15. Wiener Gemeindebezirk besteht im Wesentlichen aus im Raster angeordneten Wohnhäusern, hat aber auch andere weitläufigere Strukturen wie Parks und eine Kleingartensiedlung.

Betreffend die Clustergröße lässt sich feststellen, dass das GOA-Verfahren mit einer geforderten 5-Anonymität für „Purbach“ und „Wien 15ter“ eine durchschnittliche Clustergröße von etwas unter 6 Geoobjekten pro Cluster liefert, während für die „weniger kompakte“ Gemeinde Neukirchen die durchschnittliche Clustergröße 6.54 beträgt. Auch die maximale Clustergröße von 13 Geoobjekten im Falle dieser Gemeinde ist in diesem Zusammenhang erwähnenswert.

Wird die Verteilung der Clustergröße (siehe Abb. 19 auf Seite 67) betrachtet, so lässt sich festhalten, dass 50% der Cluster 6 oder weniger Geoobjekte im Falle von „Neukirchen“ und „Wien 15ter“ besitzen bzw. das Minimum von 5 Geoobjekten im Fall von „Purbach“. Während 99% der Geoobjekte für die Datensätze „Purbach“

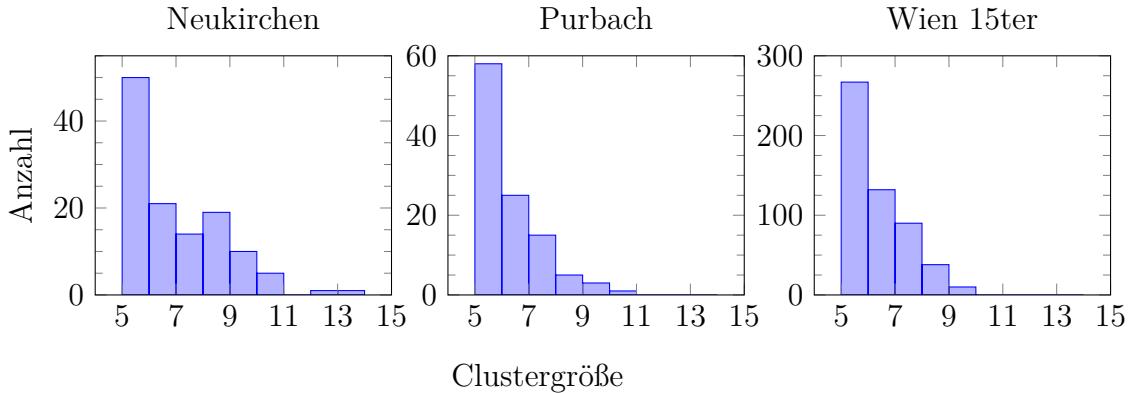


Abb. 19: Histogramm Clustergröße

und „Wien15ter“ zu Clustern mit einer Größe von 9 oder weniger zusammengefasst sind, beträgt das 99%-Perzentil für „Neukirchen“ 11.60 Geoobjekte. Wie später im Kapitel der Verbesserungsvorschläge (siehe Abschnitt 9.2.2 auf Seite 71) angeführt ist diese Beobachtung insofern interessant, da sich für Cluster, welche die zweifache oder höhere Anzahl von  $a$  Elementen beinhalten, die Frage aufdrängt, ob solche Cluster nicht in zwei Cluster aufgeteilt werden können.

Zur Knotenentfernung lässt sich festhalten, dass in den Testdatensätzen immer wieder „isolierte“ Geoobjekte existieren, die etwas weiter entfernt sind. Dadurch lässt sich die maximale Knotenentfernung von knapp 400 bis knapp 800 Metern erklären. Die durchschnittliche Knotenentfernung entspricht der Charakteristik der Gemeinden und ist umso höher, je weniger „kompakt“ die Gemeinde ist. Die Verteilungen der Knotenentfernungen (siehe Abb. 20 auf Seite 68) liefern keine großen Überraschungen. Im Falle von „Wien 15ter“ zeigt das 99%-Perzentil von 133.53 m im Vergleich zur max. Knotenentfernung von 399.90 m, dass isolierte Geoobjekte im städtischen Bereich eher die Ausnahme als die Regel sind. Dahingegen zeigt die Betrachtung der 95%- und 99%-Perzentile im Falle von „Neukirchen“, dass 5% der Geobjekte über 335.37 m und 1% über 604.33 m von ihrem jeweiligen Clusterzentrum entfernt sind.

Betreffend die nicht zusammenhängenden Territorien kann gesagt werden, dass die Anzahl dieser sich im einstelligen Prozentbereich bewegt. Damit sind nicht zusammenhängende Territorien eher die Ausnahme als die Regel. Diese Erkenntnis mündet in einem Verbesserungsvorschlag (siehe Abschnitt 9.2.3 auf Seite 72), denn falls gewünscht könnten diese Territorien aufgelöst und die entsprechenden Geobjekte als Residuen zu den Nachbarterritorien zugeordnet werden. Das würde einem Trade-off zwischen der Forderung nach zusammenhängenden Territorien und der Knotenentfernung entsprechen.

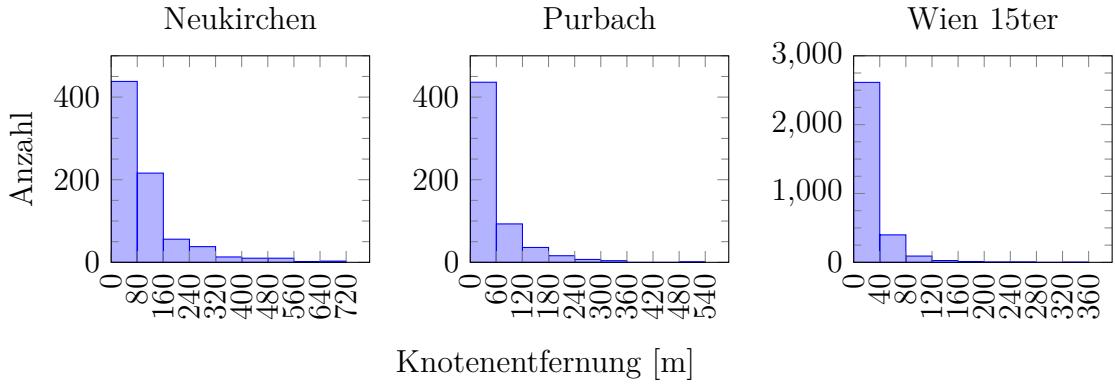


Abb. 20: Histogramm Knotenentfernung

### 8.3 Kritische Betrachtung der Häuser als Haushalte

Grundsätzlich gilt es anzumerken, dass Haushalte auf Grund der Existenz eines Gebäudes mit einer Hausnummer angenommen werden. In der Praxis ist jedoch davon auszugehen, dass durch Leerstand oder Zweitwohnsitze diese Annahme nicht immer zulässig ist.

Daher sollten vor dem Produktiveinsatz des vorgestellten Verfahrens noch Strategien entwickelt werden, wie überprüft werden kann, ob tatsächlich eine Aggregation über die geforderte Anzahl an Haushalten geschieht.

Ein möglicher Ansatzpunkt würden statistische Daten zur Bevölkerungsdichte liefern, welche in einem entsprechend engmaschigen Raster zur Verfügung stehen.

## 9 Verbesserungspotential

Das hier vorgestellte GOA-Verfahren ist vermutlich nur eine von vielen Möglichkeiten, um die Aufgabenstellung der Aggregation von Geoobjekten zu erfüllen. In diesem Kapitel sollen Ideen festgehalten werden, wie das vorliegende Verfahren zum einen beschleunigt und zum anderen auch qualitativ weiterentwickelt werden kann. Jede dieser Ideen soll dabei kurz dahingehend bewertet werden, wie relevant sie für die Praxis ist und wie einfach sie umzusetzen wäre.

Weiters wird der Frage nachgegangen, wie das Verfahren aussehen könnte, wenn mit dem Wissensstand nach Abschluss dieser Arbeit ein alternatives Nachfolgeverfahren für die GOA entworfen würde.

## 9.1 Laufzeitverbesserungen

### 9.1.1 Eliminierung der „unechten“ Kreuzungsknoten

Wie bereits erwähnt kommt es beim GSD-Graphen-Export dazu, dass unechte Kreuzungsknoten mit dem Grad  $\deg = 2$  entstehen. Die Ursache hierfür ist, dass sich die in der Datenbank hinterlegten Straßenzüge an einem Punkt berühren können und somit eine Fortsetzung, jedoch keine echte Kreuzung bilden. Laut den Eckdaten der Testdatensätze (siehe Tab. 2 auf Seite 36) hat jeder dieser GSD-Graphen mehrere hundert solcher unechten Kreuzungsknoten. Diese könnten ohne Informationsverlust einfach entfernt werden.

Dieser Bereinigungsschritt ist vermutlich einfach umzusetzen und würde die Anwendung des „geodesic kNN“-Algorithmus beschleunigen. Hier stellt sich aber die Frage, ob dieser zusätzliche Schritt sich so effizient implementieren lässt, dass auch die gesamte Laufzeit des GOA-Algorithmus davon profitieren kann. Für die praktische Anwendung ist keine wesentliche Verbesserung zu erwarten, weil die Anwendung des Algorithmus „geodesic kNN“ nur einen kleinen Prozentsatz der Gesamtaufzeit ausmacht. Von einem „ästhetischen“ Gesichtspunkt ist solch eine „Aufräumarbeit“ durchaus zu begrüßen.

### 9.1.2 Eliminierung aller Kreuzungsknoten

Der zuvor genannte Gedanke kann auch noch fortgesetzt werden und es kann der Frage nachgegangen werden, ob der GSD-Graph nicht in einen Graphen transferiert werden kann, welcher keine Kreuzungsknoten beinhaltet. Da die relevante Information darin besteht, wie weit ein Geoobjekt-Knoten von seinen Nachbar-Geoobjekt-Knoten entfernt ist, könnte diese Information auch durch eine direkte Kante repräsentiert werden.

Die Umsetzung dürfte einfach zu implementieren sein, es gilt jedoch zu bedenken, dass die Auswahl des „geodesic kNN“-Algorithmus vor allem wegen seiner Unterscheidung von gekennzeichneten und ungekennzeichneten Knoten erfolgt ist. Mit dieser vorgeschlagenen Änderung wäre diese Unterscheidung nicht mehr notwendig. Der „geodesic kNN“-Algorithmus könnte jedoch ohne Änderungen weiterverwendet werden, womit eine einfache Umsetzbarkeit möglich ist. Auf Grund des Laufzeitverhaltens des oben genannten Algorithmus ist jedoch nicht auszuschließen, dass diese Änderung sich negativ auf die Laufzeit auswirken kann, da ein Graph ohne Kreuzungsknoten eine höhere Dichte aufweisen muss. Für die Praxis wird dieser Vorschlag als irrelevant bewertet.

Allerdings kann für die Entwicklung eines alternativen Verfahrens diese Vorgehensweise mitgedacht werden (siehe Abschnitt 9.3 auf Seite 73).

### 9.1.3 Verwendung von mehreren Threads

Das größte Optimierungspotential bietet der Schritt „optimierte Auswahl der Teilgraphen“ (siehe Abschnitt A.5.2 auf Seite 80), welcher als ein genetischer Algorithmus implementiert ist. GAs sind auf Grund der Vorgänge innerhalb der Population prädestiniert dazu, parallel implementiert zu werden. Hier ist zu beachten, dass gewisse Vorgänge wie die Auswahl der Eltern-Individuen sich eventuell nicht parallelisieren lassen. Andere wie Crossover, Mutation, Berechnen der Fitness oder etwaige Bereinigungsschritte lassen sich jedoch gut parallelisieren. Unter der Voraussetzung, dass die Auswahl der Eltern nicht parallel implementiert werden kann, ist davon auszugehen, dass nach der Berechnung jeder Generation eine Synchronisation zwischen den einzelnen Threads erforderlich ist.

Im konkreten Fall ist die Standardeinstellung eine Populationsgröße von 100 Individuen. Die Verwendung von Threads wird vermutlich Geschwindigkeitsvorteile bringen. Jedoch ist zu erwarten, dass nur deutlich weniger Threads als die Populationsgröße zu einer Verbesserung der Laufzeit führen.

Die Umsetzung erfordert, sich über die Synchronisation und das Aufteilen der Eltern-Paare auf verschiedene Threads Gedanken zu machen.

In der Praxis ist mit einer deutlichen Verbesserung der Laufzeit zu rechnen, daher wird diese Verbesserungsmöglichkeit als sinnvoll eingestuft.

### 9.1.4 Just-in-Time-Compiler

Die Verwendung eines Just-in-Time-Compilers, wie pypy3, kann unter Umständen einen Geschwindigkeitsvorteil bringen. Sobald die zusätzlichen Pythonpakete wie numpy, scipy usw. installiert sind (siehe Abschnitt A.1 auf Seite 77), kann das Programm „runGOA.py“ ohne weitere Modifikation mit pypy3 gestartet werden.

Allerdings sollen an dieser Stelle die Erwartungen etwas gedrosselt werden. Je nach Testdatensatz haben erste Experimente ergeben, dass nur der größte Testdatensatz „Wien 15ter“ eine moderate Beschleunigung mit einer Gesamlaufzeit von 175.294 s (versus 201.039 s) erfährt und die anderen tendenziell langsamer ausgeführt werden: „Neukirchen“ mit 31.982 s (19.180 s) und „Purbach“ mit 25.386 s (14.970 s).

Für die Praxis ist diese Möglichkeit daher nur mit einem entsprechenden Profiling interessant. Dadurch erhöht sich der Aufwand für die Umsetzung.

## 9.2 Qualitative Verbesserungen

### 9.2.1 Berücksichtigung von Straßen außerhalb des Gemeindegebiets

Wie das Beispiel vom Bahnhof, welcher nicht über Straßen, die innerhalb des Gemeindegebiets verlaufen, erreichbar ist, zeigt, müssen auch Straßen außerhalb des Gemeindegebiets berücksichtigt werden. Besonders im Hinblick auf den Produktiveinsatz müssen hier Strategien entwickelt werden, die mit solchen Fällen umgehen können. Die Beschränkung auf das Gemeindegebiet erfolgt in einem dem GOA-Verfahren vorgelagerten Schritt, welcher die Ausgangsdaten entsprechend auf Geobjekte und Straßenverläufe reduziert, die sich zumindest teilweise innerhalb der Administrationsgrenzen der Gemeinde befinden. Dieser Schritt müsste nun dahingehend modifiziert werden, dass für den Fall von Straßenverläufen auch eine „Pufferzone“ außerhalb der Gemeindegrenzen mit einbezogen wird. Großzügiger gedacht könnten die gesamten Straßenverläufe der Nachbargemeinden mitberücksichtigt werden.

Die Umsetzung ist eine relativ einfache Anpassung der entsprechenden SQL-Abfragen (siehe Abschnitt A.3 auf Seite 78) durch Hinzufügen einer Pufferzone um das Polygon, welches das Gemeindegebiet beschreibt, mit der PostGIS-Funktion „ST\_Buffer()“. Für die Praxis ist diese Verbesserung unbedingt notwendig, vor allem wenn eine händische Nachbearbeitung des GOA-Ergebnisses ausgeschlossen wird.

### 9.2.2 Aufteilen von großen Clustern

Cluster, die doppelt so viele oder mehr Geoobjekte als die gewünschte  $a$ -Anonymität aufweisen, werfen die Frage auf, ob sich durch ein einfaches Aufteilen in zwei Cluster eine bessere Lösung, im Sinne der Minimierung der Gesamt-Knotendistanzen, erreichen lässt.

In der Abbildung (siehe Abb. 21 auf Seite 72) ist ein Cluster mit 13 Geoobjekten zu sehen. Die Aufteilung in zwei Cluster, bestehend aus den Geoobjekten innerhalb der Ellipse und den restlichen Objekten, würde zu einer besseren Lösung führen.

Durch eine einfache rekursive Anwendung des GOA-Verfahrens auf die 13 Geoobjekte würde dies jedoch nicht funktionieren, da dieses versucht, zunächst Cluster von den 5 nächsten Nachbarknoten im GSD-Graphen zu „platzieren“. In diesem konkreten Beispiel würde der Knoten, der dem größten Gebäude in der Mitte zugeordnet ist, für jeden anderen Knoten einer der 5 nächsten Nachbarknoten sein. Daher würde das GOA-Verfahren in seiner jetzigen Form nur ein Zentrum erstellen.

Zur Umsetzung müssten für die Teilung von großen Clustern daher weitere Strategien entwickelt werden. Die Relevanz in der Praxis ist je nach Anwendungsfall



Abb. 21: Cluster mit 13 Geoobjekten

zu bewerten. Zum Beispiel ist diese Änderung für die Datensätze „Purbach“ und „Wien 15ter“ irrelevant, da in diesen Fällen 99% oder mehr der Cluster kleiner oder gleich ( $2a - 1$ ) sind. Für Datensätze wie „Neukirchen“ könnten die Gesamt-Knotendistanzen zu den Zentren moderat verbessert werden.

### 9.2.3 Eliminieren von nicht zusammenhängenden Territorien

Für den Fall, dass eine Anwendung zusammenhängende Territorien erfordert, wäre eine einfache Strategie, Cluster, welche Territorien erzeugen, die nicht zusammenhängen, aufzulösen und die entsprechenden Knoten als Residuen den Zentren der Nachbarcluster zuzuordnen. Da nicht zusammenhängende Territorien die Ausnahme darstellen, wäre solch ein Vorgehen praktikabel.

Es ist jedoch anzumerken, dass die Forderung nach zusammenhängenden Territorien ein Trade-off zur Minimierung der Gesamt-Knotendistanzen darstellt und sich das Ergebnis im Hinblick auf die Distanzen mit dem Auflösen von Clustern wahrscheinlich verschlechtern wird.

Die Umsetzung wäre relativ einfach, hat jedoch auch eine Implikation betreffend die Implementierung. Das GOA-Verfahren operiert derzeit ausschließlich auf dem GSD-Graphen. Ob ein Territorium zusammenhängend ist oder nicht, kann nur in der Ebene festgestellt werden. Diese Information müsste anschließend wieder in die Aufteilung des GSD-Graphen einfließen. In der Praxis entscheidet die konkrete Anwendung, ob zusammenhängende Territorien erforderlich sind oder nicht.

#### 9.2.4 Einbinden der Residuen in die optimierte Teilgraphenauswahl

Das aktuelle Verfahren sieht vor, dass mit Hilfe eines genetischen Algorithmus eine optimierte Auswahl von Teilgraphen erfolgt. In einem nachgelagerten Schritt werden die verbleibenden Knoten als Residuen den ausgewählten Teilgraphen zugeordnet. Die Auswahl der Individuen wird von einer Kostenfunktion bestimmt, welche die Knotendistanzen zum Zentrum summiert und zusätzlich bewertet, wie viele Knoten noch nicht zugeordnet worden sind.

Anstelle der Bewertung der nicht zugeordneten Knoten könnten alle Residuen gleich zu ausgewählten Zentren zugeordnet werden und die Kostenfunktion könnte die entsprechenden Distanzen der Residuen gleich mit berücksichtigen.

Dieser Idee wurde in dieser Arbeit auch kurz nachgegangen, allerdings hatte die Implementation zu diesem Zeitpunkt noch das Problem des fehlenden Crossovers (siehe Abschnitt 6.3 auf Seite 45). Dadurch sind die Experimente mit dieser Idee im ersten Anlauf gescheitert und die Idee wurde zur Seite gelegt.

Für die Umsetzung müsste besonderes Augenmerk auf die Effizienz der Zuweisung der Residuen gelegt werden. Während aktuell die Residuen einmal im Verlauf des Verfahrens zugewiesen werden, müssten mit dieser Änderung die Residuen für jeden Crossover zugewiesen werden. Für die Praxis ist vorstellbar, dass sich die Lösungen in Hinblick auf die Minimierung der gesamten Knotendistanzen verbessern lassen.

### 9.3 Alternativer Ansatz

Das hier vorgestellte GOA-Verfahren besteht aus acht Schritten, welche hintereinander abgearbeitet werden (siehe Abschnitt 5 auf Seite 37). Der Lloyd-Algorithmus, welcher im *k*-means-Verfahren zum Einsatz kommt, führt im Gegensatz dazu die Schritte der Zuweisung zu einem Zentrum und der Aktualisierung der Position des Zentrums immer wieder aus, bis sich ein stabiler Zustand einstellt [80].

Wenn die letzten vier Schritte des GOA-Verfahrens betrachtet werden, die Zuweisung der Residuen, die Neuberechnung des zentralen Knotens, das „Stehlen“ von Knoten, die Berechnung des minsum-Zentrums, so kann auch hier eine sich wiederholende Sequenz von Zuweisung und Zentrumsaktualisierung erahnt werden.

Bereits während der Recherche phase der vorliegenden Arbeit wurde über die Verwendung von *k*-means oder einem ähnlichen Verfahren nachgedacht. Diese Idee wurde nicht weiter verfolgt, da im Vorhinein nicht feststeht, zu wie vielen Clustern sich die Menge der Geoobjekte aggregieren lässt. Im GOA-Verfahren wird im Schritt der „optimierten Auswahl von Teilgraphen“ festgelegt, wie viele Cluster sich aus den

Ausgangsdaten ergeben. Damit ist ab diesem Schritt die Anzahl der Cluster bekannt.

Nun wäre es denkbar, die letzten vier Schritte des GOA-Verfahrens durch eine dem Lloyd-Algorithmus ähnliche Prozedur von wiederholter Zuweisung und Aktualisierung zu ersetzen.

Weiter gedacht könnte auch danach gefragt werden, ob es möglich ist, ohne das Wissen um die Anzahl der Cluster einen Algorithmus zu gestalten, welcher, wann immer es sinnvoll ist, Cluster teilt und somit neue Zentren einfügt bzw., wenn es notwendig zur Wahrung der  $a$ -Anonymität ist, Cluster wieder zusammenfügt. Damit würde sich ein Algorithmus ergeben, welcher die drei Schritte Cluster teilen bzw. zusammenführen, Knoten neu zuweisen und Clusterzentren aktualisieren wiederholt durchführt, bis eine akzeptable Lösung erreicht ist.

Der Ansatz der Verwendung der kNN im Graphen würde nach wie vor seine Gültigkeit behalten. Während im  $k$ -means-Algorithmus die euklidische Distanz für die Zuweisung zu den Zentren verwendet wird, würde im Falle des GSD-Graphen die Liste der  $k$ -nächsten Nachbarn samt der dazugehörigen Distanzen herangezogen werden.

Als Startpunkte für eine weitere Literaturrecherche in diesem Zusammenhang wird an dieser Stelle auf [20, 68] verwiesen.

## 10 Einsatz in der Praxis

In dieser Arbeit wurden das GOA-Verfahren vorgestellt und auf OpenStreetMap-Testdatensätze angewandt. Damit es in der Praxis eingesetzt werden kann, müssen jedoch noch weitere „Integrationsarbeiten“ geleistet werden. Für den Anwendungsfall der Anonymisierung von Fahrgastdaten eines lokalen Bedarfverkehrssystems sollten folgende Punkte mitbedacht werden:

- Die Ausgangsdaten sollen den Datenquellen des übrigen Systems entsprechen. Wenn z. B. ein Bedarfsverkehrssystem in Österreich auf die Basemap.at-Daten aufbaut, dann sollte auch das GOA-Verfahren auf dieselben Daten angewendet werden, damit der Aufwand der Datenwartung minimal bleibt.
- In dieser Arbeit wurden die Geoobjekte durch einen relativ einfachen Filter, wie einer gültigen Adresse, identifiziert. Für die Anwendung in der Praxis muss die Frage gestellt werden, ob solch ein Filter ausreicht. Ein unbewohntes Haus trägt wohl nicht zu einer Anonymisierung bei. Zu diesem Zweck sollten die Ausgangsdaten mit weiteren Datenquellen korreliert werden. So könnten z. B.

Rasterdaten der Bevölkerungsdichte für eine Plausibilitätsprüfung herangezogen werden.

- Im Hinblick darauf dass sich die Menge der Geoobjekte verändern kann, wenn z. B. neue Häuser gebaut oder alte abgerissen werden, sollte ein Prozess definiert werden, wie und in welchem Abstand ein Update der Cluster erfolgt. Dazu empfiehlt es sich, eine Versionierung des Clusterings einzuführen und diese Versionsinformation gemeinsam mit den Fahrgastdaten abzuspeichern.
- Es muss der Frage nachgegangen werden, auf welches Gebiet die Clusterbildung beschränkt wird? Werden nur Häuser innerhalb eines Gemeindegebiets angefahren oder müssen auch die Nachbargemeinden mit berücksichtigt werden? Gibt es vielleicht Häuser, die nur über gemeindefremde Straßen erreicht werden können (siehe Abb. 16 auf Seite 62)? Diesen Fragen verdeutlichen, dass eine klare Definition des Einsatzgebietes und dessen Grenzen notwendig ist.

## 11 Zusammenfassung und Ausblick

Diese Arbeit befasst sich mit einem Verfahren, welches Geoobjekte, wie z. B. Häuser, anhand ihrer Entfernung im Straßenverlauf zu Cluster zusammenfasst. Diese Methode, genannt „Geoobjekte-Aggregations-Verfahren“ (kurz GOA), wurde primär in Hinblick auf die Anonymisierung von Fahrgastdaten von Bedarfsverkehrssystemen im ländlichen Raum entwickelt. In diesem Anwendungsfall dürfen die Fahrten nicht einzelnen Häusern zugeordnet werden, da dadurch Rückschlüsse auf die jeweiligen BenutzerInnen möglich wären. Für die statistische Erhebung der durchgeföhrten Fahrten kann nun anstelle der Koordinate des Fahrtbeginns oder des Fahrtendes, eine Referenz auf das Territorium des Clusters, in welchem sich die jeweilige Koordinate befindet, gespeichert werden.

Die vorliegende Arbeit lässt sich grob in drei Abschnitte einteilen. Im ersten Teil werden ausgehend von OpenStreetMap-Daten, die entsprechenden Geoobjekte und Straßenzüge identifiziert und daraus ein Graph generiert, wobei die Geoobjekte den Knoten, und die Distanzen zwischen den Geoobjekten im Straßenverlauf den Kanten entsprechen. Dazu werden die Ausgangsdaten zunächst in das räumliche Datenbanksystem PostGIS importiert und im Anschluss nach den gewünschten Geoobjekten gefiltert – im Fall von bewohnten Häusern ist eine Option nach gültigen Adressen zu filtern. Da Geoobjekte sich in der Regel nicht direkt auf der Straße befinden, müssen sie das Straßennetz projiziert werden und werden durch einen Punkt

darauf repräsentiert. Ein weiteres Problem bei der Erstellung dieses „Geoobjekte-Straßendistanz-Graphen“ ist das Auffinden von Kreuzungen von Straßenzügen, welches unter Zuhilfenahme eines räumlichen Index des Datenbanksystems effizient gelöst werden kann. Der zweite Teil der Arbeit beschäftigt sich, ausgehend von diesem Graphen, mit der Clusterbildung, wobei jeder Cluster eine Mindestanzahl von  $a$  Geoobjekten beinhalten muss. Der Parameter  $a$  ist damit eine untere Schranke und steht für die „Anonymität“ eines Clusters. Sollen z. B. fünf oder mehr Geoobjekte zu Cluster zusammengefügt werden, dann beträgt  $a = 5$ , die entsprechenden Cluster sind somit 5-anonym. Der triviale Fall von  $a = 1$  entspricht keiner Anonymität, da hier jedes Geoobjekt für sich steht. Die Clusterbildung wird durch ein mehrstufiges Verfahren umgesetzt, welches zunächst für jeden Knoten seine  $a$ -nächsten Nachbarn bestimmt, und damit potentielle Kandidaten für Cluster generiert. Für das Auffinden der nächsten Nachbarn im Graph, wird auf einen Algorithmus namens „geodesic kNN“ zurückgegriffen, welcher vom dem bekannten Dijkstra Algorithmus abgeleitet ist. Nach diesem Schritt stehen nun mehr Cluster-Kandidaten als notwendig zur Verfügung. Daher wird anschließend versucht, den ursprünglichen Graph möglichst komplett und eindeutig (ohne Überlappungen) mit einer entsprechenden Auswahl abzudecken. Diese Auswahl erfolgt mit Hilfe eines genetischen Algorithmus, welcher kompakte Cluster bevorzugt. Nachdem etwaige Residuen zugeordnet wurden, wird für jeden Cluster der zentrale Knoten und das korrespondierende Territorium in der Ebene bestimmt. Der dritte Teil beschäftigt sich mit der Anwendung des Verfahrens auf verschiedene Testdatensätze, der Evaluierung der Ergebnisse und einer Diskussion über Verbesserungsmöglichkeiten.

Das GOA-Verfahren eignet sich in seiner jetzigen Form bereits zur Anonymisierung von Fahrgastdaten. Als Schritt in die praktische Nutzung stehen an dieser Stelle mehre Wege offen. Zum einem ist die Integration in ein bestehendes Bedarfsverkehrssystem möglich. Andererseits könnte auch eine Art „Fahrgastdaten-Anonymisierungsdienst“ als ein Microservice umgesetzt werden.

Obwohl die vorliegende Methode für ländliche Bedarfsverkehrssysteme entwickelt wurde, sind auch alternative Anwendungen möglich, da sich damit eine nach unten beschränkte Anzahl an ortsfesten Geoobjekten mit Straßenbezug zusammenfassen lässt. Die Einbeziehung des Straßengraphen ansich, ist für andere Anwendungen im Bereich der Standortplanung mit einer oberen Schranke interessant. Im Kontext von ländlichen Gemeinden sind Anwendungen zur haushaltsübergreifenden Nutzung von Ressourcen oder der Optimierung kommunaler Infrastruktur denkbar.

# A Anhang

## A.1 Softwareinstallation unter Ubuntu 20.04

Unter Ubuntu 20.04 sind alle benötigten Pakete schon in den Standard-Repositorien enthalten. Das Datenbankmanagementsystem PostgreSQL, dessen Pythonanbindung, die PostGIS-Erweiterung, das OpenStreetMap Import Tool, die Visualisierungssoftware QGIS sowie die SciPy-Python-Bibliothek (Unterstützung von dünn besetzten Matrizen) können ohne Umwege direkt installiert werden.

```
1 $ sudo apt update
2 $ sudo apt -y install postgresql-12 postgresql-client-12
3 $ sudo apt -y install python3-psycopg2
4 $ sudo apt -y install postgis postgresql-12-postgis-3
5 $ sudo apt -y install osm2pgsql
6 $ sudo apt -y install qgis
7 $ sudo apt -y install python3-scipy python3-jsonpickle
```

Optional kann auch das Paket „Osmium Tool“ installiert werden, dieses kann OSM-Dateien bearbeiten (z. B.: zwei OSM-Dateien mergen)

```
1 $ sudo apt -y install osmium-tool
```

Für den Fall, dass pypy3 eingesetzt werden soll, können unter Ubuntu Zusatzpakete mit folgendem Befehl installiert werden:

```
1 $ pypy3 -m pip install --extra-index https://antocuni.github.io/
    pypy-wheels/ubuntu cpython numpy scipy heapdict
```

## A.2 Download von OpenStreetMap-Daten

Es gibt verschiedene Möglichkeiten, OpenStreetMap-Daten herunterzuladen; wenn der Bereich von Interesse lokal eingegrenzt ist (z. B.: einzelne Gemeinden), dann stellt das OpenStreetMap HTTP API eine gute Möglichkeit dar, den gewünschten Datensatz herunterzuladen. Dazu muss eine Bounding Box über die Längen- und Breitengrade definiert werden. Die Argumente hierfür haben die Reihenfolge: minimaler Längengrad, minimaler Breitengrad, maximaler Längengrad, maximaler Breitengrad [56].

Das HTTP API erlaubt jedoch nur den Download einer beschränkten Anzahl von OSM-Elementen. Es kann daher notwendig sein, dass der gewünschte Bereich in mehrere Teile zerlegt werden muss. Zum Beispiel ist der Download der Daten der Gemeinde Neukirchen an der Vöckla über dieses API nicht in einem Stück möglich. Diese Gemeinde erstreckt sich von 48,0142 °N, 13,4977 °O bis 48,0866 °N 13,5905 °O.

Abhilfe kann durch die Verwendung mehrerer Kacheln geschaffen werden. Hierfür teilen wir den Bereich in zwei Kacheln, entlang des Längengrades 48,0504 °N.

Der Download der Daten erfolgt nun in zwei Etappen:

```
1 $ wget -O 20211012_neukirchen_south.osm "https://api.openstreetmap.org/api/0.6/map?bbox=13.4977,48.0142,13.5905,48.0504"
2 $ wget -O 20211012_neukirchen_north.osm "https://api.openstreetmap.org/api/0.6/map?bbox=13.4977,48.0504,13.5905,48.0866"
```

Alternativ können auch OSM-Daten von ganzen Ländern oder der ganzen Erde heruntergeladen werden und mit dem „osmium“-Tool kann der Bereich von Interesse eingegrenzt und exportiert werden, bevor der Datenbankimport erfolgt.

```
1 $ osmium extract --bbox=16.2963,48.1822,16.3410,48.2101 austria-latest.osm.pbf -o wien15ter.osm
```

Die Website „<https://download.geofabrik.de/>“ bietet hierzu tagesaktuelle Datensätze an.

### A.3 Entfernen von Elementen außerhalb der Administrationsgrenzen

Mit folgenden SQL-Statements können alle Elemente, welche sich außerhalb des gewünschten Bereichs befinden, aus den Tabellen gelöscht werden:

```
1 database=> DELETE FROM planet_osm_line WHERE NOT ST_Intersects(way,
    (SELECT way FROM planet_osm_polygon WHERE "boundary" = 'administrative' AND "name" = 'Name der Gemeinde'));
2 database=> DELETE FROM planet_osm_point WHERE NOT ST_Intersects(way,
    (SELECT way FROM planet_osm_polygon WHERE "boundary" = 'administrative' AND "name" = 'Name der Gemeinde'));
3 database=> DELETE FROM planet_osm_polygon WHERE NOT ST_Intersects(
    way, (SELECT way FROM planet_osm_polygon WHERE "boundary" = 'administrative' AND "name" = 'Name der Gemeinde'));
```

### A.4 Commandline GOA Tools

Die Implementierung des vorgestellten Verfahrens erfolgt durch mehrere Commandline-Tools. In der Tabelle 4 auf Seite 79 findet sich ein Überblick über die Aufgaben der einzelnen Tools. Der Sourcecode gemeinsam mit den Testdatensätzen kann unter folgender URL heruntergeladen werden:

<https://github.com/ottingerg/GOA>

Tool	Aufgabe
boundOSM.py	entfernt Datenbankelemente, sofern sie sich außerhalb der angegebenen Administrationsgrenze befinden
exportGSD.py	exportiert die OSM-Daten aus einer Datenbank in einen GSD-Graphen
runGOA.py	aggregiert Geoobjekte zu anonymisierten Clustern
statsGOA.py	Statistiken zu den Geoobjekt-Clustern
statsGSD.py	Statistiken zu GSD-Graphen
vizGOA.py	erstellt Territorien und Zuordnungslinien in der Datenbank für die Visualisierung der Geoobjekt-Cluster
vizGSD.py	erstellt Linien in der Datenbank für die Visualisierung der Kanten des GSD-Graphen

Tab. 4: Überblick Commandline-Tools

## A.5 Laufzeit Messungen

Alle Messungen wurden auf einem Laptop Dell Latitude E5450 mit einer Intel-CPU i5-5300U durchgeführt.

### A.5.1 Export des GSD-Graphen

Die Laufzeit des Exports des GSD-Graphen (exportGSD.py) aus der PostGIS-Datenbank beträgt:

	Neukirchen	Purbach	Wien 15ter
GSD Export Time [s]	0.982	0.761	0.719

Tab. 5: Laufzeit von exportGSD.py

Version exportGSD.py: 0.1.0

### A.5.2 Laufzeit des GOA-Algorithmus

Die Laufzeit des GOA-Algorithmus (runGOA.py) beträgt:

	Neukirchen	Purbach	Wien 15ter
geodesic kNN [s]	1.287	0.706	3.329
Select Sets [s]	0.004	0.003	0.017
GA Select [s]	17.042	13.809	195.134
Residuals [s]	0.001	0.001	0.004
Recals Centers (central vertex) [s]	0.425	0.238	1.247
Steal Nodes [s]	0.003	0.002	0.018
Recalc Centers (minsum) [s]	0.402	0.201	1.246
GOA Total Time [s]	19.180	14.970	201.039

Tab. 6: Laufzeit von runGOA.py

Version runGOA.py: 0.3.4

## A.6 Auffinden von nicht zusammenhängenden Territorien

Um nicht zusammenhängende Territorien in der Datenbank aufzufinden, kann der Umstand ausgenutzt werden, dass zusammenhängende Territorien als ein einfaches Polygon (Typ: ST\_Polygon) und nicht zusammenhängende als ein Multipolygon (Typ: ST\_Multipolygon) in der Datenbank abgelegt werden.

```
1 database=> SELECT cluster_id from territories WHERE ST_GeometryType
          (polygon) = 'ST_MultiPolygon';
```

## A.7 Weitere Ergebnisvisualisierungen

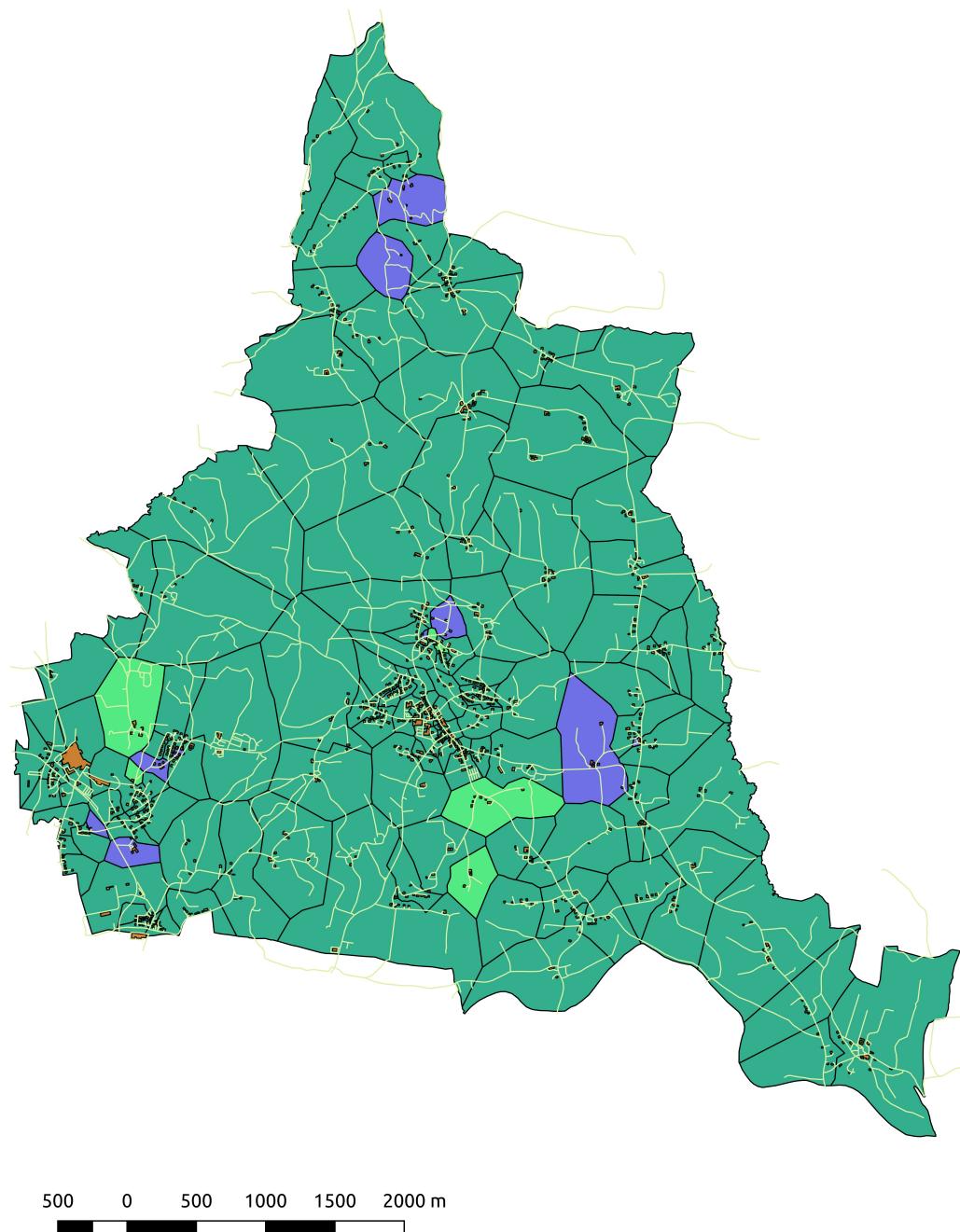


Abb. 22: Neukirchen Gebietseinteilung

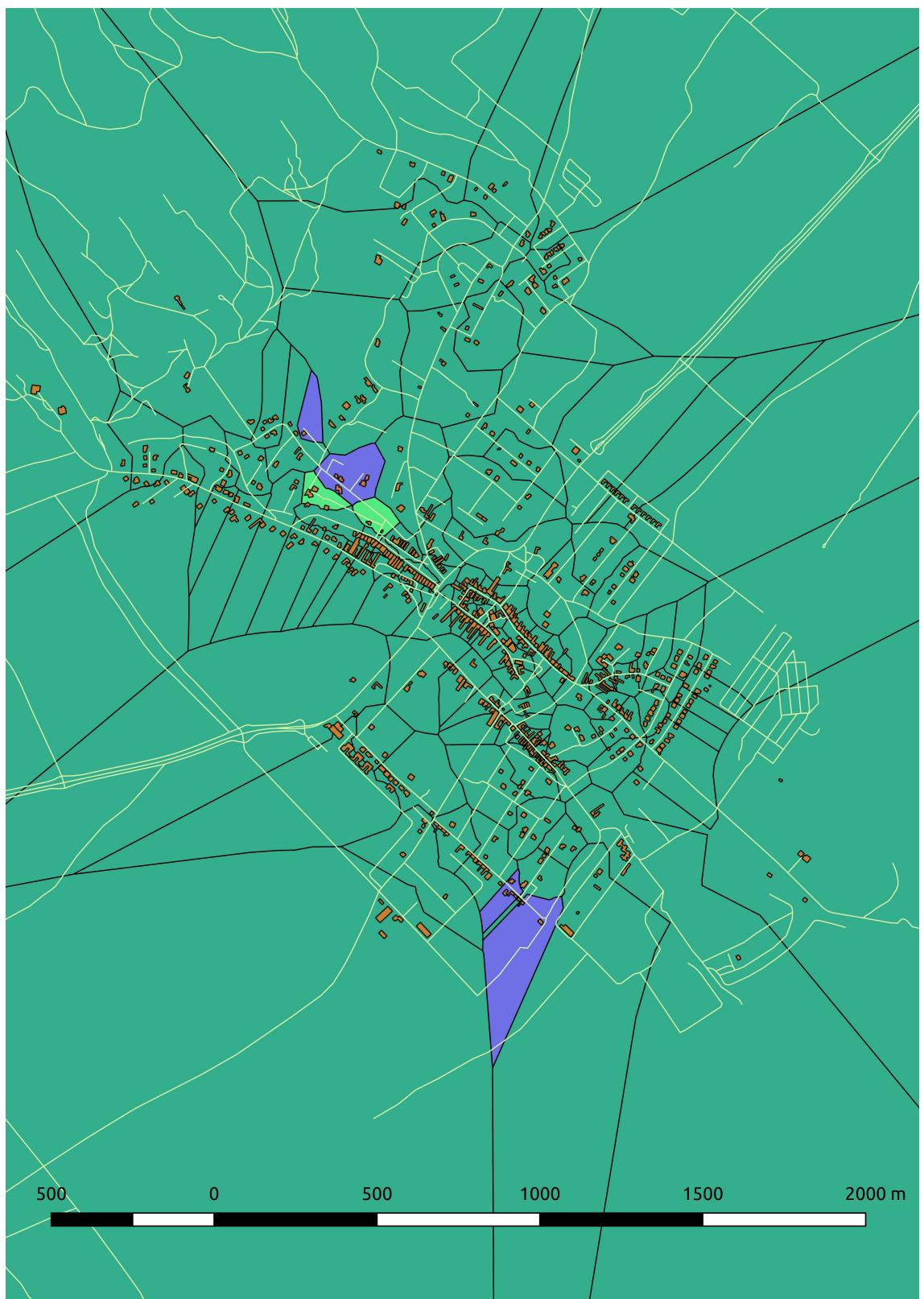


Abb. 23: Purbach Gebietseinteilung

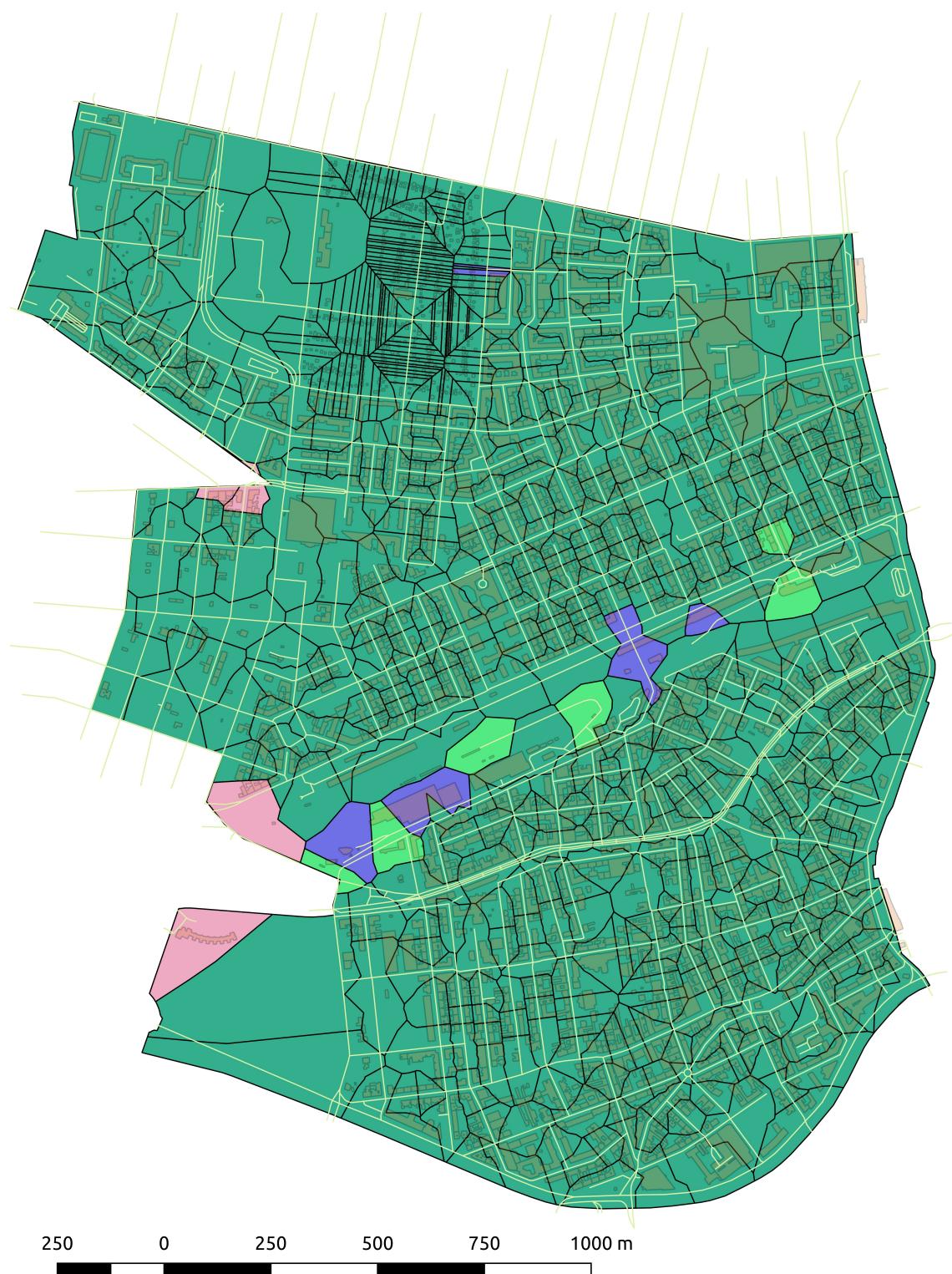


Abb. 24: Wien 15ter Gebietseinteilung

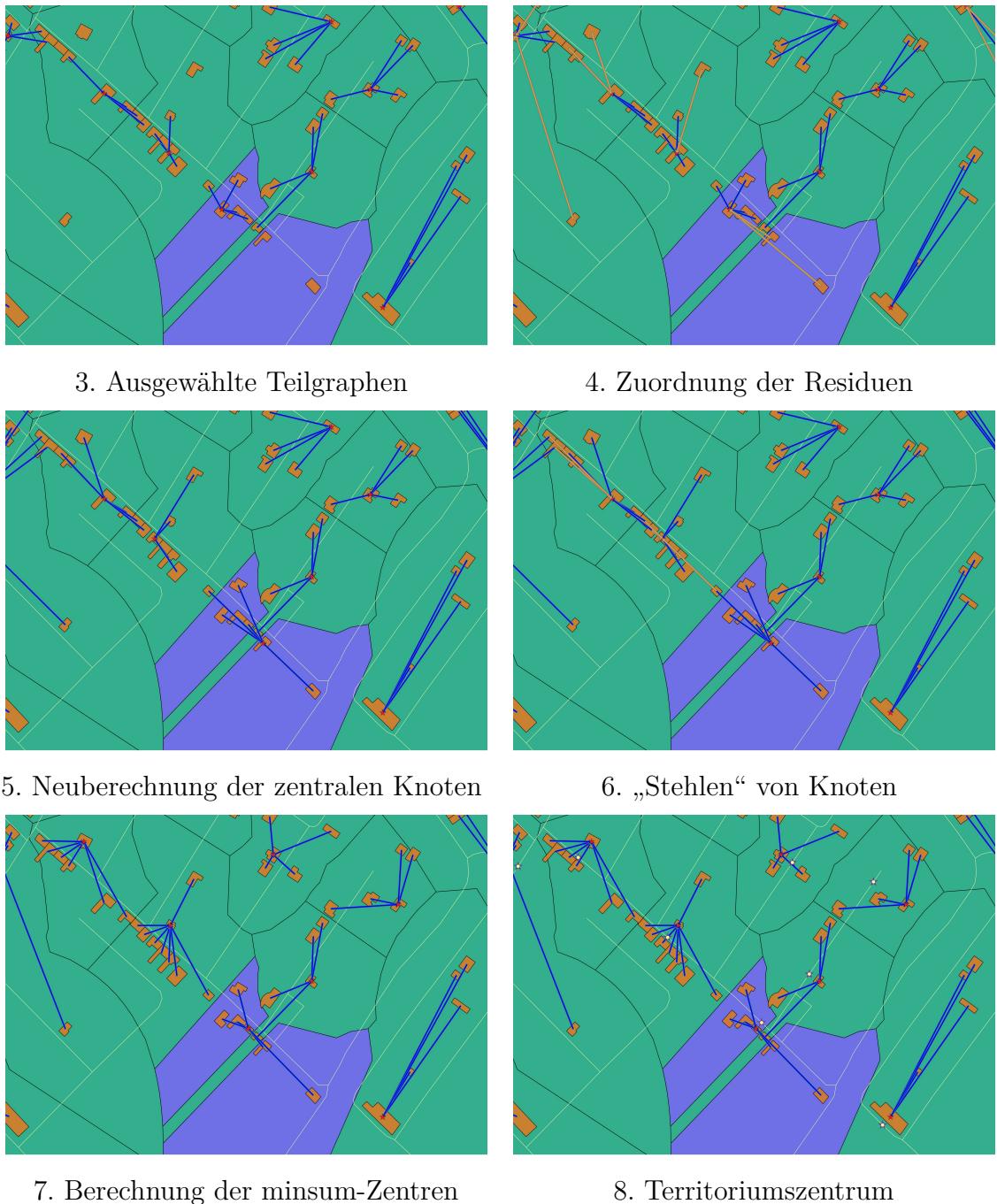


Abb. 25: GOA-Verfahren ab Teilgraphauswahl (Purbach)

# Abbildungsverzeichnis

1	Virtuelle Haltestellen der Aufzeichnungs-App . . . . .	4
2	Einteilung der Basisgebiete in Territorien (Ausschnitt) . . . . .	7
3	Similarity-Partitioning nach Güting (Ausschnitt) . . . . .	9
4	Straße mäandert am Berg . . . . .	9
5	Haus im Wald . . . . .	9
6	Häuser werden demselben Punkt auf der Straße zugeordnet . . . . .	11
7	GeoHash . . . . .	23
8	Traversale Mercator-Projektion . . . . .	25
9	Gemeinde mit Straßennetz . . . . .	27
10	Häuser an der Straße . . . . .	27
11	Bypass-Situation . . . . .	29
12	Unterführung . . . . .	29
13	Gefundene Kreuzungsknoten . . . . .	31
14	GOA-Verfahren ab Teilgraphauswahl . . . . .	51
15	Cluster im Ortszentrum . . . . .	61
16	Cluster an der Gemeindegrenze . . . . .	62
17	Nicht zusammenhängende Territorien . . . . .	64
18	Kleingartensiedlung auf der Schmelz . . . . .	64
19	Histogramm Clustergröße . . . . .	67
20	Histogramm Knotenentfernung . . . . .	68
21	Cluster mit 13 Geoobjekten . . . . .	72
22	Neukirchen Gebietseinteilung . . . . .	81
23	Purbach Gebietseinteilung . . . . .	82
24	Wien 15ter Gebietseinteilung . . . . .	83
25	GOA-Verfahren ab Teilgraphauswahl (Purbach) . . . . .	84

# Literatur

- [1] K. Aardal, P. L. van den Berg, D. Gijswijt, and S. Li. Approximation algorithms for hard capacitated k-facility location problems. *European Journal of Operational Research*, 242(2):358–368, 2015.
- [2] M. Adamczyk, J. Byrka, J. Marcinkowski, S. M. Meesum, and M. Włodarczyk. Constant factor fpt approximation for capacitated k-median. *arXiv preprint arXiv:1809.05791*, 2018.
- [3] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.
- [4] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows: theory, algorithms, and applications, 1993.
- [5] Anton (rp). Schematic diagram of cylinder projection used for utm (universal transverse mercator), November 2021. URL <https://de.wikipedia.org/wiki/UTM-Koordinatensystem#/media/Datei:Utmzylinderrp.jpg>.
- [6] W. G. Aref and I. F. Ilyas. Sp-gist: An extensible database index for supporting space partitioning trees. *Journal of Intelligent Information Systems*, 17(2):215–240, 2001.
- [7] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner. *Graph partitioning and graph clustering*, volume 588. American Mathematical Soc., 2013.
- [8] Z. Balkić, D. Šoštarić, and G. Horvat. Geohash and uuid identifier for multi-agent systems. In *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pages 290–298. Springer, 2012.
- [9] Basemap. Basemap Factsheet, 2017. URL <https://cdn.basemap.at/basemapat-fact-sheet.pdf>.
- [10] Basemap. Basemap homepage, January 2022. URL <https://basemap.at/>.
- [11] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, 28(09):643–647, 1979.
- [12] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information sciences*, 237:82–117, 2013.

- [13] J. Brownlee. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.
- [14] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.
- [15] J. Byrka, K. Fleszar, B. Rybicki, and J. Spoerhase. Bi-factor approximation algorithms for hard capacitated k-median problems. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 722–736. SIAM, 2014.
- [16] J. Byrka, B. Rybicki, and S. Uniyal. An approximation algorithm for uniform capacitated k-median problem with  $1 + \epsilon$  capacity violation. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 262–274. Springer, 2016.
- [17] D. A. Coley. *An introduction to genetic algorithms for scientists and engineers*. World Scientific Publishing Company, 1999.
- [18] A. S. da Cunha. Formulation and branch-and-cut algorithm for the minimum cardinality balanced and connected clustering problem. In *INOC*, pages 25–30, 2019.
- [19] G. Demirci and S. Li. Constant approximation for capacitated  $k$ -median with  $(1 + \epsilon)$ -capacity violation. *arXiv preprint arXiv:1603.02324*, 2016.
- [20] C.-H. Deng and W.-L. Zhao. Fast k-means based on k-nn graph. In *2018 IEEE 34th international conference on data engineering (ICDE)*, pages 1220–1223. IEEE, 2018.
- [21] D. B. des Inneren. Dritter GeoFortschrittsbericht der Bundesregierung, 2012.
- [22] E. W. Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- [23] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.
- [24] P.-O. Fjällström. *Algorithms for graph partitioning: A survey*, volume 3. Linköping University Electronic Press Linköping, 1998.

- [25] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.
- [26] I. A. für Geoinformationswesen. Behördenleitfaden zum Datenschutz bei Geodaten und -diensten, 2017.
- [27] D. Gijswijt and S. Li. Approximation algorithms for the capacitated k-facility location problems. *CoRR abs/1311.4759*, 2013.
- [28] S. Goderbauer. Optimierte Einteilung der Wahlkreise für die Deutsche Bundestagswahl. *OR News*, 52:19–21, 2014.
- [29] A. V. Goldberg, É. Tardos, and R. Tarjan. Network flow algorithm. Technical report, Cornell University Operations Research and Industrial Engineering, 1989.
- [30] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [31] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.
- [32] M. F. Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007.
- [33] R. H. Güting. An introduction to spatial database systems. *the VLDB Journal*, 3(4):357–399, 1994.
- [34] R. H. Güting, T. Behr, and J. K. Nidzwetzki. Distributed arrays: an algebra for generic distributed query processing. *Distributed and Parallel Databases*, pages 1–56, 2021.
- [35] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.
- [36] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1995.
- [37] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

- [38] W. Hochstättler and A. Schliep. *CATBox: An Interactive Course in Combinatorial Optimization*. Springer Science & Business Media, 2010.
- [39] J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [40] T. Ito, X. Zhou, and T. Nishizeki. Partitioning a weighted graph to connected subgraphs of almost uniform size. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 365–376. Springer, 2004.
- [41] U. Jensen, S. Netscher, and K. Weller. *Forschungsdatenmanagement sozialwissenschaftlicher Umfragedaten: Grundlagen und praktische Lösungen für den Umgang mit quantitativen Forschungsdaten*. Verlag Barbara Budrich, 2019.
- [42] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.
- [43] Jusline. DSVGO Artikel 4 Begriffsbestimmungen, January 2022. URL <https://www.jusline.at/gesetz/dsgvo/paragraf/4>.
- [44] J. Kalcsics, S. Nickel, and M. Schröder. A generic geometric approach to territory design and districting. 2009.
- [45] M. Karg and T. Weichert. Datenschutz und Geoinformation. Rechtsgutachten. kiel: Unabhängiges Landeszentrum für Datenschutz Schleswig-Holstein im Auftrag des Bundesministeriums für Wirtschaft und Technologie, 2007.
- [46] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- [47] D. E. Knuth. Dancing links. *arXiv preprint cs/0011047*, 2000.
- [48] R. B. Langley. The utm grid system. *GPS world*, 9(2):46–50, 1998.
- [49] M. Lewis, H. Wang, and G. Kochenberger. Exact solutions to the capacitated clustering problem: A comparison of two models. *Annals of Data Science*, 1(1):15–23, 2014.
- [50] Mobyome. Mobyome Aufzeichnungs-App – Anonymisierung der aufgezeichneten Geodaten, Februar 2019.
- [51] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. 1966.

- [52] A. Moscovich, A. Jaffe, and N. Boaz. Minimax-optimal semi-supervised regression on unknown manifolds. In *Artificial Intelligence and Statistics*, pages 933–942, 2017.
- [53] A. Moscovich, A. Jaffe, and B. Nadler. Minimax-optimal semi-supervised regression on unknown manifolds: supplementary material. 2017.
- [54] R. Obe and L. Hsu. Postgis in action, 2011.
- [55] I. A. of Oil & Gas Producers. *IOGP Publication 373-7-2: Geomatics Guidance Note Number 7, part 2*. Springer, 2021.
- [56] OpenStreetMap. Downloading data from openstreetmap, Oktober 2021. URL [https://wiki.openstreetmap.org/wiki/Downloading\\_data](https://wiki.openstreetmap.org/wiki/Downloading_data).
- [57] OpenStreetMap. Unterführung Jochling, Oktober 2021. URL <https://www.openstreetmap.org/export#map=17/48.01138/13.58306>.
- [58] OpenStreetMap. Openstreetmap key highway, January 2022. URL <https://wiki.openstreetmap.org/wiki/Key:highway>.
- [59] OpenStreetMap. Openstreetmap key tunnel, January 2022. URL <https://wiki.openstreetmap.org/wiki/Key:tunnel>.
- [60] S. C. Park and H. Shin. Polygonal chain intersection. *Computers & Graphics*, 26(2):341–350, 2002.
- [61] Paul Ramsey. Waiting for postgis 3: Geos 3.8, August 2021. URL <https://blog.cleverelephant.ca/2019/08/postgis-3-geos.html>.
- [62] Peter de Padua Krauss. Geohashes of odd and even digits are based on different space-filling curves, November 2021. URL <https://en.wikipedia.org/wiki/Geohash#/media/File:Geohash-OddEvenDigits.png>.
- [63] PostgreSQL. Postgresql documentation, November 2021. URL <https://www.postgresql.org/docs/current/plpgsql.html>.
- [64] Python. Frozenset, Dezember 2021. URL <https://docs.python.org/3/library/stdtypes.html?highlight=frozenset#frozenset>.
- [65] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.
- [66] S. E. Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

- [67] S. Schwartz. An overview of graph covering and partitioning. 2020.
- [68] S. Sieranoja. *Clustering with kNN graph and k-means*. PhD thesis, Itä-Suomen yliopisto, 2020.
- [69] D. Spielman. Spectral graph theory. *Combinatorial scientific computing*, 18, 2012.
- [70] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, 3(4):333–354, 2012.
- [71] T. Ulrich. Heuristiken für kombinierte Standort-und Gebietsplanung mit vorgegebenen und zusätzlichen, frei wählbaren Standorten. *Karlsruhe: Fakultät für Informatik Institut für Theoretische Informatik KIT*, 2013.
- [72] V. V. Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.
- [73] A. Wagner. Graphpartitionierung zur Berücksichtigung des Straßengraphen in der Gebietsplanung. 2013.
- [74] T. Weise. Global optimization algorithms-theory and application. *Self-Published Thomas Weise*, 2009.
- [75] D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.
- [76] Wikipedia. Evolutionary algorithm, December 2021. URL [https://en.wikipedia.org/wiki/Evolutionary\\_algorithm](https://en.wikipedia.org/wiki/Evolutionary_algorithm).
- [77] Wikipedia. Dense graph, November 2021. URL [https://en.wikipedia.org/wiki/Dense\\_graph](https://en.wikipedia.org/wiki/Dense_graph).
- [78] Wikipedia. Graph distance, November 2021. URL [https://en.wikipedia.org/wiki/Distance\\_\(graph\\_theory\)](https://en.wikipedia.org/wiki/Distance_(graph_theory)).
- [79] Wikipedia. Epsg geodetic parameter dataset, January 2022. URL [https://en.wikipedia.org/wiki/EPSG\\_Geodetic\\_Parameter\\_Dataset](https://en.wikipedia.org/wiki/EPSG_Geodetic_Parameter_Dataset).
- [80] Wikipedia. k-means-algorithmus, January 2022. URL [https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering).
- [81] Wikipedia. Web mercator projection, January 2022. URL [https://en.wikipedia.org/wiki/Web\\_Mercator\\_projection](https://en.wikipedia.org/wiki/Web_Mercator_projection).

- [82] Wikipedia. World geodetic system 1984, January 2022. URL [https://en.wikipedia.org/wiki/World\\_Geodetic\\_System#WGS84](https://en.wikipedia.org/wiki/World_Geodetic_System#WGS84).
- [83] H. P. Young. Measuring the compactness of legislative districts. *Legislative Studies Quarterly*, pages 105–115, 1988.
- [84] J. Yu and M. Sarwat. Indexing the pickup and drop-off locations of nyc taxi trips in postgresql—lessons from the road. In *International Symposium on Spatial and Temporal Databases*, pages 145–162. Springer, 2017.
- [85] Q. Zhou, U. Benlic, Q. Wu, and J.-K. Hao. Heuristic search to the capacitated clustering problem. *European Journal of Operational Research*, 273(2):464–487, 2019.

Ich erkläre, dass ich die vorliegende Abschlussarbeit mit dem Thema *Clusterbildung von Geoobjekten im Straßennetz* selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich, inhaltlich oder sinngemäß entnommenen Stellen als solche den wissenschaftlichen Anforderungen entsprechend kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate überprüft und ausschließlich für Prüfungszwecke gespeichert wird. Außerdem räume ich dem Lehrgebiet das Recht ein, die Arbeit für eigene Lehr- und Forschungstätigkeiten auszuwerten und unter Angabe des Autors geeignet zu publizieren.

Neukirchen, den 7. April 2022

---

Georg Ottinger