Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Agentenbasierte Veloverkehrsfluss Simulation für die Agglomeration Luzern“

Übertragung eines Modells aus Salzburg nach Luzern

vorgelegt von

Dipl.-Ing. (FH) Thomas Stadelmann
106914, UNIGIS MSc Jahrgang 2021

Betreuer/in:
Dr. Martin Loidl

Zur Erlangung des Grades
„Master of Science – MSc”

Buchrain, 7. Februar 2024

Danksagung

An dieser Stelle bedanke ich mich bei Dr. Martin Loidl für die Betreuung meiner Masterarbeit.

Bei Stefan Oberer vom Kanton Luzern, Martin Luternauer von der Stadt Luzern und Daniel Knoblauch von der Stadt Kriens bedanke ich mich für die zur Verfügung-gestellten Daten der Velozähler.

Bei Enrico Moresi von LUSTAT Statistik Luzern bedanke ich mich für seine Erläuterungen zu den Erwerbslosen und Schüler/Studenten Statistiken auf Stufe Kanton und Bund.

Auch mein Arbeitgeber, der Kanton Luzern, gilt der Dank für die finanzielle und zeitliche Unterstützung.

Ein spezieller Dank geht auch an meine Familie. Es war nicht immer einfach, Familie, Beruf und Studium zu organisieren, insbesondere mit drei Kleinkindern im Alter von 4 Jahren und zwei Mal 2 Jahren. Meine Frau hat mich während des Studiums immer wieder motiviert. Auch durfte ich, dank ihrem Einsatz, regelmässig an Wochenenden an meiner Masterarbeit arbeiten.

Zusammenfassung

Das Velo nimmt in der nachhaltigen Mobilität eine zentrale Stelle ein. Für die Verkehrsplanung sind daher Veloverkehrflüsse interessant. Viele Städte verwenden schon länger Velozähler, nur erfassen diese Durchfahrten an einzelnen Punkten und keine Flüsse. Mit einer Simulation lassen sich solche Verkehrsflüsse modellieren. Dazu gibt es eine vielversprechende Studie mit einer agentenbasierten Simulation aus Salzburg (Kaziyeva et al., 2021a). In dieser Arbeit wurde untersucht, ob sich das Modell aus dieser Studie von Salzburg auf Luzern übertragen lässt. Dabei wurde die Agglomeration Luzern als Systemgrenze verwendet. Für eine valide Grundlage wurde zuerst die Simulation mit 200 Durchläufen für Salzburg wiederholt und die Resultate verglichen. Für eine unterschiedliche Anzahl Durchläufe wurde der Variationskoeffizient pro Zählstation berechnet, um die Stabilität des Modells zu verifizieren. Danach wurden für Luzern die meisten Daten ersetzt. Nur die demografische Parametrisierung des Mobilitätsverhaltens von Salzburg wurde beibehalten, da angenommen wurde, dass sich die Salzburger bezüglich Velofahren ähnlich verhalten, wie die Luzerner. Für den Erwerbsstatus musste ein Ersatz gefunden werden.

Die Simulation konnte für Salzburg erfolgreich wiederholt werden. Nur eine Zählstation entsprach nicht ganz der Studie aus Salzburg, da vermutlich die Daten nicht mit dieser Studie übereinstimmten. Auch für Luzern lieferte die Simulation gute Resultate und es wurde für alle Daten ein geeigneter Ersatz gefunden. In naher Zukunft wird in der Schweiz voraussichtlich das Projekt *Verkehrsnetz CH* eine Vereinfachung bringen.

Abstract

The bicycle takes a centre place in sustainable mobility. Bicycle traffic flows are therefore of interest for transport planning. Many cities have been using bicycle counters for some time now, but these record passages at individual points and not flows. Such traffic flows can be modelled with a simulation. There is a promising study on this with an agent-based simulation from Salzburg (Kaziyeva et al., 2021a). This thesis investigated whether the model from this study can be transferred from Salzburg to Lucerne. The agglomeration of Lucerne was used as the system boundary. To ensure a valid basis, the simulation was first repeated with 200 runs for Salzburg and the results compared. The coefficient of variation per counting station was calculated for a different number of runs in order to verify the stability of the model. Most of the data for Lucerne was then replaced. Only the demographic parameterisation of the mobility behaviour of Salzburg was retained, as it was assumed that the people of Salzburg behave similarly to the people of Lucerne with regard to cycling. A replacement had to be found for the employment status.

The simulation was successfully repeated for Salzburg. Only one counting station did not fully correspond to the study from Salzburg, as the data presumably did not match the study. The simulation also produced good results for Lucerne and a suitable replacement was found for all the data. In the near future, the *Verkehrsnetz CH* project is expected to bring a simplification in Switzerland.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BFS
: Bundesamt für Statistik

CSV
: Comma-separated values

GIP
: Graphenintegrations-Plattform

GNSS
: Globales Navigationssatellitensystem (englisch global navigation satellite system)

LUSTAT
: LUSTAT Statistik Luzern

OSM
: OpenStreetMap

POI
: Points of Interest

Inhalt

1 Einleitung

Für eine nachhaltige Mobilität nimmt das Velo einen zentrale Stelle ein (Bos and Temme, 2014; Dimter et al., 2019). Als zusätzlichen Bonus bietet das Pendeln mit dem Velo auch gesundheitliche Vorteile (Neumeier et al., 2020). Daher sollten unter anderem Behörden grundsätzlich daran interessiert sein, das Velofahren zu fördern.

Für eine nachhaltige Verkehrsplanung ist die Erfassung der Velobewegungen eine wichtige Grundlage (Beitel et al., 2018) und in vielen Städten, wie z.B. in Salzburg, Wien und Luzern, werden stationäre Velozähler eingesetzt. Allerdings erfassen diese keine Velobewegungen, nur Durchfahrten an einem Punkt. Unklar ist, wie sich diese Punktdaten am besten nutzen lassen, um daraus Bewegungen zu modellieren. Für Velobewegungen gibt es den Ansatz GNSS Daten aus Fitness-Tracker und Smartphone-Anwendungen, wie Strava zu nutzen (Jestico et al., 2016). Diese Daten sind jedoch stark durch die jeweilige Zielgruppe verzerrt (Garber et al., 2019). Deswegen verwenden einige Studien zusätzlich räumliche sozioökonomischen Daten und Landnutzungsdaten um diese Verzerrung auszugleichen (Munira and Sener, 2020). Eine weitere Methode sind Simulationen, um den Verkehrsfluss zu modellieren (Leao and Pettit, 2017; Ziemke et al., 2017; Kaziyeva et al., 2018, 2021a). Dabei integriert nur die Simulationsstudie von Kaziyeva et al. (2021a) auch Daten aus einem Velozählernetzwerk. Noch ungeklärt ist bei der vorliegenden Studie die Übertragbarkeit des Modells auf andere Regionen.

In dieser Master Thesis wurde untersucht, ob das Simulationsmodell aus der Studie von Kaziyeva et al. (2021a) auch für die Agglomeration Luzern funktioniert. Für eine solide Grundlage wurde das Modell zuerst um ein Batch-Experiment ergänzt, welches die Simulation mehrfach laufen liess. Dies sollte Schwankungen der Korrelationskoeffizienten mit den Salzburger Daten aufzeigen. Zusätzlich wurde auch der Variationskoeffizient untersucht. Danach wurden alle räumlichen Daten durch Luzerner Daten ersetzt. Diese mussten teilweise aufwendig aufbereitet werden. Die demografische Parametrisierung des Mobilitätsverhaltens von Salzburg wurden beibehalten. Damit wurde, angelehnt an an die Originalstudie, für alle Zählstellen ein signifikanter ($p \leqslant 0.05$) Korrelationskoeffizient von 0.6 und höher erwartet. Die Mehrheit der Stationen sollten einen Koeffizienten von klar über 0.7 erreichen. Auf einen Vergleich mit GNSS Daten von Strava wurde verzichtet, da diese schon im Modell für Salzburg nicht korrelierten

(Kaziyeva et al., 2021a, p. 12). Prinzipiell liessen sich mit geeigneten Methoden Strava Daten nutzen, wie dies (Lee and Sener, 2021) belegen, jedoch würde dies den Rahmen dieser Arbeit sprengen.

2 Daten und Methoden

2.1 Bicycle model (version 2.0.0)

Als Grundlage für die Simulation wurde das «Bicycle model version 2» (Kaziyeva et al., 2021b) verwendet. Dieses enthielt die Daten aus der Studie für Salzburg. Dabei waren die Dateien *facilities.shp*, *homes.shp*, *workplaces.shp*, *intersections.shp*, *network.shp*, *outline_city.shp* und *outline_region.shp* enthalten, welche nach der Analyse, durch Luzerner Daten ersetzt wurden. Was die Daten genau enthielten und wie diese ersetzt wurden, wird in den folgenden Kapiteln beschrieben.

Es wurden weiter vier Wahrscheinlichkeitstabellen mitgeliefert. Es handelte sich dabei um die Abfolge der Aktivitätstypen, der Mobilitätsmodi, dem Start und der Dauer einer Aktivität. Diese wurden unverändert für Luzern übernommen.

Zusätzlich waren auch die erfassten Daten der Zählstellen enthalten, diese waren auf die Stunde aggregiert. Es wurden die Daten der Monate Oktober und November der Jahre 2012 bis 2019 verwendet und gemittelt. Diese zwei Monate wurden gewählt, da diese dem Zeitrahmen der Datenerhebung bezüglich der Verhaltensregeln entsprach.

2.2 Datenvergleich «Bicycle model v2» und Studie

Um sicherzustellen, dass das «Bicylce model v2» und dessen Daten dem Modell aus der Studie entspricht, wurden die mitgelieferten Daten mit der Studie verglichen. Dabei wurden auch Tabellen in der Studie, welche im Model (GAML Code) abgebildet worden sind, berücksichtigt.

Dieser Vergleich diente als Grundlage, um allfällige Abweichungen in den Resultaten zu erklären.

2.3 Analyse des «Bicycle model v2»

Durch das stochastische Verhalten der Agenten im Modell, war es sinnvoll, die Simulation mehrmals mit den selben Parameter laufen zu lassen. In der Originalstudie lief die Simulation maximal zehn Mal mit den selben Parametern. Dabei wurde für 20 zu-

fällige Standorte der Variationskoeffizient der Durchfahrten untersucht. Die Autoren kamen zum Schluss, dass ein Simulationsdurchlauf reicht. Um dies zu verifizieren, wurde eine Analyse des Modells mit 200 Durchläufen durchgeführt.

Dazu wurde zuerst das Skript um ein *Batch* Experiment mit den selben Parametern wie das Originalexperiment ergänzt:

```
experiment repeated type: batch repeat: 200 keep_seed: true { …
```

Damit keine grösseren Anpassungen notwendig waren, welche mit dem Risiko zusätzlicher Fehlers einhergehen, wurde auf eine Abbruchbedingung verzichtet und im *Reflex* `stopSimulation` (Zeile 851) das `do pause` durch ein `do die` ersetzt. Die vier Ausgabe Pfade `heatmapFileName`, `activeCyclistsFileName`, `countsFileName` und `tripsFileName` wurden um den Platzhalter `[simulation]` ergänzt, um die Dateien pro Durchlauf in einem separaten Ordner zu speichern. Im *init* Teil wurde dieser Platzhalter wie folgt überschrieben:

```
countsFileName <- replace(countsFileName, '[simulation]', name);
```

`name` entspricht dabei dem Namen der Simulation. Wenn nichts angeben wird, ist dies jeweils «Simulation» und die Nummer des Durchlaufs, welche bei Null startet, also z.B. *Simulation 42*.

Aus den Simulationsresultaten wurde für alle Zählstationen der Pearson-Korrelationskoeffizient, der p-Wert, die simulierten Durchfahrten und der Variationskoeffizient untersucht. Der Variationskoeffizient (c_v) ist das Verhältnis zwischen der Standartabweichung (s) und dem arithmetischem Mittel($\bar{x}$) (Hendricks and Robey, 1936):

$$c_v = \frac{s}{\bar{x}}$$

Die Ergebnisse wurden in einer Tabelle mit den Originalwerten verglichen, wie auch als Boxplot aufbereitet. Der Variationskoeffizient wurde zu den 200 Simulationen, zusätzlich noch für jeweils 5, 10, 15, 25, 50 und 100 Simulationen berechnet. Dabei handelte es sich jeweils um ein eigenständiges Set von Simulationen und nicht nur um einen Teil der ursprünglichen 200 Simulationen. Details zur Datenaufbereitung sind im vollständigen R-Script im Anhang G ersichtlich.

2.4 Definition Agglomeration Luzern

Die räumliche Ausdehnung des Studienperimeters entsprach der Agglomeration Luzern. Dazu existieren jedoch unterschiedliche Definitionen. Für die Simulation wurde die Definition der Kerngemeinden von LUSTAT Statistik Luzern gewählt. Dies sind

die 10 Gemeinden Adligenswil, Buchrain, Dierikon, Ebikon, Emmen, Horw, Kriens, Meggen, Rothenburg und die Stadt Luzern als Kernstadt ("Gemeindetypologie - LUSTAT," 2023). Die zur Stadt Luzern gehörende Exklave Bürgenstock wird für die Studie nicht berücksichtigt, da diese nicht direkt an des Verkehrsnetz der Agglomeration angeschlossen ist und innerhalb dieser nur per Schiff erreichbar ist. Die Agglomeration entspricht dabei der Systemgrenze in der Simulation.

Um Randeffekte zu vermeiden, wurde als eigentliches Analysegebiet nur der Stadtkern verwendet. Dabei wurde das seit 2010 zur Stadt Luzern gehörende Littau ("Littau," 2023) der Agglomeration zugeordnet.

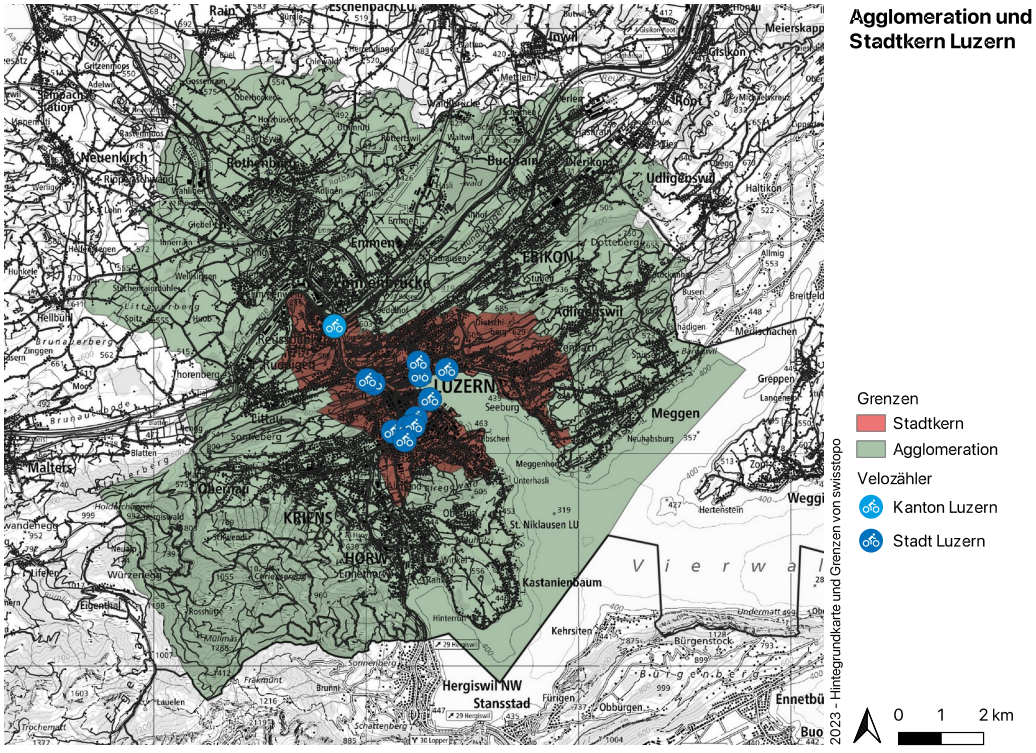

Abbildung 1: Agglomeration und Stadtkern Luzern (ohne Bürgenstock)

2.5 Facilities Daten

Bei den «Facilities» handelt es sich um einen Punktdatensatz. In diesem werden für das Model Lokalitäten, wie Shops, Schulen, Universitäten, usw. abgelegt. Dabei werden die Arbeitsplätze und die Wohnungen separat behandelt und sind nicht im Facilities Datensatz. Es wurde zwischen folgenden sieben Typen unterschieden:

- authority

- doctor
- kindergarten
- recreation
- school
- shop
- university

facilities	
id	NUMERIC(10,0)
type	CHARACTER VARYING(16)
osm_id	CHARACTER VARYING(12)
fclass	CHARACTER VARYING(40)
name	CHARACTER VARYING(100)
wkb_geometry	geometry

Abbildung 2: Attribute von facilities

Der Kanton Luzern besitzt keinen flächendeckenden Points of Interest (POI) Datensatz. Auch fehlen georeferenzierte Daten amtlicher Gebäude. Nur die kantonalen Gebäude sind vollständig vorhanden, wobei diese meistens für die Öffentlichkeit nicht von Interesse sind. Daher wurden für die «Facilities» OpenStreetMap (OSM) Daten verwendet, welche von Geofabrik ("OpenStreetMap. OSM Dump — Switzerland," 2023) als Shapefile bezogen wurden. Davon wurden die beiden POI Layer (*gis_osm_pois_free_1* und *gis_osm_pois_a_free_1*) verwendet. Die Daten wurden zuerst mit einem Puffer von 50 Metern auf die Systemgrenzen zugeschnitten. Danach wurde das Referenzsystem von *WGS84* auf *LV95* (EPSG:2056) geändert. Um im Polygonlayer Mehrfacheinträge zu verhindern, wurden diese anhand dem Feld `osm_id` aufgelöst (*dissolve*). Danach wurden die Zentroide aller Polygone ermittelt. Das Ergebnis wurde mit dem Punkt POI-Daten zusammengeführt.

Alle POI Daten von der Geofabrik GmbH enthalten das Attribut fclass (Ramm, n.d., p. 3). Anhand diesem Attribut wurde ein «Mapping» zu den sieben Facility Typen erstellt (siehe Tabelle 1). Dabei wurde für `recreation` bewusst auf die Klasse `swimming_pool` verzichtet, da diese zu viele unnötige Schwimmbecken enthielt und die wichtigsten öffentlichen Bäder schon mit `sports_centre` abgedeckt wurden.

Tabelle 1: Mapping von fclass auf type

type	***fclass***
authority	town_hall
doctor	doctors, hospital, clinic
kindergarten	kindergarten
recreation	arts_centre, artwork, attraction, bar, biergarten, cafe, castle, cinema, community_centre, dog_park, fast_food, guesthouse, hostel, hotel, ice_rink, library, nightclub, park, playground, pub, restaurant, sports_centre, stadium, theatre, theme_park, tourist_info, viewpoint, zoo
school	school
shop	bakery, beauty_shop, beverages, bicycle_shop, bookshop, butcher, car_dealership, chemist, clothes, computer_shop, convenience, department_store, doityourself, florist, furniture_shop, garden_centre, general, gift_shop, greengrocer, hairdresser, jeweller, kiosk, laundry, mall, mobile_phone_shop, newsagent, optician, outdoor_shop, pharmacy, shoe_shop, sports_shop, stationery, supermarket, toy_shop, travel_agent, video_shop
university	university

Alle Details sind im Processing Script im Anhang A ersichtlich, dabei wurden jeweils die definierten Standartwerte verwendet. Das Mapping wurde auch mit Daten für Salzburg getestet und mit dem Facilities Datensatz, welcher im Bicycle Model v2 (Kaziyeva et al., 2021b) mitgeliefert wurde, verglichen. Daher funktionierte das Processing Script auch mit *WGS 84 / UTM zone 33N* (EPSG:32633).

2.6 Workplaces Daten

Bei den «Workplaces» Daten handelt es sich um einen Polygondatensatz. Dieser besteht aus einem Hektarraster, somit entspricht jedes Feature einem 100x100 Meter Rechteck und enthält jeweils die aggregierte Anzahl der Arbeitsplätze. Die Daten stammen vom Bundesamt für Statistik (BFS) und wurden als Comma-separated values (CSV) Datei zur Verfügung gestellt «Statistik der Unternehmensstruktur (STATENT), Beschäftigte und Arbeitsstätten 2021» (Bundesamt für Statistik, 2023a). Aus Datenschutzgründen gibt das BFS Attribute mit einem Wert zwischen 1 und 4 als 4 aus. Bezogen auf die Arbeitsplätze, stellte diese Reduktion der Genauigkeit kein Problem dar.

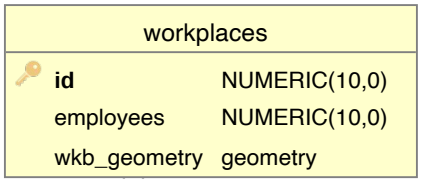

Abbildung 3: Attribute von workplace

In der Datei wurde jeweils jede Hektare durch die Koordinaten ihres südwestlichen Eckpunkts identifiziert. Dabei wurde für E- und N-Koordinaten der Bezugsrahmen LV95 verwendet. Um diese Daten in die benötigte Form zu bringen, wurden diese als Punktdaten importiert. Das Hektarraster wurde über die Systemgrenzen erstellt. Im Hektarraster wurden alle Featuers auf einen passenden Wert im südwestlichen Eckpunkt überprüft. Dabei wurde ein 2x2 Meter Rechteck verwendet, um allfällige Rundungsfehler zu vermeiden. Details sind im Processing Script im Anhang B ersichtlich.

2.7 Homes Daten

Bei den «Homes» Daten handelt es sich auch im einen Polygondatensatz. Im Original wird ein 250x250 Meter Raster verwendet. Jedes Feature enthält die aggregierte Anzahl Bewohner und deren geschlechtergetrennte Altersstruktur (siehe Abbildung 4). Zusätzlich ist auch der Erwerbsstatus, respektive der aktuelle Ausbildungsstatus enthalten. Dabei wird zwischen den folgenden sieben Werten unterschieden, wobei alle ausser «pupils» geschlechtergetrennt sind (siehe auch Abbildung 20):

- employed
- unemployed
- pension
- students
- inactive (ökonomisch inaktive und nicht als arbeitslos erfasst)
- unknown
- pupils

homes	
id	NUMERIC(10,0)
residents	NUMERIC(10,0)
m_tot	NUMERIC(10,0)
m_below_5	NUMERIC(10,0)
m_5_9	NUMERIC(10,0)
m_10_14	NUMERIC(10,0)
m_15_19	NUMERIC(10,0)
m_20_24	NUMERIC(10,0)
m_25_29	NUMERIC(10,0)
m_30_34	NUMERIC(10,0)
m_35_39	NUMERIC(10,0)
m_40_44	NUMERIC(10,0)
m_45_49	NUMERIC(10,0)
m_50_54	NUMERIC(10,0)
m_55_59	NUMERIC(10,0)
m_60_64	NUMERIC(10,0)
m_65_69	NUMERIC(10,0)
m_70_74	NUMERIC(10,0)
m_75_79	NUMERIC(10,0)
m_80_84	NUMERIC(10,0)
m_85_89	NUMERIC(10,0)
m_over_90	NUMERIC(10,0)
f_tot	NUMERIC(10,0)
f_below_5	NUMERIC(10,0)
f_5_9	NUMERIC(10,0)
f_10_14	NUMERIC(10,0)
f_15_19	NUMERIC(10,0)
f_20_24	NUMERIC(10,0)
f_25_29	NUMERIC(10,0)
f_30_34	NUMERIC(10,0)
f_35_39	NUMERIC(10,0)
f_40_44	NUMERIC(10,0)
f_45_49	NUMERIC(10,0)
f_50_54	NUMERIC(10,0)
f_55_59	NUMERIC(10,0)
f_60_64	NUMERIC(10,0)
f_65_69	NUMERIC(10,0)
f_70_74	NUMERIC(10,0)
f_75_79	NUMERIC(10,0)
f_80_84	NUMERIC(10,0)
f_85_89	NUMERIC(10,0)
f_over_90	NUMERIC(10,0)
m_below_15	NUMERIC(10,0)
f_below_15	NUMERIC(10,0)
wkb_geometry	geometry

Abbildung 4: Attribute von homes ohne Erwerbsstatus

Die ständige Wohnbevölkerung nach Altersklasse und Geschlecht ist, wie die Arbeitsplätze, beim BFS als Hektarraster erhältlich «Statistik der Bevölkerung und Haus-

halte (STATPOP) 2022» (Bundesamt für Statistik, 2023b). Aus Datenschutzgründen werden hier Werte von 1 bis und mit 3 als 3 gelistet. Somit stimmte das Total nicht immer mit der Summe aller Klassen überein. In Bezug auf die Genauigkeit der Simulation wurde mit keinen negativen Auswirkungen gerechnet. Das Hecktarraster wurde nicht in ein 200x200 Meter umgewandelt, obwohl dies näher an dem 250x250 Meter Raster gewesen wäre, um die Abweichung nicht zu vergrössern. Die beiden Felder `m_below_15` und `f_below_15` mussten jeweils aus drei Altersklassen zusammengezählt werden. Details sind im Processing Script im Anhang C ersichtlich.

Daten für den Erwerbsstatus existieren gemäss LUSTAT keine in der nötigen Auflösung. Diese werden nur stichprobenartig erhoben und sind nur für grössere Gemeinden verlässlich. Somit wurden schweizweite Daten verwendet, welche die Verteilung anhand der Bevölkerungsstruktur nach Alter und Geschlecht zuliessen. Während der Initialisierung des Modells, wurden die statistischen Daten auf das Hecktarraster verteilt. Dies wird im Abschnitt 2.11 genauer erläutert und im Anhang F sind die Details ersichtlich.

2.8 Erwerbsstatus Daten

Daten zum Erwerbsstatus sind original in den «Homes» Daten enthalten. Da diese für Luzern fehlen, wurde auf schweizweite Daten zurückgegriffen und als CSV aufbereitet (*employment_status_probabilities.csv*). Als Ausgangsdaten wurde «Arbeitsmarktstatus nach Geschlecht, Nationalität, Altersgruppen, Familientyp» (Bundesamt für Statistik, 2023c) und «Schulbesuchsquoten der 3-31-Jährigen» (Bundesamt für Statistik, 2023d) verwendet. Die Daten wurden für die folgenden 5 Status aufbereitet.

- pupil
- student
- employed
- unemployed
- inactive

Es wurden dabei die Altersklassen der Home Daten verwendet, jedoch nur bis zum Alter von 69. Details zur Aufbereitung sind im R-Script im Anhang D ersichtlich.

Tabelle 2: Auszug aus Erwerbsstatus Daten (employment_status_probabilities.csv)

type	m_below_5	m_5_9	m_10_14	m_15_19	m_20_24	m_25_29	...
pupil	9.87	99.02	99.47	82.19	13.43	2.65	
student	0.00	0.00	0.00	3.15	27.04	17.42	
employed	0.00	0.00	0.00	61.58	61.58	90.42	
unemployed	0.00	0.00	0.00	4.98	4.98	3.74	
inactive	0.00	0.00	0.00	33.44	33.44	5.85	

2.9 Network Daten

Bei den «Network» Daten handelt es sich um einen Liniendatensatz und dieser entspricht dem Strassennetz. Jedes Strassensegment enthält mehrere Attribute. Diese ermöglichen es der Simulation, das Netz mit einem Sicherheitsindex zu gewichten. Dies erlaubt wiederum eine sicherheitsorientierte Routenfindung des Veloverkehrs. Bei den Attributen handelt es sich um die Strassenkategorie, das Vorhandensein von Veloinfrastruktur und eingerichteten Velowegen, Fahrzeugbeschränkungen, Steigung, Fahrbahnbelag und die Höchstgeschwindigkeit für motorisierte Fahrzeuge. Mehrere Attribute wurden für beide Richtung vergeben, dabei bedeutet die Endung *_ft* in Richtung zur gezeichneten Linie und *_tf* in umgekehrt Richtung.

network	
id	NUMERIC(10,0)
osm_id	NUMERIC(10,0)
road_name	CHARACTER VARYING(254)
basetype	CHARACTER VARYING(254)
brunnel	NUMERIC(10,0)
restric_ft	NUMERIC(10,0)
restric_tf	NUMERIC(10,0)
oneway_ft	NUMERIC(10,0)
oneway_tf	NUMERIC(10,0)
bic_inf_ft	CHARACTER VARYING(254)
bic_inf_tf	CHARACTER VARYING(254)
mit_vol_ft	NUMERIC(10,0)
mit_vol_tf	NUMERIC(10,0)
d_route_ft	CHARACTER VARYING(254)
d_route_tf	CHARACTER VARYING(254)
road_categ	CHARACTER VARYING(254)
max_sp_ft	NUMERIC(10,0)
max_sp_tf	NUMERIC(10,0)
ad_edge_ft	NUMERIC(10,0)
ad_edge_tf	NUMERIC(10,0)
parking_ft	CHARACTER VARYING(254)
parking_tf	CHARACTER VARYING(254)
pavement	CHARACTER VARYING(254)
width_lane	NUMERIC(24,15)
gradient_f	NUMERIC(10,0)
gradient_t	NUMERIC(10,0)
rails	CHARACTER VARYING(254)
n_lanes_ft	NUMERIC(24,15)
n_lanes_tf	NUMERIC(24,15)
land_use	CHARACTER VARYING(254)
wkb_geometry	geometry

Abbildung 5: Attribute von network

Im Original baut dieser Datensatz auf der Graphenintegrations-Plattform (“ÖVDAT - Österreichisches Institut für Verkehrsdateninfrastruktur,” 2023) auf. In der Schweiz ist GIP das Vorbild für das Verkehrsnetz CH (“Bundesamt für Landestopografie,” 2023b). Ende 2023 existiert mit swissTNE Base (“Bundesamt für Landestopografie,” 2023a) ein erster Datensatz aus dem Verkehrsnetz CH. Dieser eignet sich jedoch noch nicht, da die meisten der benötigten Attribute fehlen. Somit wurden auch hier OSM Daten verwendet. Diese eignen sich trotz Crowd-Source typischer Inkonsistenzen (Graser et al., 2013; Neis, 2014) als Ersatz von Behördendaten für einen *Saftyindex*

(Loidl and Zagel, 2014). Das im Original vorhandene Attribut *link_id* wurde durch osm_id ersetzt. *link_id* war der Verweis zum GIP Linknetz.

Für die Aufbereitung der OSM Daten wurden folgende Schritte unternommen:

- Download OSM Daten
- Import in Datenbank
- Entfernern nicht benötigter Strassensegmente
- Nicht verbundene Strassensegmente (*disconnected islands*) entfernen
- Umwandeln der OSM Attribut in die benötigte Form
- Ergänzen fehlender Attribute durch zusätzliche Datenquellen

Mit der Ausnahme der ersten beiden Punkte wurde alles in einem Processing Script erledigt. Das vollständige Script mit allen Details ist im Anhang E gelistet.

2.9.1 Download OSM Daten

Die Daten wurden via Overpass API ("Overpass turbo," 2023) als OSM Rohdaten bezogen, da der freie OSM Export der Geofabrik nicht alle für ein Strassennetz benötigten Daten enthält.

In Overpass turbo wurde Zentrum und Zoom mit dem URL-Parameter «C» angegeben `C=47.092566;8.261373;11`. Für die Abfrage wurde folgendes Overpass Query verwendet:

```
[out:xml] [timeout:25];
(
    node["highway"]( {{bbox}});
    way["highway"]( {{bbox}});
    // get cycle route relations
    relation[route=bicycle]({{bbox}});
);
(._;>;);
out body;
```

2.9.2 Import in Datenbank

Der OSM Export wurde danach, mit Hilfe des bewährten Tools osm2pgrouting (Bejarano et al., 2015; Felício et al., 2022; "pgRouting Community," 2023) in eine PostgreSQL Datenbank importiert. Dabei wurden die mitgelieferten Voreinstellungen für Velos unverändert übernommen. Dadurch wurden unter anderem Autobahnen nicht importiert. Damit die benötigten OSM Tags erhalten blieben, wurden die beiden Parameter `--addnodes` und `--tags` benötigt. Der erste Parameter ergänzte den Import um die Tabelle `osm_ways` und der zweite Parameter ergänzte diese Tabelle um das Feld `tags`. Der gesamte Aufruf von *osm2pgrouting* sah damit wie folgt aus:

```
osm2pgrouting --f ./luzern.osm --conf
/usr/local/Cellar/osm2pgrouting/2.3.8_8/share/osm2pgrouting/mapconfig
_for_bicycles.xml --dbname osm --addnodes --tags –clean
```

Durch einen Bug ("osm_relations table does not get populated · Issue #301 · pgRouting/osm2pgrouting," 2023) wurde die `osm_ralations` Tabelle nicht befüllt. Diese Tabelle wurde für die Velorouten benötigt. Die nötigen Daten liessen sich einfach mit osm2pgsql ("Osm2pgsql," 2023) importieren:

```
osm2pgsql -d osm --hstore --proj=4326 --slim ./luzern.osm
```

Dies erstellte die Tabelle `planet_osm_rels` und befüllte diese mit den benötigten Beziehungen zu den Velorouten. Im Network Processing Script (Anhang E) wurde die Tabelle `osm_ways` um die Routen via folgendem SQL Query als `network` im `tags` Feld ergänzt:

```
UPDATE
    osm_ways
SET
    tags = tags || hstore('network', subquery.route)
FROM
    (   SELECT
            (ARRAY_AGG(hstore(r.tags)->'network' ORDER BY array_position(
                ARRAY['icn','ncn','rcn','lcn'], hstore(r.tags)->'network') ))[1]
              AS route,
            w.osm_id
        FROM
            osm_ways AS w
        INNER JOIN
            planet_osm_rels AS r
        ON
            'network' = ANY(r.tags)
        AND 'w'||w.osm_id = ANY(r.members)
        GROUP BY
            w.osm_id) AS subquery
WHERE
    osm_ways.osm_id=subquery.osm_id;
```

Dabei wurde nur jeweils die am höchsten gewichtete Route übernommen.

1. *icn*: International Cycling Network
2. *ncn*: National Cycling Network
3. *rcn*: Regional Cycling Network
4. *lcn*: Local Cycling Network

2.9.3 Entfernern nicht benötigter Strassensegmente

Eine ganze Reihe von Strassen sind für den Veloverkehr nicht oder noch nicht interessant. Darunter fallen Autobahnen oder auch erste geplante Strassen.

Zuerst wurden alle Strassen ausserhalb der Systemgrenzen, sprich der Agglomeration, via *clip* Funktion entfernt. Danach wurden alle Strassensegmente gemäss folgender Liste mit Werten für das *highway* Tag gezielt entfernt.

- *proposed*, *construction*: geplante und im Bau befindende Strasse

- *corridor*: Gang im Inneren eines Gebäudes
- *elevator*: Lift
- *platform*: Perron im Bahnhof oder Busshaltestelle
- *motorway*, *motorway_link*: Autobahn und Autobahnzufahrt
- *trunk*, *trunk_link*: Autostrasse und Autostrassenzufahrt
- *raceway*: Rennstrecke

Wobei ein Grossteil der nicht benötigten Strassensegment schon *osm2pgrouting* entfernte.

2.9.4 Nicht verbundene Strassensegmente (disconnected islands) entfernen

Das Entfernen einzelner Strassensegmente, wie auch der Zuschnitt, führte zu Strassensegmenten ohne Verbindung zum restlichen Strassennetz. Dies ist z.B. bei einer Treppe bei der Sportanlage Bramberg gut ersichtlich.

Abbildung 6: Treppe Sportanlage Bramberg mit "disconnected islands"

Dabei wurden durch *osm2pgrouting* die Segmente mit dem Wert *steps* im Tag *highway* und ohne *tracktype* Tag entfernt. Einzelne verbleibenden Segmente mit dem Wert *footway* hatten danach keine Verbindung mehr zum restlichen Strassennetz.

Um diese Segmente zu erkennen und zu entfernen, wurde das Python Netzwerkanalyse Tool NetworkX ("NetworkX developers," 2023) eingesetzt. Der nötige Python-Code wurde analog des QGIS Plugins Disconnected Islands ("Disconnected Islands — QGIS Python Plugins Repository," 2020) umgesetzt. Dabei wurde für jedes Segment das zugehörende Teil-Netzwerk ermittelt. Das Grösste Netzwerk erhielt dabei die Gruppen-ID 0. Somit konnten danach alle Segmente, welche eine Gruppe-ID grösser als 0 hatten, gelöscht werden.

2.9.5 Umwandeln der OSM Attribut in die benötigte Form

Die meisten für die Simulation benötigten Attribute waren in den OSM Daten schon enthalten und mussten nur in die benötigte Form umgewandelt werden.

2.9.5.1 Attribut basetype

Das Attribut basetype entspricht dem Nutzungsstreifentyp im GIP. Es sind mehrere Typen pro Segment aus einer Wertedomäne möglich.

- *1*: Fahrbahn
- *2*: Radweg
- *4*: Schienenweg
- *5*: Verkehrsinsel
- *6*: Stiege (Treppe)
- usw.

In der Simulation wurde nur der Typ 6 (*Stiege*) berücksichtigt. Somit wurde aus den OSM Daten dieser Wert gesetzt, falls das Tag `highway == steps`.

2.9.5.2 Attribut brunnel

brunnel steht für **Br**ücke / T**unnel**. Dabei steht *1* für Brücke und *2* für Tunnel. Dieses konnte über die beiden Tags *bridge* und *tunnel* vergeben werden (siehe Abbildung 7).

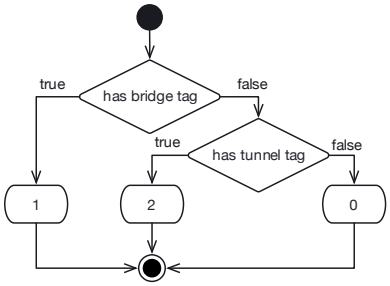

Abbildung 7: Attribut brunnel

2.9.5.3 Attribute oneway_ft und oneway_tf

Mit *oneway_ft* und *oneway_tf* wird die Verfügbarkeit des Weges in eine Richtung eingeschränkt. Wenn es sich um eine Einbahnstrasse handelt, ist jeweils einer der beiden Werte *1*. Einbahnstrassen werden in OSM über das Tag *oneway* gekennzeichnet. Zusätzlich existiert für Velos das Tag *oneway:bicycle*. Für beide bedeutet der Wert -1 Einbahn in umgekehrt Richtung zur gezeichneten Linie.

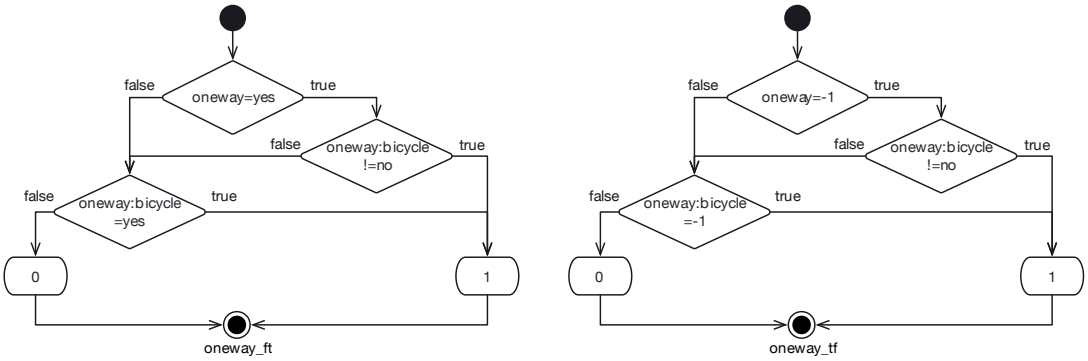

Abbildung 8: Attribute oneway_ft und oneway_tf

2.9.5.4 Attribute restric_ft und restric_tf

In restric_ft und restric_tf sind Einschränkungen bezüglich des Veloverkehrs definiert. Dabei gelten folgende Werte:

- *0*: keine Einschränkungen
- *1*: Fussgänger erlaubt und somit ist auch Veloschieben erlaubt
- *2*: Velo und Fussgänger nicht erlaubt

Bei Einbahnen gelten diese Einschränkungen nur für eine Richtung. Daher wurden zuerst anhand des Einbahnresultates die Einschränkung vergeben (siehe Abbildung 9).

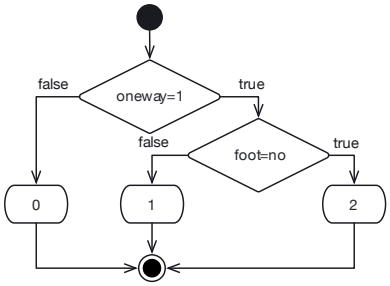

Abbildung 9: Attribut restric nach oneway

Danach wurden generelle Einschränkungen, welche entweder explizit über das *bicycle* Tag definiert sind oder über den Wegtype wie z.B. Reitweg vergeben sind überprüft (siehe Abbildung 10).

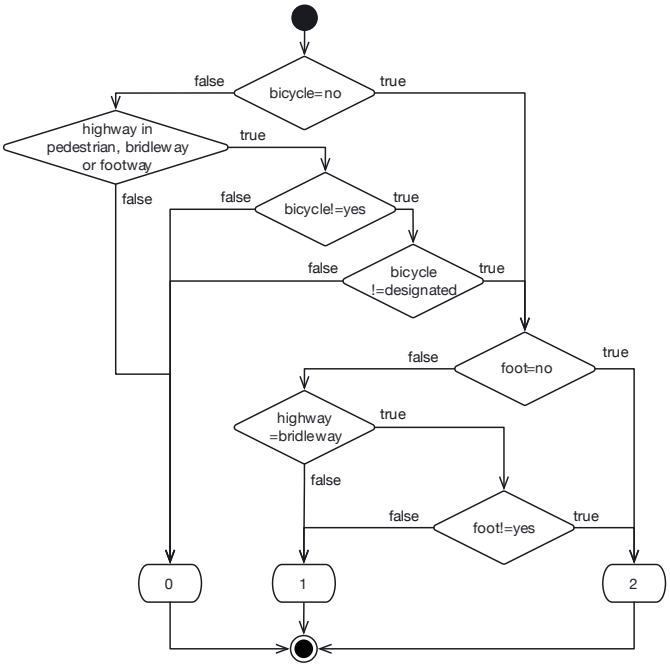

Abbildung 10: Attribut restrict für beide Richtungen

2.9.5.5 Attribute bic_inf_ft und bic_inf_tf

In den Attribute *bic_inf_ft* und *bic_inf_tf* wurde die Art der Veloinfrastruktur definiert, dabei sind folgende Werte möglich:

- *bicycle_way*: physisch getrennte Velospur oder Veloweg
- *mixed_way*: gemischter Weg für Fussgänger und Velofahrer

- *bicycle_lane*: Velospur neben der Fahrspur für Motorfahrzeuge
- *bus_lane*: Busspur welche von Velofahrern genutzt werden dürfen
- *no*: keine spezielle Veloinfrastuktur

In OSM sind mehrere Tags für die Veloinfrastruktur zuständig. Dazu gehört *highway*, *cycleway*, *cycleway:both*, *cycleway:right*, *cycleway:left* und *bicycle*. In folgendem Diagramm ist die Zuweisung ersichtlich, dabei wurde die Darstellung vereinfacht und die vier *cycleway* Tags zusammengefasst.

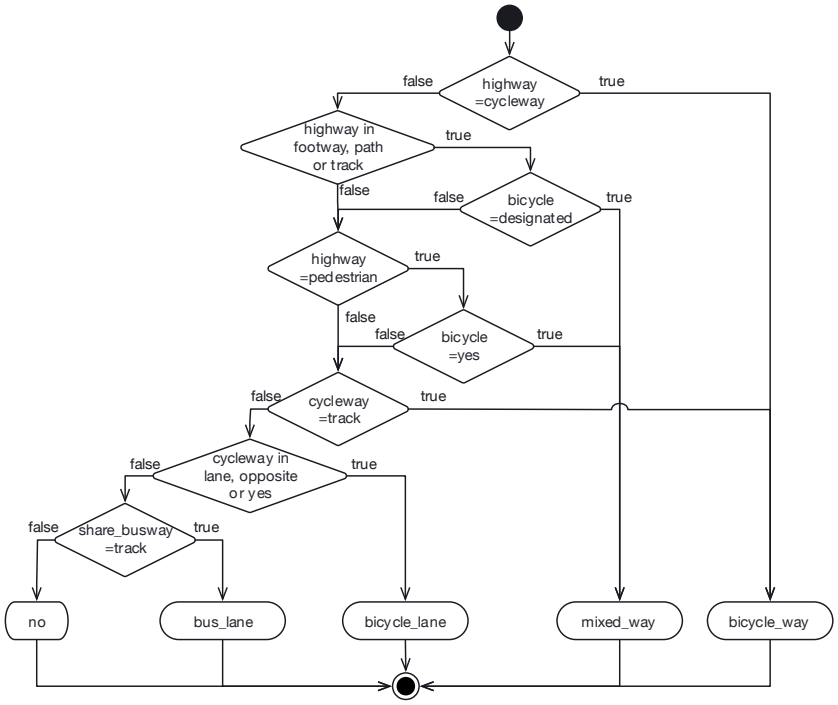

Abbildung 11: Attribut bic_inf (vereinfachte Darstellung)

2.9.5.6 Attribute d_route_ft und d_route_tf

Die beiden Attribute *d_route_ft* und *d_route_tf* wurden jeweils identisch vergeben und stehen für ausgewiesene Velorouten. Dies war über das im Import ergänzte *network* Tag möglich.

Tabelle 3: Mapping von network zu d_route

d_route	***network***
national	ncn
regional	rcn
local	lcn
no	network Tag nicht vorhanden

2.9.5.7 Attribut road_categ

Das Attribut *road_categ* steht für die Strassenkategorie, dabei besteht folgende Wertedomäne:

- *primary*: höchste Strassenkategorie, welche Velos befahren können
- *secondary*: Nebenstrassen
- *residential*: Gemeindestrassen, die nicht zu einer der beiden höheren Kategorien gehören
- *service*: alle Arten von Zufahrtsstrassen
- *calmed*: Verkehrsberuhigte Strassen, z.B. Begegnungszone
- *no_mit*: Strassen ohne motorisierten Verkehr
- *path*: Wege ohne Gebotszeichen für nicht motorisierten Verkehr

Mit der Ausnahme von *no_mit*, liessen sich alle Werte via das OSM Tag *highway* abfüllen (siehe Tabelle 4). Wie in Abbildung 12 ersichtlich mussten für no_mit mehrere OSM Tags verwendet werden.

Tabelle 4: Mapping von highway zu road_categ

road_categ	***highway***
primary	primary, primary_link
secondary	secondary, secondary_link, tertiary, tertiary_link
residential	residential
service	service, unclassified, track
calmed	living_street, pedestrian
no_mit	Siehe Abbildung 12
path	path

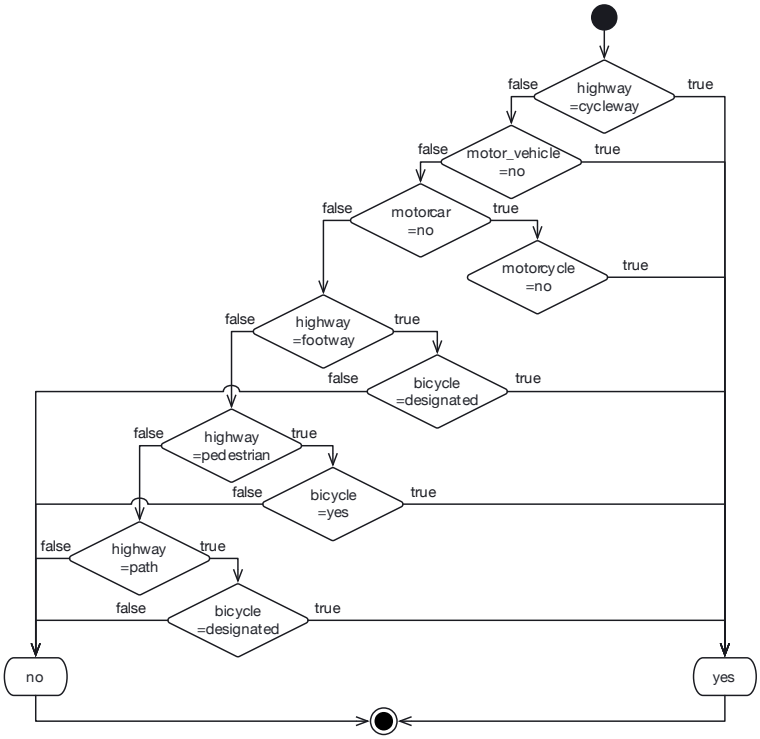

Abbildung 12: Wert no_mit für Attribut road_categ

2.9.5.8 Attribute max_sp_ft und max_sp_tf

Attribute *max_sp_ft* und *max_sp_tf* stehen für die zulässige Höchstgeschwindigkeit. Diese konnte aus der durch *osm2pgrouting* erstellten Tabelle übernommen werden und befand sich in den beiden Felder *maxspeed_forward* und *maxspeed_backward*.

2.9.5.9 Attribut rails

Das rails Attribut steht für Schienen auf der Strasse und es sind die beiden Werte *yes* und *no* möglich. Dies konnte über des OSM Tag *railway* vergeben werden, sobald dieses existiert wird *rails* auf *yes* gesetzt, ansonsten *no*.

2.9.5.10 Attribute parking_ft und parking_tf

Die Attribute *parking_ft* und *parking_tf* stehen für Parken auf der Strasse und es sind die beiden Werte *yes* und *no* möglich. In OSM gibt es mehrere Tags für Parkmöglichkeiten.

- parking:both
- parking:lane:both

- parking:right
- parking:lane:right
- parking:left
- parking:lane:left

Die Werte wurden, wie im Diagramm in Abbildung 13 ersichtlich, vergeben. Dabei wurden alle 6 *parking* Tags zusammengefasst.

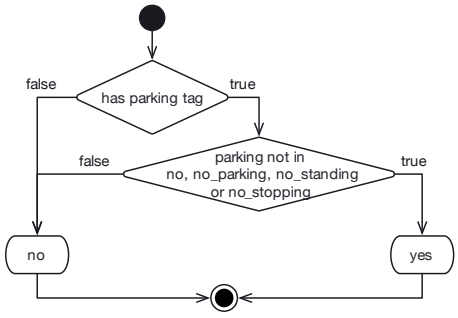

Abbildung 13: Parking Attribut (vereinfacht)

2.9.5.11 Attribut surface

Das *surface* Attribut beschreibt den Strassenbelag. In OSM erfüllt das *pavement* Tag die selbe Funktion, wobei die Werte nicht deckungsgleich sind. Die Werte wurden gemäss der Tabelle 5 übernommen

Tabelle 5: Mapping von pavement zu surface

surface	***pavement***
asphalt	asphalt, chipseal, concrete, concrete:plates
gravel	gravel, fine_gravel, compacted
soft	unpaved, ground
cobble	paving_stones, sett, unhewn_cobblestone, cobblestone, brick, paving_stones, pebblestone

2.9.5.12 Attribute n_lanes_ft und n_lanes_tf

Die Attribute *n_lanes_ft* und *n_lanes_ft* definieren die Anzahl Spuren pro Richtung. Die Anzahl Spuren wurden aus den vier OSM Tags *lanes*, *lanes:forward*, *lanes:backward* und *lanes:both_ways* berechnet. Dabei wurden im Falle einer Einbahnstrasse der Wert aus *lanes* an der entsprechenden Richtung zugewiesen. Falls es keine Einbahn war, wurde der Wert zuerst durch 2 geteilt und aufgerundet auf die beiden Richtungen

verteilt. Danach wurden die drei Tags *lanes:forward*, *lanes:backward* und *lanes:both_ways* ausgewertet, um die Werte entsprechend anzupassen.

2.9.6 Ergänzen fehlender Attribute durch zusätzliche Datenquellen

2.9.6.1 Attribute gradient_f und gradient_t

Die Attribute *gradient_f* und *gradient_t* stehen für die Neigungskategorie gemäss der Klassifizierung für Steigung und Gefälle (siehe Tabelle 6).

Tabelle 6: Klassifizierung Steigung und Gefälle

Steigung	<= -12%	<= -6%	<= -3%	<= -1.5	< 1.5	< 3	< 6	< 12	>= 12
Klasse	-4	-3	-2	-1	0	1	2	3	4

Weder Steigung/Gefälle noch Höheninformationen waren in den OSM Strassendaten enthalten. Um diese zu ergänzen, wurde ein digitales Geländemodell verwendet, dabei gibt es zwei Varianten. Das digitales Oberflächenmodell (DOM) und das Digitales Terrainmodell (DTM). Das DOM repräsentiert die Erdoberfläche samt allen darauf befindlichen Objekte (inkl. Bewuchs und Bebauung). Das DTM bildet die Topografie der Erdoberfläche ohne Bewuchs und Bebauung ab. Da die Strassen nicht immer ganz exakt zu den Höhenmodellen passen, kommt es beim DOM in Häuserschluchten oder bei Strassen entlang von Bäumen zu Fehlern. Dies ist in der Abbildung 14 gut zu erkennen, wo im DOM ein Haus erwischt wird. Deswegen wurde mit dem DTM die Steigung bestimmt.

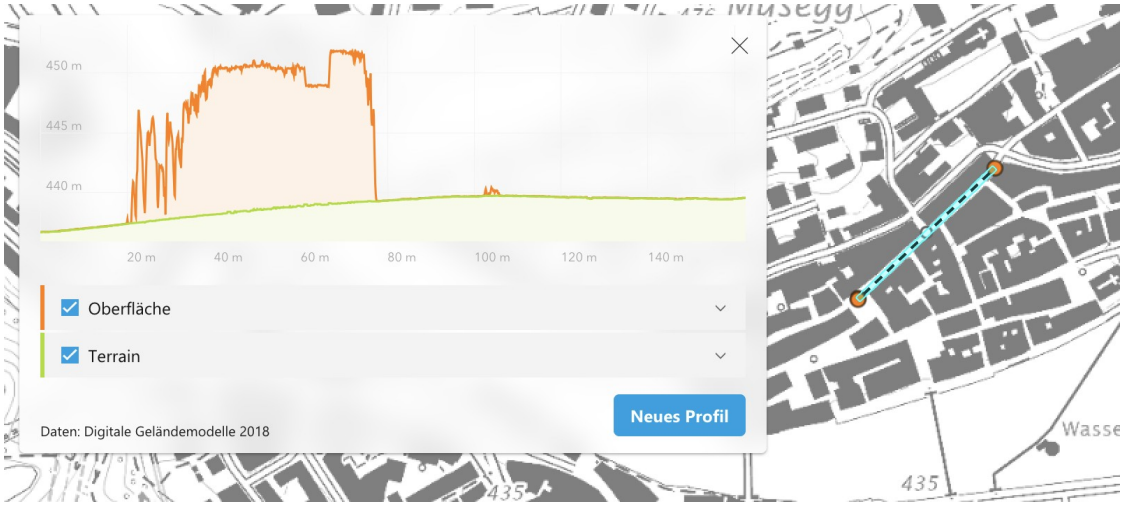

Abbildung 14: Höhenprofil Altstadt Luzern (Weggisgasse)

Mit dem DTM mussten jedoch Brücken weggelassen werden, da diese nicht im Terrain enthalten sind. Dies ist kein Problem, da die meisten, für Velos zugänglichen Brücken, keine oder nur geringe Steigung besitzen.

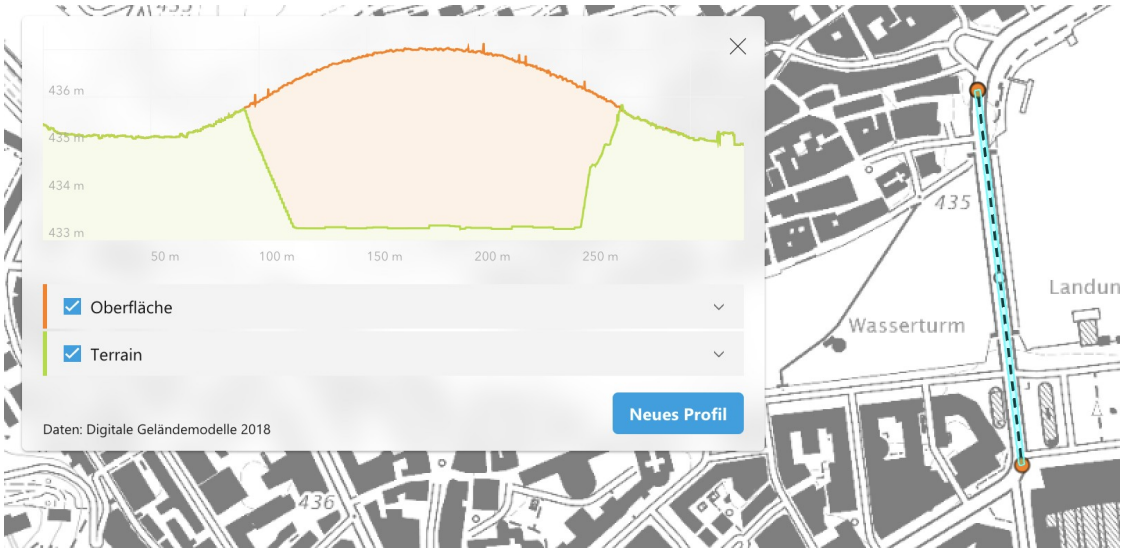

Abbildung 15: Höhenprofil Seebrücke Luzern

Neben den Brücken, mussten aus naheliegenden Gründen auch Tunnels ignoriert werden. Dies wurde über das Attribut *brunnel* bewerkstelligt. Für die Berechnung der Steigung wurde jeweils die Höhe des Startpunktes und Endpunktes des Strassensegment verwendet.

2.9.6.2 Attribut width_lane

Das Attribut *width_lane* steht für die Spurbreite in Metern. In den OSM Daten existiert das Tag *width.* Dieses war nur in wenigen Fällen befüllt. Die Spurbreite konnte mit der Hilfe der Bodenbedeckung ("Amtliche Vermessung - Kanton Luzern," 2023) eruiert werden. Dabei wurden die Daten nach dem Attribut *ART* und den beiden folgenden Werten gefiltert:

- *1:* Strasse, Weg
- *2:* Trottoir

Die OSM Strassen überlappten nicht exakt die Bodenbedeckungsdaten (siehe Abbildung 16). Deswegen wurde für die Breitenberechnung jeweils das Element mit der längsten Überlappung gewählt. Überlappungen von weniger als 2 Meter wurden ignoriert.

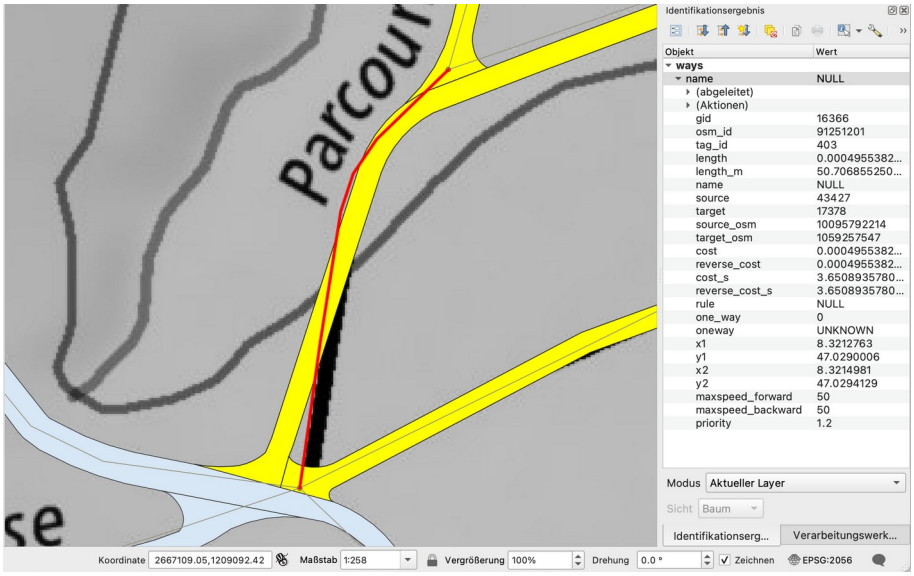

Abbildung 16: OSM Strassensegment mit Bodenbedeckung (Birregwald)

Für die Strassenbreite (w) wurde folgende Berechnung verwendet, wobei P für Perimeter steht und A für die Fläche (whuber, 2012):

$$w = \frac{\left(P - \sqrt{P^2 - 16 * A}\right)}{4}$$

Das Resultat wurde verworfen, falls die Breite grösser oder gleich war, wie die Länge des Strassensegments. Dies trat etwa bei Plätzen auf, nicht aber bei spiralförmigen Wegen wie in der Abbildung 17 ersichtlich. Wobei auch die starke Steigung gut im *gradient* Attribut erkennbar ist.

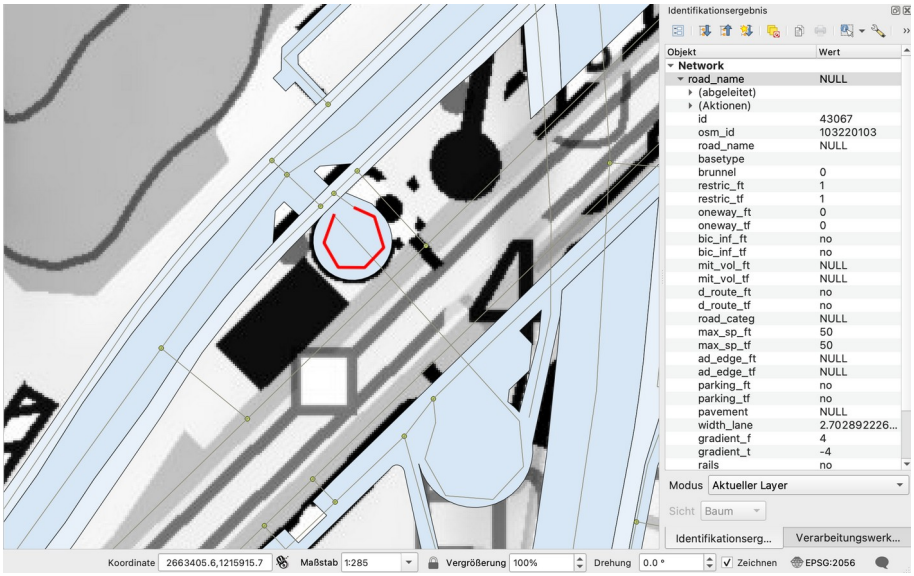

Abbildung 17: Spiralweg zu Unterführung (Emmenbrücke Kapf)

2.9.6.3 Attribute mit_vol_ft, mit_vol_tf, ad_edge_ft, ad_edge_tf und land_use

Die fünf Attribut *mit_vol_ft*, *mit_vol_tf*, *ad_edge_ft*, *ad_edge_tf* und *land_use* sind eigentlich in der Simulation vorgesehen. Da diese in den original Daten vom Bicycle model nicht befüllt waren, wurde darauf verzichtet diese zu befüllen.

2.10 Intersections Daten

Bei «Intersections» handelt es sich um einen Punktdatensatz mit allen Strassenkreuzungen. Abgesehen von der Geometrie und einer «ID» enthält der Datensatz keine Attribute.

intersections	
id	NUMERIC(10,0)
wkb_geometry	geometry

Abbildung 18: Attribute von intersections

In der Originalstudie stammten diese Daten vom GIP. Für Luzern wurden die Kreuzungen aus den «network» Daten generiert. Dabei wurden Tunnels und Brücken getrennt behandelt, damit es nicht zu nicht existierenden Kreuzungen über einem Tunnel, respektive unter einer Brücke kommt. Dazu wurden zuerst alle nach dem *brunnel* Attribut ausgewählten Linien aufgelöst und das Multipart Resultat in Singlepart umgewandelt. Danach konnten die Linienschnittpunkte bestimmt werden. Dies führte zu mehreren Punkten pro Kreuzung und musste noch bereinigt werden (spatialthoughts, 2020). Details sind am Ende des Anhang E ersichtlich.

2.11 Bicycle Model Version 2.1

Damit die Simulation mit den Daten für Luzern funktionierte, waren mehrere Anpassungen am Modell nötig. Dies hatte zwei Hauptgründe. Erstens waren die 9 Salzburger Velozähl-Stationen fix im Modell und zweitens waren die Luzerner Daten nicht ganz identisch. Die Altersklasse gingen nur bis *over_90*. In Salzburg war die höchste Altersklasse *over_100*. Die Daten für den Erwerbsstatus fehlten komplett und wurden jeweils aus schweizweiten Daten erstellt.

Zusätzlich wurden noch ein paar kleine Codeanpassungen für die aktuelle Gama Version 1.9 vorgenommen, z.B. wurde in der Funktion *save* der Parameter *type* auf

format umgestellt. Weiter wurden noch zwei potentielle *null pointer exception* verhindert, respektive abgefangen.

Die Spalte mit den observierten Durchfahrten für eine Station wurden im Model 2.0 fix einer Station zugeordnet. Um dies flexibler zu gestalten, wurde die Datei *hourly_counts_stations.csv* inklusive Kopfzeile eingelesen, um danach in dieser Zeile nach dem passenden Stationsnamen aus der *counting_stations.shp* Datei zu suchen. Diese Datei wurde noch um das optionale Feld *key* ergänzt. Dieses ermöglicht im Feld *stat_name* eine besser lesbare Bezeichnung.

Die Zählstationen lagen jeweils nicht exakt auf dem Strassennetz, sondern zwischen den Spuren (siehe Abbildung 19). Deswegen wurden alle Stationen um einen 12 Meter Puffer ergänzt.

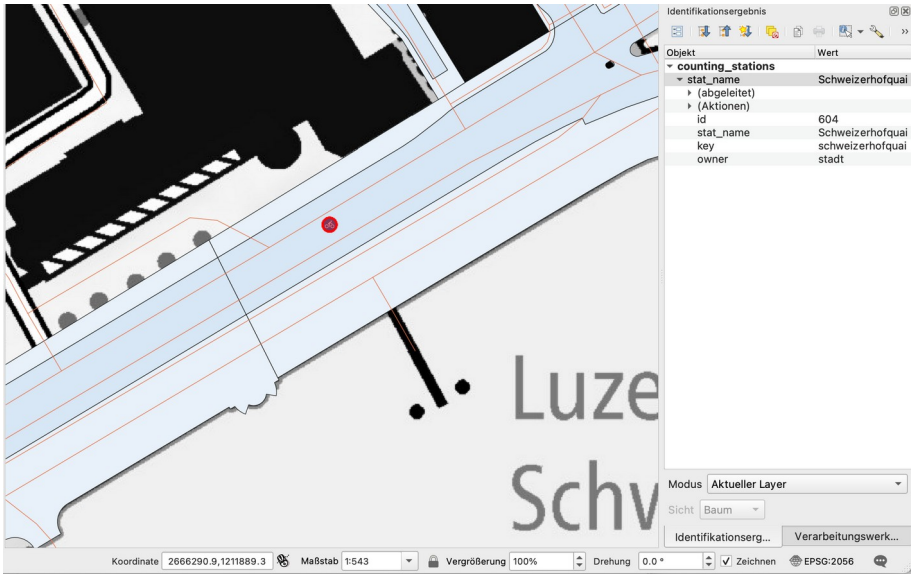

Abbildung 19: Zählstation Schweizerhofquai zwischen den Spuren

Um die beiden fehlenden Altersklassen zu kompensieren, wurde einfach die höchste Altersklasse um ein minimales maximal Alter von 104 ergänzt. Damit blieb es bei Salzburger *over_100* bei 100 ≤ 104, das *over_90* in Luzern wurde zu 90 ≤ 104.

2.11.1 Erwerbsstatus

Für den Erwerbsstatus wurde die *action prepareHomes* ergänzt. Diese wurde jeweils in der Initialisierung der Simulation ausgeführt und ergänzte die *Homes* Daten um die Erwerbsstatus Daten. Das Ergebnis wurde pro Simulation als *Shape* Datei gespeichert.

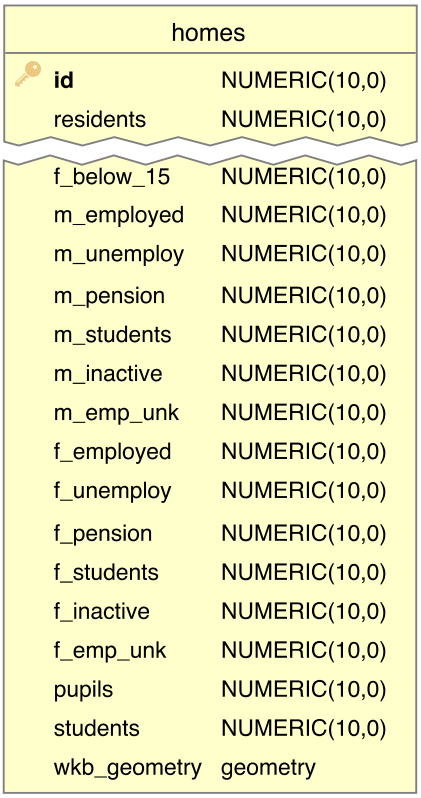

Abbildung 20: Erwerbsstatus Attribute von homes

Zuerst wurde pro Altersklasse ein Array erstellt, welches die ID jedes Home Features in der Anzahl des Altersklasse-Werts enthielt.

Tabelle 7: Beispiel Altersklassen Werte in Homes

Id	*f_30_34*	*f_35_39*
42	0	5
43	5	3
44	5	2

Aus den Werten in der Beispiel Tabelle 7 entstanden folgende Einträge in den beiden Arrays:

- *f_30_34*: 43, 43, 43, 43, 43, 44, 44, 44, 44, 44
- *f_35_39*: 42, 42, 42, 42, 42, 43, 43, 43, 44, 44

Diese Arrays wurden für jeden Erwerbsstatus neu gemischt, um daraus die jeweilige Anzahl IDs gemäss *employment_status_probabilities.csv* auszuwählen. Der entsprechende Status wurde für die zur ID gehörenden Features jeweils um eins erhöht. Falls dabei die maximale Anzahl Frauen/Männer (*f_tot*/*m_tot*) schon erreicht war, wurde das

nächste Feature gewählt. Beispielsweise könnten man annehmen, dass nur die drei Features aus der Tabelle 7 exisiteren und dass der *f_employed* Wert um die beiden Altersklassen *f_30_34* und *f_34_39* ergänzt werden soll. Weiter könnte man annehmen, dass in der *employment_status_probabilities.csv* für *f_30_34* der Wert 70% steht und für *f_35_39* der Wert 80%. Die zufällig gemischten Arrays könnten wie folgt aussehen (die ausgewählten IDs sind jeweils fett dargestellt):

- *f_30_34*: **44, 43, 44, 43, 43, 44, 43**, 44, 43, 44 (70% Ausgewählt)
- *f_35_39*: **42, 42, 44, 43, 42, 44, 42, 43**, 43, 42 (80% Ausgewählt)

Die Tabelle 8 zeigt die Werte für *f_employed* vor und nach abarbeiten der beiden Altersklassen *f_30_34* und *f_34_39*.

Tabelle 8: f_employed vor und nach abarbeiten der Altersklasse f_30_34 und f_35_39

Id	f_employed vorher	Zuwachs	f_employed nachher
42	5	+4	9
43	0	+6	6
44	10	+5	15

Es wurden jeweils für beide Geschlechter die Werte, wie oben beschrieben, in folgender Reihenfolge abgearbeitet:

1. *employed*
2. *pupils*
3. *students*
4. *unemployed*

Für *unemployed* wurde bei der Totalüberprüfung auch die Werte aus *pupils* und *studens* dazu gezählt. Ein Schüler oder Student kann zwar gleichzeitig auch als *employed* eingestuft werden, jedoch nur in den wenigsten Fällen als *unemployed*.

Die beiden Status *inactive* und *pension* wurden am Schluss auf die Differenz der ersten vier Werte zum Total vergeben. Dabei wurden für die Einteilung in *inactive* oder *pension* das Verhältnis zwischen den Bewohnern unter 65 Jahren und den restlichen Bewohnern genommen.

2.11.2 Analyse

Für ein Vergleich mit der Version 2.0 wurden die selben Werte mit 200 Durchläufen berechnet. Da mit den Erwerbsstatus eine zusätzliche Zufallsverteilung dazu kam, wurde auch der Variationskoeffizient zu den 200 Simulationen und zusätzlich noch für

jeweils 5, 10, 15, 25, 50 und 100 Simulationen berechnet. Dabei wurde das selbe R-Script (Anhang G) verwendet.

3 Ergebnisse

3.1 Datenvergleich «Bicycle model v2» und Studie

Die gemessenen Durchfahrten aus der Datei *hourly_counts_stations.csv* liessen sich nicht zuverlässig überprüfen. Anhand der Grafik mit den täglichen Durchfahrten auf Seite 15 in der Studie (Kaziyeva et al., 2021a, p. 15) und den Resultaten in Tabelle 9, ist ersichtlich, dass die Zahlen für die Station *Schanzlgasse* abweichen. In der Grafik wurden etwas über 2'500 Velofahrer gezählt. Aus den mitgelieferten Daten (Kaziyeva et al., 2021b) ergibt sich jedoch ein Total von täglich 7'272 Velofahrern.

Neben der Abweichung der gemessenen Velos an der *Schanzlgasse* wurden im Modell weitere Abweichungen gefunden. In der Tabelle «Durations by activity type» (Kaziyeva et al., 2021a, p. 19) wichen zwei Werte ab. Gemäss dieser Tabelle bestand für *home* keine *closing time*, im GAML Code wurde jedoch 23:00 Uhr (Zeile 983) verwendet. Für *shop* wurde nur eine maximale Dauer von 2 Stunden (Zeile 1010), anstelle von 3 Stunden verwendet. Eine weitere Abweichung wurde in der Tabelle «School durations by age» (Kaziyeva et al., 2021a, p. 20) gefunden, im Alter von sechs bis sieben ist die Schuldauer nur 5 Stunden (Zeile 987) anstelle der 6. In der Datei *time_probabilities.csv* konnte nur der Aktivitätstyp *work* überprüft werden (Kaziyeva et al., 2021a, p. 19). Dabei wurde keine Abweichung gefunden. Die Daten der drei Dateien *activity_probabilities.csv, mode_probabilities.csv* und *work_duration_probabilities.csv* waren vollständig in der Studie enthalten und es konnte keine Abweichung zur Studie gefunden werden.

3.2 Analyse des «Bicycle model v2»

In der Tabelle 9 ist gut ersichtlich, dass die meisten Mittelwerte des Pearson-Korrelationskoeffizient aus 200 Simulationsdurchläufen, nahe an den Werten aus der Studie (Kaziyeva et al., 2021a, p. 16) liegen. Nur der Standort *Schanzlgasse* wich stark vom Studienergebnis ab. Dies war auch der einzige Standort mit einem mittleren p-Wert von über 0.01. Alle anderen Standorte erreichten, wie in der Studie, einen Wert von 0.001 und darunter, wobei abgesehen von der *Ischlerbahntrasse* die Mehrheit klar darunter liegen. An der *Schanzlgasse* wurden über 34 mal mehr Fahrten durch die Zähl-

stelle erfasst, als simuliert. Eine solch hohe Abweichung war an keiner anderen Station zu beobachten. Alle Stationen weisen einen relativ niedrigen Variationskoeffizient auf.

Tabelle 9: Vergleich des Korrelationskoeffizient (r) zwischen Studie und Mittelwert der Simulations Resultate mit 200 Durchläufen und ergänzenden Zahlen

Station	r Studie	r Mittel	p Mittel	Fahrten Mittel	Variations-koeffizient	Erfasste Fahrten	Δ Fahrten
Alterbach	0.76	0.75	0.0000	2'343	0.03	835	-64%
Elisabethkai	0.77	0.78	0.0000	883	0.04	2'303	161%
Giselakai	0.79	0.82	0.0000	1'705	0.03	3'992	134%
Ischlerbahntrasse	0.60	0.64	0.0010	793	0.05	330	-58%
Kaufmann Steg	0.78	0.72	0.0001	1'929	0.03	2'444	27%
Moosbrucker Weg	0.79	0.73	0.0001	613	0.05	197	-68%
Rudolfskai	0.79	0.81	0.0000	3'173	0.02	4'960	56%
Schanzlgasse	0.62	0.49	0.0234	205	0.08	7'272	3447%
Wallnergasse	0.89	0.88	0.0000	1'840	0.03	2'361	28%

Im Boxplot des Korrelationskoeffizienten (Abbildung 21) ist für den Standort *Schanzlgasse* eine hohe Streuung ersichtlich. Zusätzlich ist zu erkennen, dass für den Standort *Kaufmann Steg* in den 200 Durchläufen nie ein Wert von 0.78 erreicht wurde, der Mittelwert lag jedoch über 0.7.

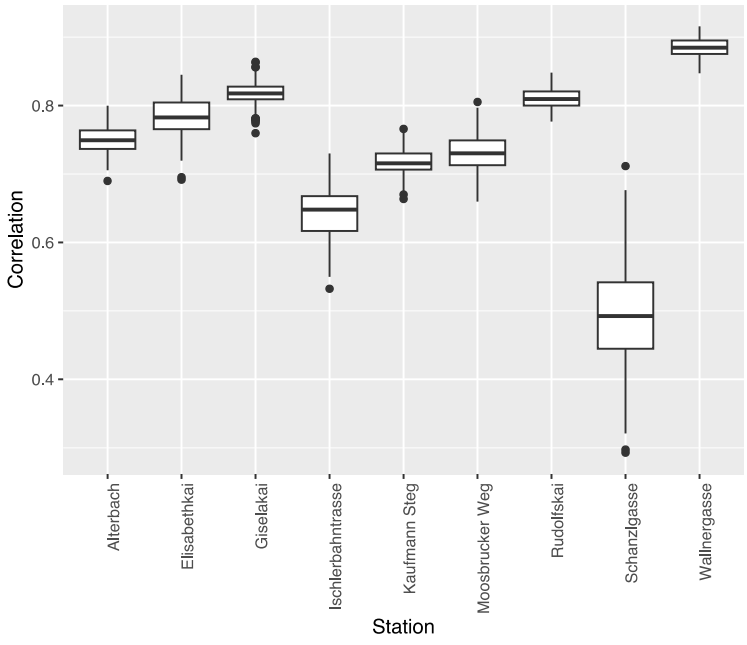

Abbildung 21: Boxplot des Korrelationskoeffizient bei 200 Durchläufen (Salzburg)

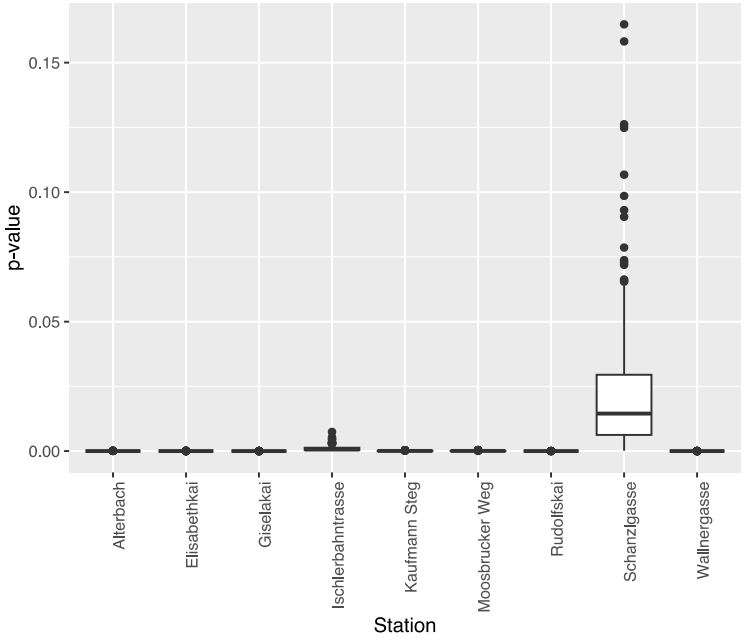

Abbildung 22: Boxplot des p-Wert bei 200 Durchläufen (Salzburg)

Im Boxplot der simulierten täglichen Durchfahrten (Abbildung 23) ist keine grosse Streuung ersichtlich.

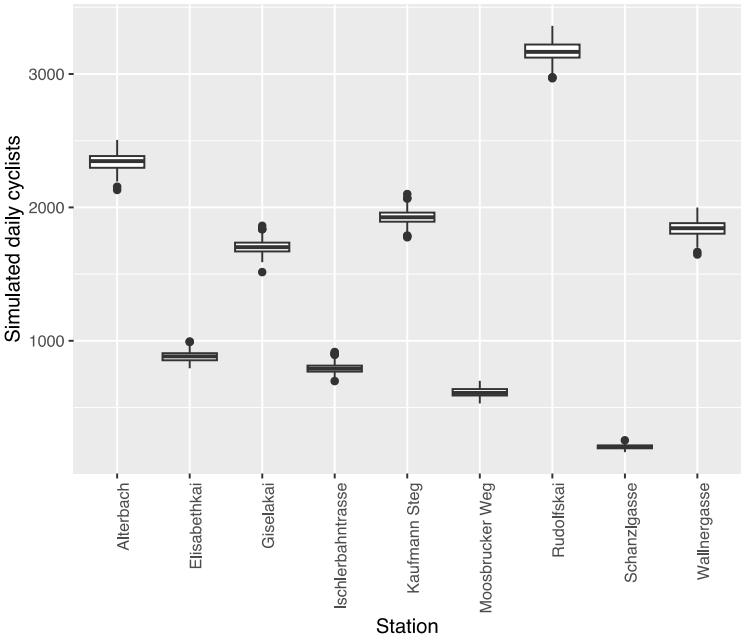

Abbildung 23: Boxplot der täglichen Durchfahrten bei 200 Durchläufen (Salzburg)

Da am Standort *Schanzlgasse* wesentlich weniger Fahrten simuliert als gemessen wurden, ist in Abbildung 24 zuerst ein Boxplot nur mit den simulierten Fahrten dargestellt und erst der zweite enthält auch die gemessenen Fahrten. Dabei wurde sichtbar, dass die gemessenen Fahrten nicht mit der Grafik aus der Studie übereinstimmten (Kaziyeva et al., 2021a, p. 12).

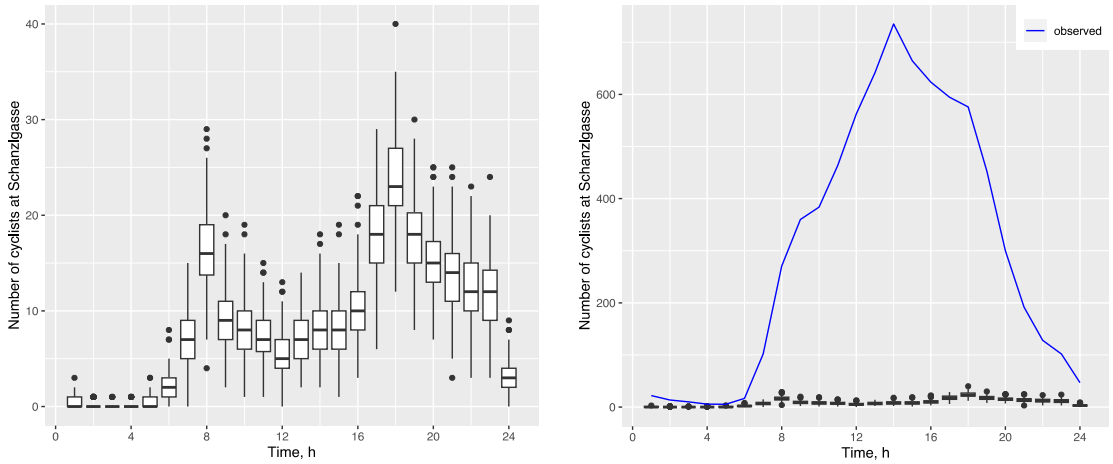

Abbildung 24: Boxplot Schanzlgasse mit und ohne gemessenen Velofahrer

In den Boxplot der restlichen 8 Stationen sind die gemessenen Fahrten jeweils dargestellt (Abbildung 25 bis 28). Bei einem visuellen Vergleich schienen die gemessenen Werte mit den Graphen aus der Studie übereinzustimmen. Jedoch fielen an den meisten Messstellen die simulierten Werte niedriger aus. Nur *Alterbach*. *Ischlerbahntrasse* und *Moosbrucker Weg* schienen ungefähr übereinzustimmen, wobei die letzten beiden schwierig zu beurteilen waren.

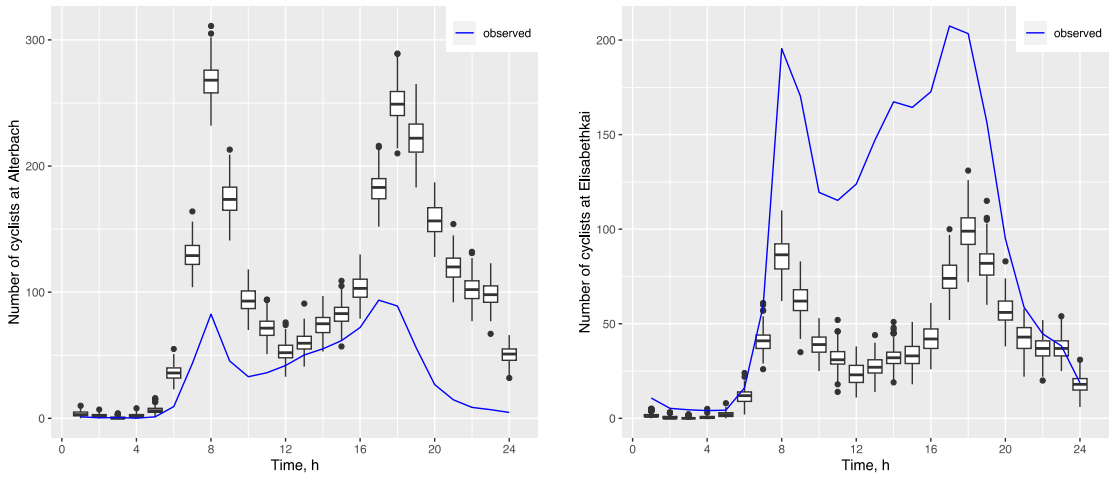

Abbildung 25: Boxplot Alterbach und Elisabethkai mit gemessenen Velofahrer

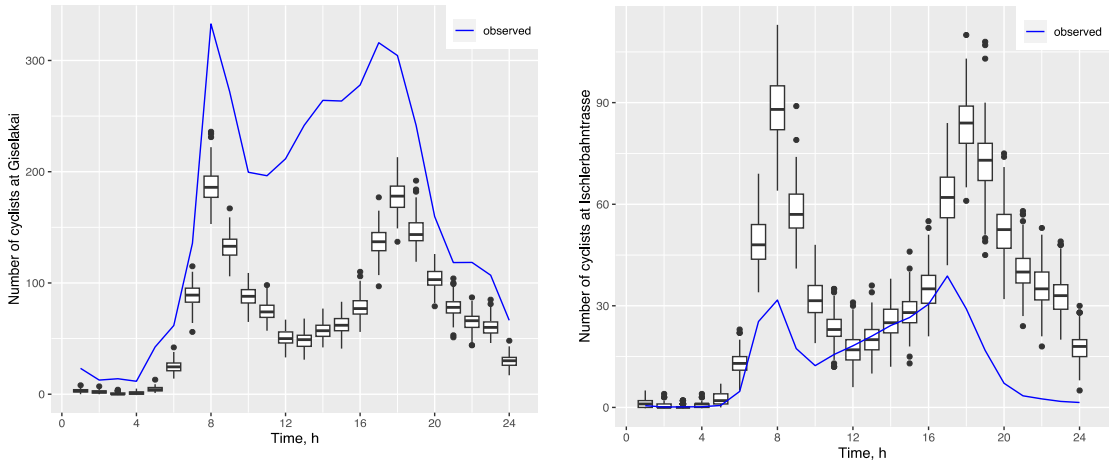

Abbildung 26: Boxplot Giselakai und Ischlerbahntrasse mit gemessenen Velofahrer

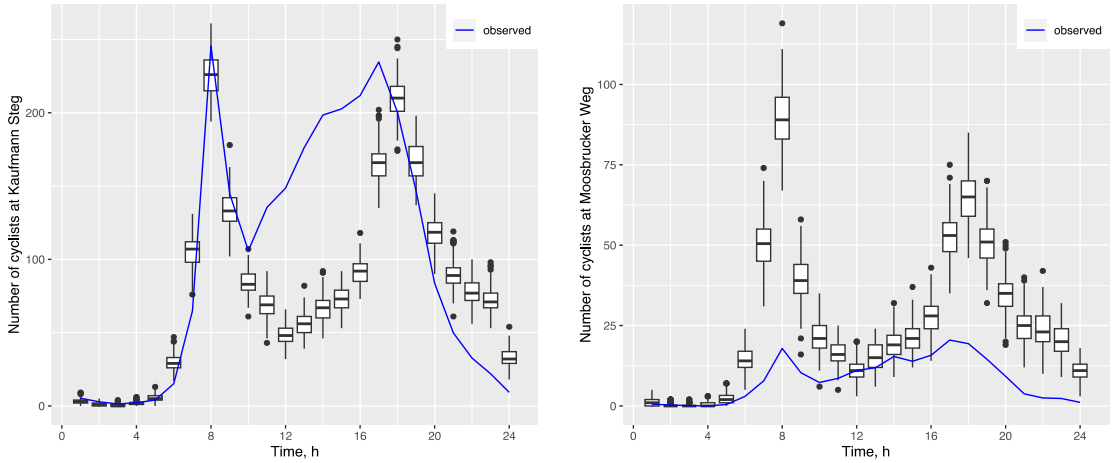

Abbildung 27: Boxplot Kaufmann Steg und Moosbrucker Weg mit gemessenen Velofahrer

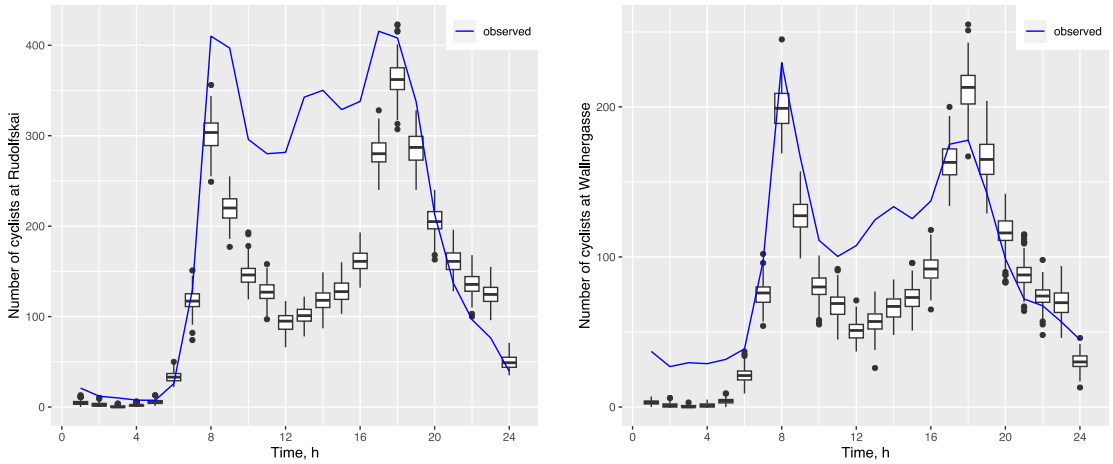

Abbildung 28: Boxplot Rudolfskai und Wallnergasse mit gemessenen Velofahrer

Die Tabelle mit den Variationskoeffizienten wurde analog der Studie von (Lorscheid et al., 2012, p. 51), welche diesen vorschlägt, auf zwei Stellen nach dem Komma gerundet. Es handelte sich jeweils um eigenständige Experimente mit separaten Simulationsdurchläufen. Dabei wurde jeweils ein niedriger Variationskoeffizienten erreicht, wobei die Station *Schanzlgasse* den höchsten Wert aufwies (Tabelle 10). Der Variationskoeffizient stabilisierte sich ab 100 Durchläufen für alle Stationen.

Tabelle 10: Variationskoeffizient für die Salzburger Zählstationen mit unterschiedlichen Simulationsdurchläufen

Station	5	10	15	25	50	100	200
Alterbach	0.03	0.02	0.03	0.02	0.03	0.03	0.03
Elisabethkai	0.03	0.04	0.04	0.05	0.04	0.04	0.04
Giselakai	0.04	0.02	0.02	0.03	0.03	0.03	0.03
Ischlerbahntrasse	0.05	0.05	0.05	0.05	0.05	0.05	0.05
Kaufmann Steg	0.03	0.04	0.03	0.03	0.03	0.03	0.03
Moosbrucker Weg	0.03	0.07	0.07	0.06	0.05	0.05	0.05
Rudolfskai	0.03	0.02	0.02	0.02	0.02	0.02	0.02
Schanzlgasse	0.08	0.08	0.06	0.10	0.09	0.08	0.08
Wallnergasse	0.03	0.02	0.03	0.03	0.03	0.03	0.03

Ein Simulationsdurchlauf dauerte inklusive Initialisierung rund 30 Minuten, wobei die Initialisierung 5 Minuten benötigte. Es liefen jeweils 5 Simulationen gleichzeitig. Pro Simulation wurden rund 187'000 Personen berücksichtigt. Das Strassennetz bestand aus rund 18'500 Objekten.

3.3 Analyse «Bicycle model v2.1» mit Luzerner Daten

In der Tabelle 11 ist gut ersichtlich, dass alle Mittelwerte des Pearson-Korrelationskoeffizient aus 200 Simulationsdurchläufen über 0.6 liegen. Wobei nur gerade der Standort *Inseli* mit 0.66 einen Wert unter 0.7 erreichte. Die Mehrheit der Werte liegt sogar klar über 0.8. Alle Standorte erreichten einen mittleren p-Wert von unter 0.001. Alle Stationen weisen einen relativ niedrigen Variationskoeffizient auf. Die Abweichung zwischen simuliert und erfassten Durchfahrten bewegen sich ihm ähnlichen Rahmen wie in Salzburg (Tabelle 10 wenn man die *Schanzlgasse* ignoriert), jedoch mit einer maximalen von 225% an der *Baselstrasse* ein wenig höher.

Tabelle 11: Mittelwert des Korrelationskoeffizient (r) mit 200 Durchläufen und ergänzenden Zahlen

Station	r Mittelwert	p Mittelwert	Fahrten Mittelwert	Variations-koeffizient	Erfasste Fahrten	Δ Fahrten
Baselstrasse	0.76	0.0001	193	0.10	628	225%
Bleicherstrasse	0.75	0.0001	508	0.07	1'519	199%
Dammstrasse	0.87	0.0000	5'940	0.02	961	-84%
Freigleis Kleinmatt	0.87	0.0000	2'721	0.03	2'197	-19%
Inseli	0.66	0.0006	1'522	0.03	1'213	-20%
Langensandbrücke	0.84	0.0000	4'920	0.02	1'631	-67%
Löwenplatz	0.91	0.0000	4'068	0.02	2'183	-46%
Neustadtstrasse	0.89	0.0000	1'193	0.03	2'265	90%
Palace	0.84	0.0000	4'912	0.02	1'404	-71%
Schweizerhofquai	0.86	0.0000	11'476	0.01	5'630	-51%
Taubenhausstrasse	0.92	0.0000	3'886	0.02	1'992	-49%
Xylophonweg	0.88	0.0000	6'120	0.02	3'009	-51%

Im Boxplot des Korrelationskoeffizienten (Abbildung 29) weisen die beiden Standorte *Baselstrasse* und *Bleicherstrasse* die höchste Streuung aus. Ohne die beiden Ausreisser an der *Baselstrasse* und *Inseli* liegen alle Werte über 0.6.

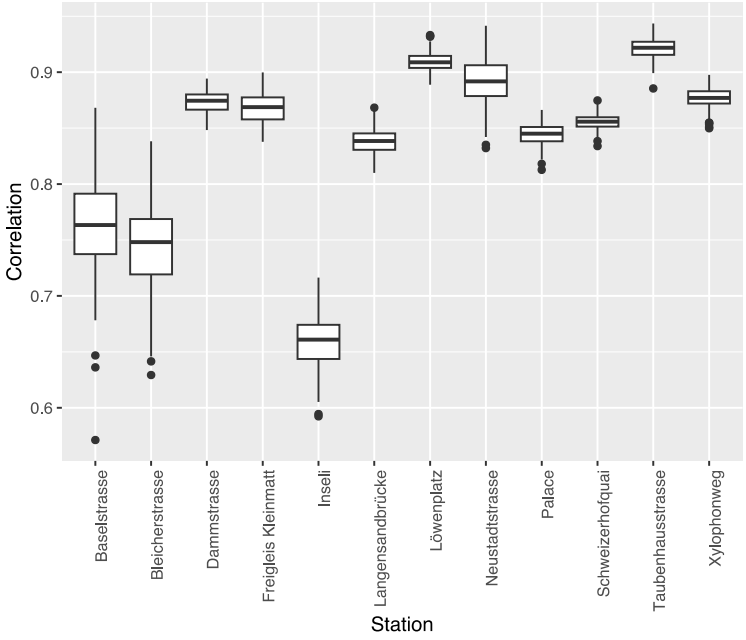

Abbildung 29: Boxplot des Korrelationskoeffizient bei 200 Durchläufen (Luzern)

Der p-Wert am Standort weis an den drei Standorten *Baselstrasse, Bleicherstrasse* und *Inseli* eine höhere Streuung aus (Abbildung 30). Bei den ersten beiden Stationen

handelte es sich jedoch nur um einzelne Ausreisser. An allen anderen Stationen waren die Werte sehr niedrig und es war keine grosse Streuung zu beobachten.

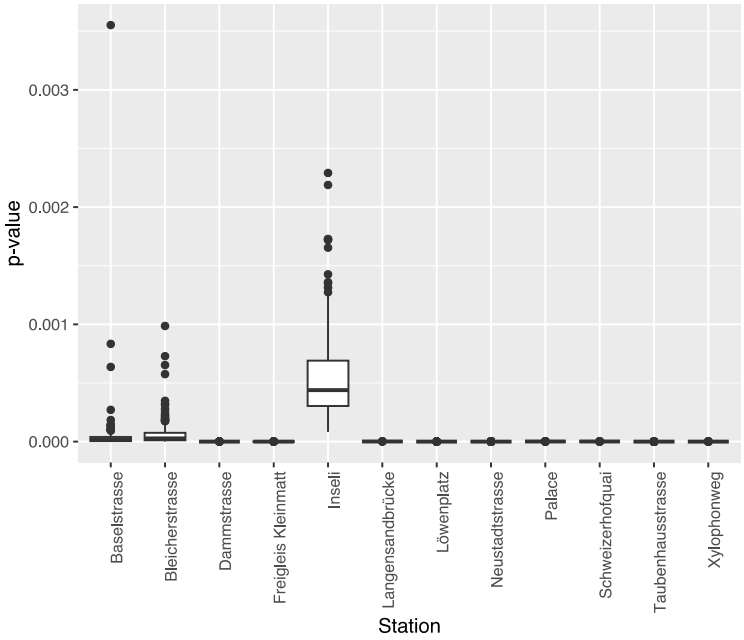

Abbildung 30: Boxplot des p-Wert bei 200 Durchläufen (Luzern)

Im Boxplot der simulierten täglichen Durchfahrten (Abbildung 31) ist keine grosse Streuung ersichtlich.

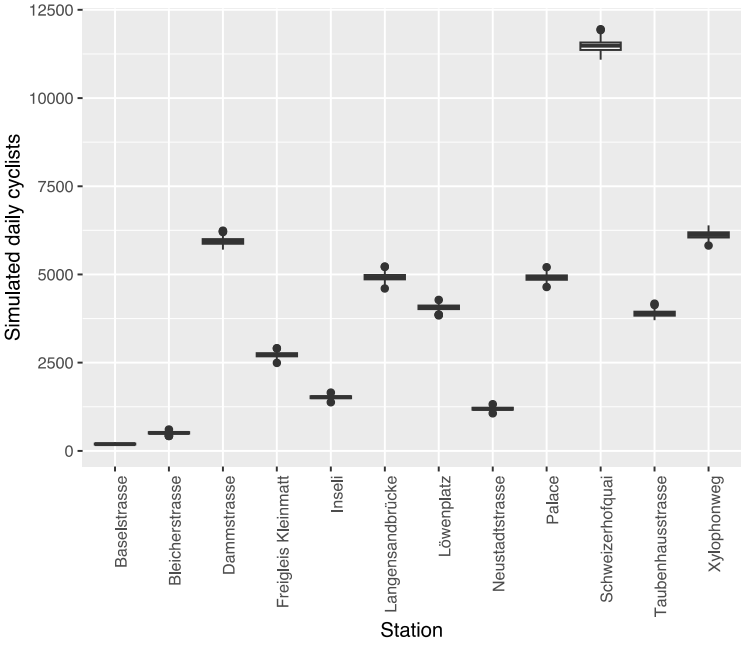

Abbildung 31: Boxplot der täglichen Durchfahrten bei 200 Durchläufen (Luzern)

In den Boxplot der 12 Stationen sind die gemessenen Fahrten jeweils dargestellt (Abbildung 32 bis 37). Die meisten gemessenen Fahrten, wiesen neben den beiden Morgen/Abend Spitzen, auch kurz nach dem Mittag, eine leichtes Mehraufkommen der Velofahrten auf. Diese Erhöhung ist in den simulierten Fahrten nicht ersichtlich. Dies trifft jedoch auch schon auf Salzburg zu, wie z.B. in Abbildung 28 für Rudolfskai und Wallnergasse.

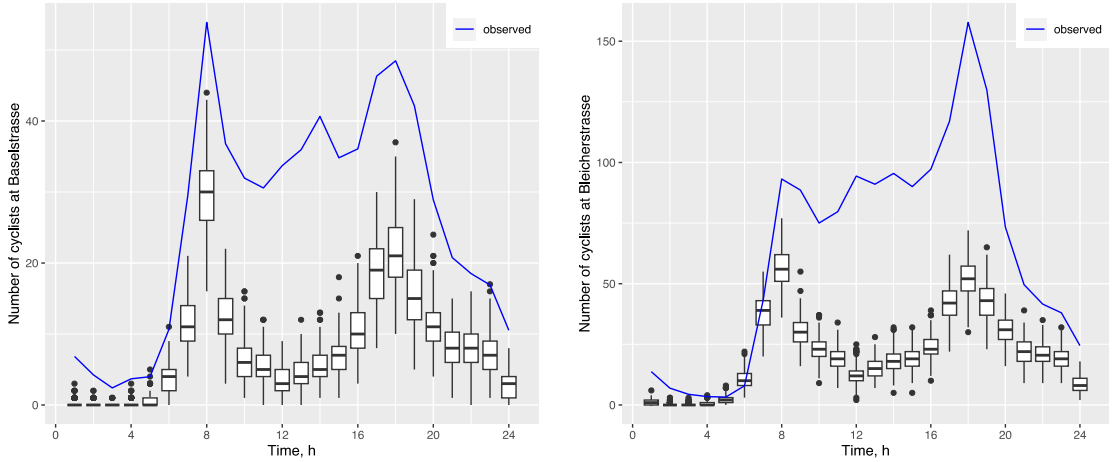

Abbildung 32: Boxplot Baselstrasse und Bleicherstrasse mit gemessenen Velofahrer

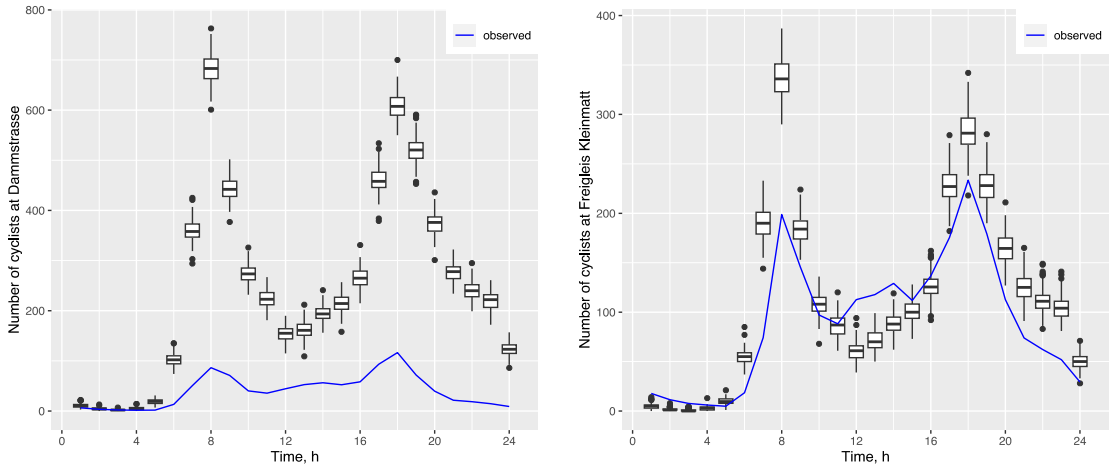

Abbildung 33: Boxplot Dammstrasse und Freigleis Kleinmatt mit gemessenen Velofahrer

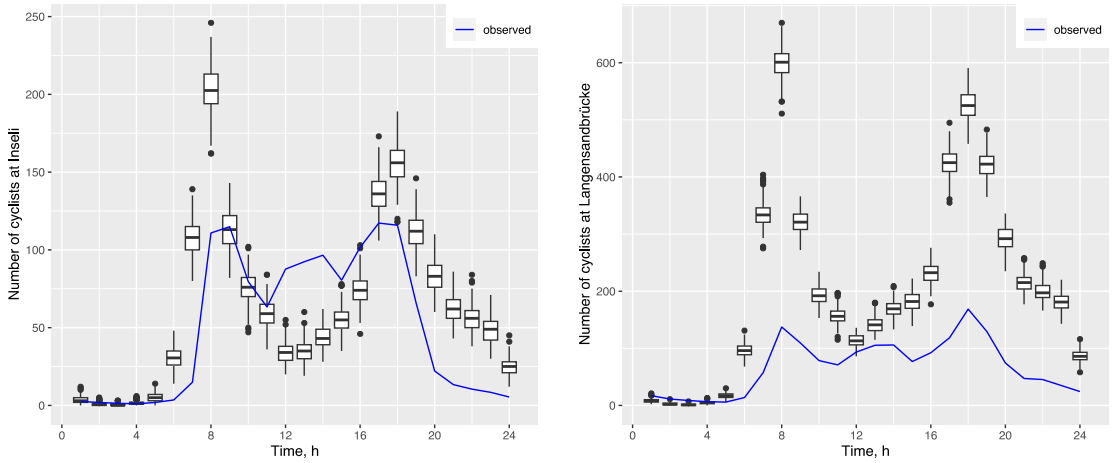

Abbildung 34: Boxplot Inseli und Langensandbrücke mit gemessenen Velofahrer

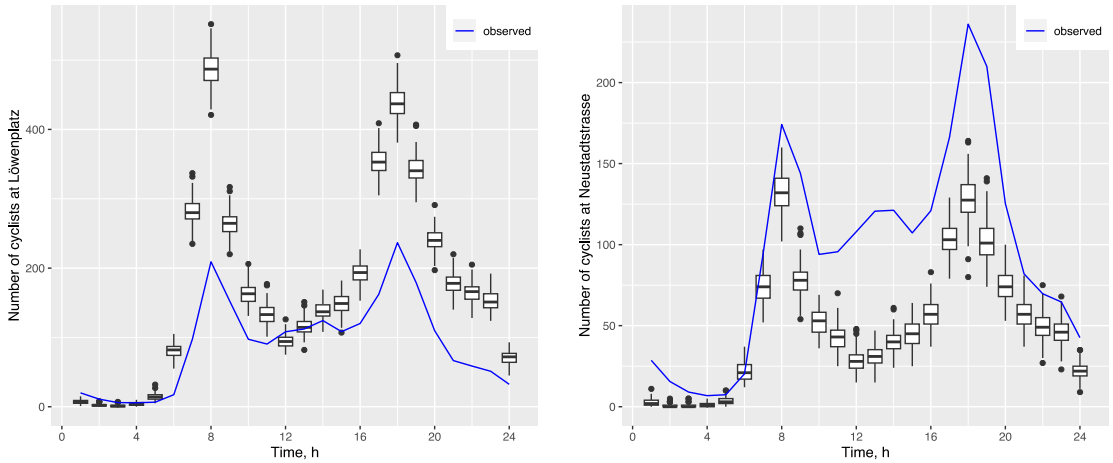

Abbildung 35: Boxplot Löwenplatz und Neustadtstrasse mit gemessenen Velofahrer

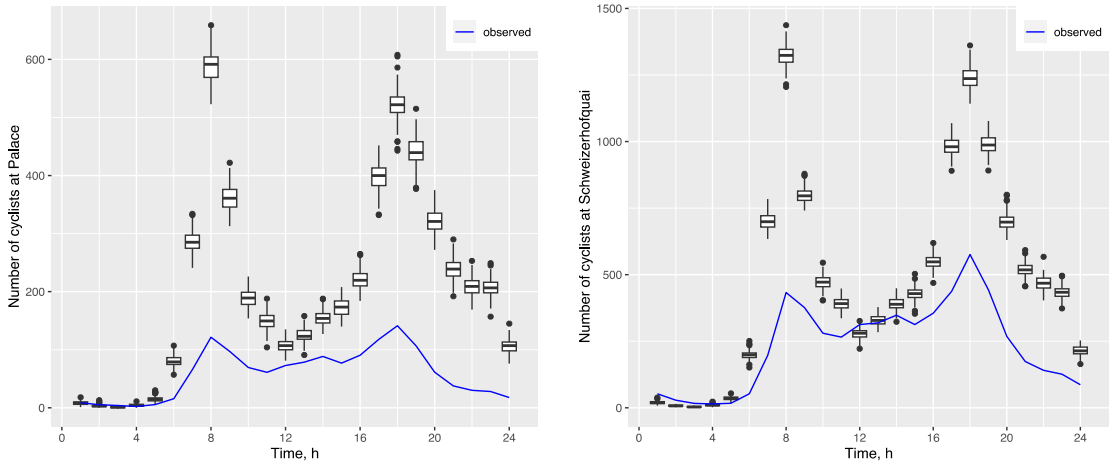

Abbildung 36: Boxplot Palace und Schweizerhofquai mit gemessenen Velofahrer

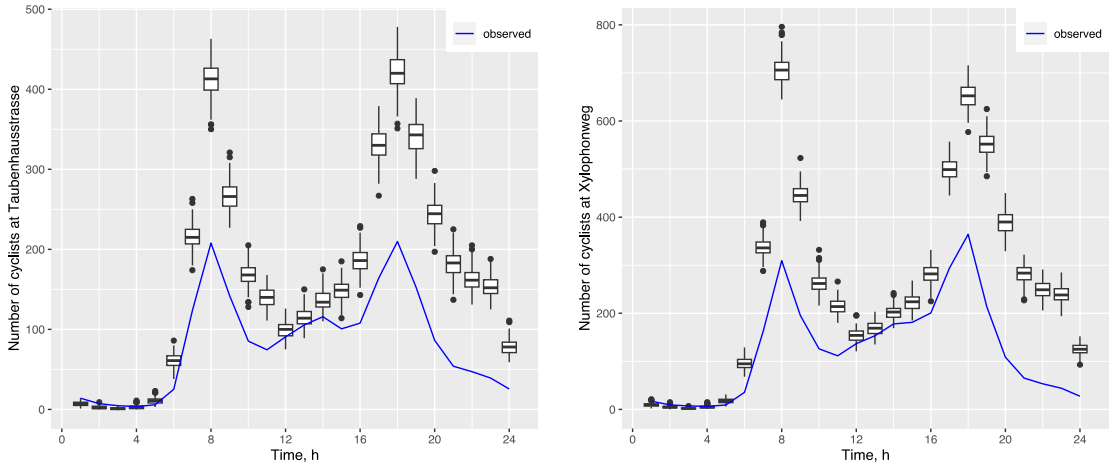

Abbildung 37: Boxplot Taubenhausstrasse und Xylophonweg mit gemessenen Velofahrer

Die Tabelle mit den Variationskoeffizienten wurde wieder auf zwei Stellen nach dem Komma gerundet. Es handelte sich jeweils um eigenständige Experimente mit separaten Simulationsdurchläufen. Dabei wurde jeweils ein niedriger Variationskoeffizienten erreicht, wobei die beiden Station *Baselstrasse* und *Bleicherstrasse* die höchsten Werte aufwies (Tabelle 12). Der Variationskoeffizient stabilisierte sich ab 100 Durchläufen für fast alle Stationen, ausser bei der *Baselstrasse* und *Bleicherstrasse*. Wobei der auf drei Stellen gerundete Wert zeigt, dass die zweistellige Rundung jeweils knapp aufgerundet wurde (Tabelle 13).

Tabelle 12: Variationskoeffizient für die Luzerner Zählstationen mit unterschiedlichen Simulationsdurchläufen

Station	5	10	15	25	50	100	200
Baselstrasse	0.06	0.11	0.09	0.11	0.10	0.11	0.10
Bleicherstrasse	0.06	0.05	0.06	0.07	0.06	0.06	0.07
Dammstrasse	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Freigleis Kleinmatt	0.02	0.04	0.02	0.03	0.03	0.03	0.03
Inseli	0.02	0.03	0.02	0.03	0.03	0.03	0.03
Langensandbrücke	0.01	0.02	0.01	0.02	0.02	0.02	0.02
Löwenplatz	0.02	0.02	0.02	0.03	0.02	0.02	0.02
Neustadtstrasse	0.03	0.06	0.03	0.03	0.03	0.03	0.03
Palace	0.01	0.03	0.02	0.02	0.02	0.02	0.02
Schweizerhofquai	0.01	0.02	0.01	0.01	0.01	0.01	0.01
Taubenhausstrasse	0.02	0.02	0.02	0.02	0.03	0.02	0.02
Xylophonweg	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Tabelle 13: Variationskoeffizient für Baselstrasse und mit unterschiedlichen Simulationsdurchläufen auf drei Stellen gerundet

Station	5	10	15	25	50	100	200
Baselstrasse	0.060	0.110	0.090	0.110	0.100	0.106	0.100
Bleicherstrasse	0.058	0.050	0.064	0.067	0.060	0.060	0.065

Ein Simulationsdurchlauf dauerte inklusive Initialisierung rund 4 Stunden, wobei die Initialisierung 10 Minuten benötigte. Es liefen jeweils 5 Simulationen gleichzeitig. Pro Simulation wurden rund 185'000 Personen berücksichtigt. Die Initialisierung benötigte damit doppelt so lange wie in Salzburg. Ein Grund dafür war die nötige Ergänzung der Erwerbsstatus Daten mit *prepareHomes*. Ein weiterer Grund war das komplexere Strassennetz aus den OSM Daten, welches aus fast 60'000 Objekten bestand.

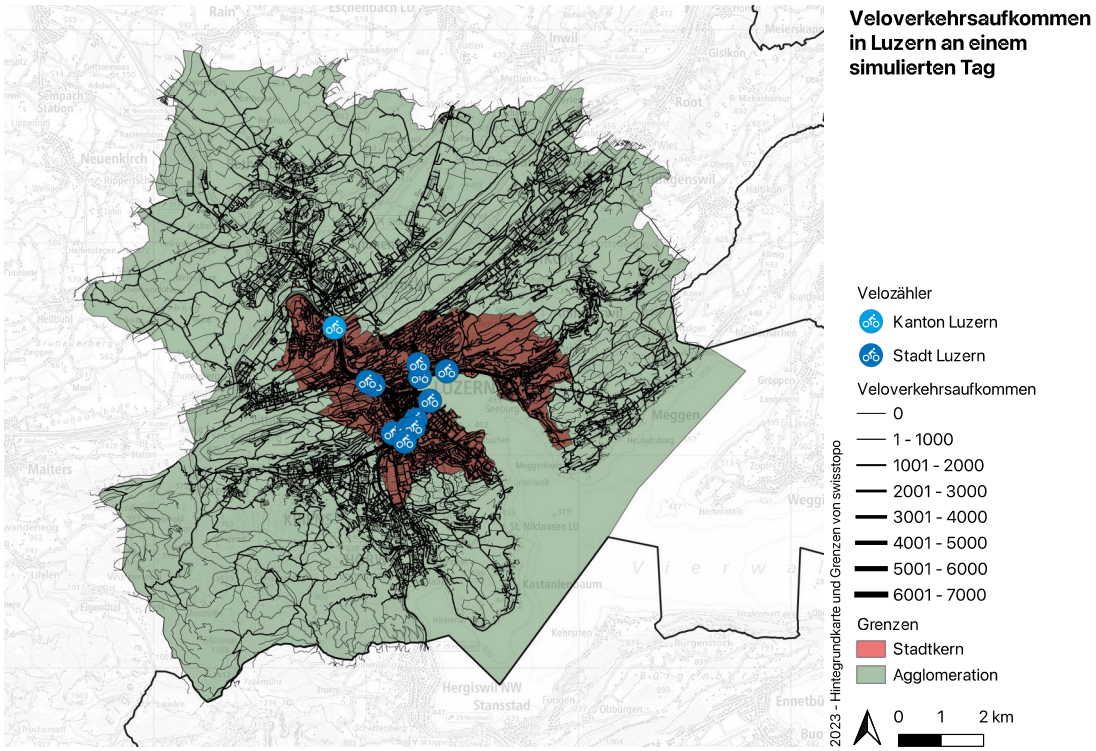

Abbildung 38: Veloverkehrsaufkommen in Luzern an einem simulierten Tag

Eine stichprobenartige Sichtung des simulierten täglichen Veloverkehrsaufkommens (Abbildung 38) scheint gut zu den eigenen Erfahrungen im luzerner Veloverkehr zu passen. Ein gutes Beispiel ist am Bahnhof Luzern ersichtlich (Abbildung 39). Hier nehmen, gemäss eigener Erfahrung, nur sehr wenige Velofahrer die Abzweigung Richtung Pilatusstrasse, sondern wählen die Bahnhofstrasse. Dies entspricht auch meiner favorisierten Route.

Abbildung 39: Simuliertes Verkehrsaufkommen am Bahnhof Luzern und an der Zürichstrasse

Allerdings sind auch einzelne Fehler zu finden. Als Beispiel, die Situation an der *Zürichstrasse:* Gemäss Simulation wählt eine Mehrheit der Velofahrer für einen kurzen Abschnitt die *Steinenstrasse*. Dazu müsste von Süden her kommend eine stark befahrenen Strasse zwei Mal überquert werden. Dies entspricht weder den eigenen Beobachtungen, noch meiner eigenen Veloroutenwahl für meinen Arbeitsweg.

4 Diskussion und Fazit

Im ersten Teil der Studie konnten die meisten Resultate aus der Studie von Kaziyeva et al. (2021a) gut wiederholt werden. Nur die Zahlen an der *Schanzlgasse* schienen abzuweichen. Hier schien es, dass die Daten der beobachten Durchfahrten, welche mit dem Modell (Kaziyeva et al., 2021b) mitgeliefert wurden, nicht mit den Daten aus der Studie übereinstimmten. Das Diagramm für die *Schanzlgasse* in der Studie (Kaziyeva et al., 2021a, p. 12) wies in den beobachteten Daten auch keine Morgen- und Abendspitzen auf. Gemäss Strassennetz-Daten aus dem Modell, liegt diese Station auf einer nationalen Veloroute. In OSM führte Ende 2023 keine bekannte Route über die Station. Jedoch befindet der *Mozart-Radweg* und der *Tauernradweg* rund 50 Meter entfernt. Es davon auszugehen, dass hier wohl viele Touristen entlang fahren, um in die Altstadt zu gelangen. Da die Simulation nur die Wohnbevölkerung enthält, kann diese auch keine Touristenströme abbilden.

Der Variationskoeffizient ist zwar bei allen Stationen generell niedrig, jedoch sind in den Boxplots mehrere Ausreisser ersichtlich. Deswegen ist es im Gegensatz zur Stu-

die empfehlenswert, mehr als einen Simulationsdurchlauf durchzuführen. Dies deckt zudem problematische Zählstationen auf, wie an der *Schanzlgasse* gut ersichtlich. Jedoch reichen hier etwa 10 Durchläufe. Eine zu hohe Anzahl Durchläufe erweckt nur eine Scheingenauigkeit der Resultate.

Die für Luzern ausgetauschten Daten führen zu einem guten Resultat und alle Stationen erfüllen die in der Einleitung gestellten Bedingungen. Auch weist dies darauf hin, dass sich die Luzerner bezüglich Velofahren ähnlich wie die Salzburger verhalten. Die kleine Erhöhung der gemessenen Velofahrten kurz nach dem Mittag, ist in den Diagrammen für Luzern ausgeprägter als in Salzburg, hier wäre zu prüfen, ob einen angepasste demografische Parametrisierung des Mobilitätsverhaltens für Luzern dies besser abbildet.

Die aus Datenschutzgründen reduzierte Genauigkeit für kleine Werte bei den «Workplaces» und «Homes» Daten scheinen keinen starken negativen Einfluss gehabt zu haben. Jedoch wäre es interessant, Daten auf die Person genau für die Simulation zu verwenden und damit zu testen, wie sich das Resultat verhält.

Die guten Resultate zeigen auch, dass der vorgestellte Ersatz der Erwerbsstatus Daten grundsätzlich funktioniert. Der Variationskoeffizient ist auch bei den Luzerner Daten generell niedrig und stabilisiert sich ähnlich wie in Salzburg.

Die OSM Daten eigneten sich grundsätzlich gut für das Strassennetzwerk und das hier vorgestellte Vorgehen lässt sich einfach auf andere Regionen in der Schweiz anwenden. Jedoch dauert die Simulation damit wesentlich länger und wäre für grössere Regionen als die Agglomeration Luzern herausfordernd. Hier wird vermutlich in naher Zukunft das Verkehrsnetz CH (“Bundesamt für Landestopografie,” 2023b) für die Schweiz Erleichterung bringen. Da sich das Verkehrsnetz CH am GIP orientiert, sollte es auch gut zum *Bicycle Model* passen. Es wäre interessant, mit dem Verkehrsnetz CH Daten die Agglomeration selbst als Analysegebiet zu wählen und die Agglomerationsgürtelgemeinden für die Systemgrenze.

Die beobachtete fehlerhafte Routenwahl an der *Zürichstrasse* lässt sich kaum generell verhindern. Daher ist es wichtig, dass jeweils jemand mit Ortskenntnis bei der Simulation beteiligt ist und falls nötig, die Parameter im Strassennetz von Hand optimiert.

Mit all den obengenannten Gründen konnte gezeigt werden, dass sich das Modell aus Salzburg gut auf Luzern übertragen liess.

5 Literaturverzeichnis

Amtliche Vermessung - Geoportal Kanton Luzern [WWW Document], 2023. URL https://daten.geo.lu.ch/produkt/amtverxx_col_v2#AVBBXXXX_DS (accessed 1.8.24).

Beitel, D., McNee, S., McLaughlin, F., Miranda-Moreno, L.F., 2018. Automated Validation and Interpolation of Long-Duration Bicycle Counting Data. Transp. Res. Rec. 2672, 75–86. https://doi.org/10.1177/0361198118783123

Bejarano, G., Astuvilca, J., Vega, P., 2015. Automation of process to load database from OSM for the design of public routes. Presented at the CEUR Workshop Proceedings, pp. 99–105.

Bos, R., Temme, R., 2014. A Roadmap towards Sustainable Mobility in Breda. Transp. Res. Procedia, Sustainable Mobility in Metropolitan Regions. mobil.TUM 2014. International Scientific Conference on Mobility and Transport. Conference Proceedings. 4, 103–115. https://doi.org/10.1016/j.trpro.2014.11.009

Bundesamt für Statistik, 2023a. Statistik der Unternehmensstruktur (STATENT), Beschäftigte und Arbeitsstätten: Geodaten 2021 [WWW Document]. Bundesamt Für Stat. URL https://www.bfs.admin.ch/asset/de/27245297 (accessed 11.9.23).

Bundesamt für Statistik, 2023b. Statistik der Bevölkerung und Haushalte (STATPOP), Geodaten 2022 [WWW Document]. Bundesamt Für Stat. URL https://www.bfs.admin.ch/asset/de/27965868 (accessed 12.2.23).

Bundesamt für Statistik, 2023c. Arbeitsmarktstatus nach Geschlecht, Nationalität, Altersgruppen, Familientyp - 1.4.1991-30.9.2023 | Tabelle [WWW Document]. Bundesamt Für Stat. URL https://www.bfs.admin.ch/asset/de/28245034 (accessed 12.2.23).

Bundesamt für Statistik, 2023d. Schulbesuchsquoten der 3-31-Jährigen - 1.8.1999-31.7.2022 | Tabelle [WWW Document]. Bundesamt Für Stat. URL https://www.bfs.admin.ch/asset/de/24130072 (accessed 12.2.23).

Dimter, S., Stober, D., Zagvozda, M., 2019. Strategic Planning of Cycling Infrastructure Towards Sustainable City Mobility - Case Study Osijek, Croatia. IOP Conf. Ser. Mater. Sci. Eng. 471, 092022. https://doi.org/10.1088/1757-899X/471/9/092022

Disconnected Islands — QGIS Python Plugins Repository [WWW Document], 2020. URL https://plugins.qgis.org/plugins/disconnected-islands/ (accessed 1.2.24).

Felício, S., Hora, J., Ferreira, M.C., Abrantes, D., Costa, P.D., Dangelo, C., Silva, J., Galvão, T., 2022. Handling OpenStreetMap georeferenced data for route planning. Transp. Res. Procedia, 24th Euro Working Group on Transportation Meeting 62, 189–196. https://doi.org/10.1016/j.trpro.2022.02.024

Garber, M.D., Watkins, K.E., Kramer, M.R., 2019. Comparing bicyclists who use smartphone apps to record rides with those who do not: Implications for representativeness and selection bias. J. Transp. Health 15, 100661. https://doi.org/10.1016/j.jth.2019.100661

Gemeindetypologie - LUSTAT Statistik Luzern [WWW Document], 2023. . Gemeindetypologie - LUSTAT Stat. Luzern. URL https://www.lustat.ch/services/lexikon/raumgliederungen/gemeindetypologie (accessed 3.1.23).

Graphenintegrations-Plattform GIP: Der multimodale, digitale Verkehrsgraph für ganz Österreich [WWW Document], 2023. URL https://www.gip.gv.at/ (accessed 12.29.23).

Graser, A., Straub, M., Dragaschnig, M., 2013. Ein systematischer Vergleich der Straßennetzwerke von GIP und OpenStreetMap im Großraum Wien, in: Strobl, J., Blaschke, T., Griesebner, G. (Eds.), Angewandte Geoinformatik 2013 - Beiträge zum 25. AGIT Symposium. Wichmann Verlag, Deutschland, pp. 424–433.

Hendricks, W.A., Robey, K.W., 1936. The Sampling Distribution of the Coefficient of Variation. Ann. Math. Stat. 7, 129–132. https://doi.org/10.1214/aoms/1177732503

Home - osm2pgsql [WWW Document], 2023. URL https://osm2pgsql.org/ (accessed 1.2.24).

Jestico, B., Nelson, T., Winters, M., 2016. Mapping ridership using crowdsourced cycling data. J. Transp. Geogr. 52, 90–97. https://doi.org/10.1016/j.jtrangeo.2016.03.006

Kaziyeva, D., Loidl, M., Wallentin, G., 2021a. Simulating Spatio-Temporal Patterns of Bicycle Flows with an Agent-Based Model. Isprs Int. J. Geo-Inf. 10, 88. https://doi.org/10.3390/ijgi10020088

Kaziyeva, D., Wallentin, G., Loidl, M., 2021b. Bicycle model v2.0.0 [WWW Document]. CoMSES Comput. Model Libr. URL https://www.comses.net/codebases/768182a3-eb7c-4d93-9c27-c42d878bb2ea/releases/2.0.0/ (accessed 3.20.22).

Kaziyeva, D., Wallentin, G., Loidl, M., Mohr, S., Neuwirth, C., 2018. Reviewing software for agent-based bicycle flow models. GI_Forum 6, 291–296. https://doi.org/10.1553/GISCIENCE2018_01_S291

Leao, S.Z., Pettit, C., 2017. Mapping bicycling patterns with an agent-based model, census and crowdsourced data. Lect. Notes Comput. Sci. Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinforma. 10051 LNAI, 112–128. https://doi.org/10.1007/978-3-319-51957-9_7

Littau [WWW Document], 2023. . Wikipedia. URL https://de.wikipedia.org/w/index.php?title=Littau&oldid=239004172 (accessed 12.28.23).

Loidl, M., Zagel, B., 2014. Assessing Bicycle Safety in Multiple Networks with Different Data Models, ISBN. Verlag der Österreichischen Akademie der Wissenschaften. https://doi.org/10.1553/giscience2014s144

Lorscheid, I., Heine, B.-O., Meyer, M., 2012. Opening the 'black box' of simulations: increased transparency and effective communication through the systematic design of experiments. Comput. Math. Organ. Theory 18, 22–62. https://doi.org/10.1007/s10588-011-9097-3

Munira, S., Sener, I.N., 2020. A geographically weighted regression model to examine the spatial variation of the socioeconomic and land-use factors associated with Strava bike activity in Austin, Texas. J. Transp. Geogr. 88, 102865. https://doi.org/10.1016/j.jtrangeo.2020.102865

Neis, P., 2014. Von Qualitätsuntersuchungen zu Nutzungspotentialen gemeinsam zusammengetragener Geodaten. KN - J. Cartogr. Geogr. Inf. 64, 130–136. https://doi.org/10.1007/BF03544142

NetworkX — NetworkX documentation [WWW Document], 2023. URL https://networkx.org/ (accessed 1.2.24).

Neumeier, L.M., Loidl, M., Reich, B., Fernandez La Puente de Battre, M.D., Kissel, C.K., Templin, C., Schmied, C., Niebauer, J., Niederseer, D., 2020. Effects of active commuting on health-related quality of life and sickness-related absence. Scand. J. Med. Sci. Sports 30, 31–40. https://doi.org/10.1111/sms.13667

OpenStreetMap. OSM Dump — Switzerland [WWW Document], 2023. URL https://download.geofabrik.de/europe/switzerland.html (accessed 11.9.23).

osm2pgrouting - Import OSM data into pgRouting Database — Open Source Routing Library [WWW Document], 2023. URL https://pgrouting.org/docs/tools/osm2pgrouting.html (accessed 12.30.23).

osm_relations table does not get populated · Issue #301 · pgRouting/osm2pgrouting [WWW Document], 2023. . GitHub. URL https://github.com/pgRouting/osm2pgrouting/issues/301 (accessed 1.2.24).

Overpass turbo [WWW Document], 2023. URL https://overpass-turbo.eu/ (accessed 12.30.23).

Ramm, F., n.d. OpenStreetMap Data in Layered GIS-Format [WWW Document]. URL https://download.geofabrik.de/osm-data-in-gis-formats-free.pdf

spatialthoughts, 2020. Answer to “Find intersections of roads in QGIS.” Geogr. Inf. Syst. Stack Exch.

swissTNE Base [WWW Document], 2023a. . Bundesamt Für Landestopografie Swisstopo. URL https://www.swisstopo.admin.ch/de/geodata/landscape/tne.html (accessed 12.29.23).

Verkehrsnetz CH [WWW Document], 2023b. . Bundesamt Für Landestopografie Swisstopo. URL https://www.swisstopo.admin.ch/de/swisstopo/verkehrsnetz-schweiz.html (accessed 12.29.23).

whuber, 2012. Answer to “Calculating average width of polygon?” Geogr. Inf. Syst. Stack Exch.

Ziemke, D., Metzler, S., Nagel, K., 2017. Modeling bicycle traffic in an agent-based transport simulation. Procedia Comput. Sci., 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal 109, 923–928. https://doi.org/10.1016/j.procs.2017.05.424

Anhänge

Anhang A: Processing Script für Facilities

https://firesoft.ch/unigis/velo/facilities_processingscript.py

```
# -*- coding: utf-8 -*-

"""
***************************************************************************
*                                                                         *
*   This program is free software; you can redistribute it and/or modify  *
*   it under the terms of the GNU General Public License as published by  *
*   the Free Software Foundation; either version 2 of the License, or     *
*   (at your option) any later version.                                   *
*                                                                         *
***************************************************************************
"""
from PyQt5.QtCore import QVariant
from qgis.PyQt.QtCore import QCoreApplication
from qgis.core import (QgsFeature,
                       QgsField,
                       QgsFields,
                       QgsProcessing,
                       QgsFeatureSink,
                       QgsProcessingException,
                       QgsProcessingAlgorithm,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterFeatureSink,
                       QgsProcessingParameterField,
                       QgsProcessingParameterNumber,
                       QgsProcessingParameterString,
                       QgsProcessingParameterVectorDestination,
                       QgsCoordinateReferenceSystem,
                       QgsProject)
from qgis import processing
import re

class ThesisFacilitiesProcessingAlgorithm(QgsProcessingAlgorithm):
    # Constants used to refer to parameters and outputs. They will be
    # used when calling the algorithm from another algorithm, or when
    # calling from the QGIS console.

    INPUT = 'INPUT'
    INPUT_BUFFER = 'outline_buffer'
    INPUT_POI_PT = 'POI_PT'
    INPUT_POI_PY = 'POI_PY'
    FILTER_AUTHORITY = 'filter_authority'
    FILTER_DOCTOR = 'filter_doctor'
    FILTER_KINDERGARTEN = 'filter_kindergarten'
    FILTER_RECREATION = 'filter_recreation'
    FILTER_SCHOOL = 'filter_school'
    FILTER_SHOP = 'filter_shop'
    FILTER_UNIVERSITY = 'filter_university'

    OUTPUT1 = 'OUTPUT1'
    OUTPUT2 = 'OUTPUT2'

    TARGET_CRS = '2056' # 2056: LV95, 32633: WGS 84 / UTM zone 33N
    CRS_OPERATION = {
        '2056': '+proj=pipeline +step +proj=unitconvert +xy_in=deg +xy_out=rad '+
            '+step +proj=push +v_3 +step +proj=cart +ellps=WGS84 +step '+
            '+proj=helmert +x=-674.374 +y=-15.056 +z=-405.346 +step +inv '+
            '+proj=cart +ellps=bessel +step +proj=pop +v_3 +step +proj=somerc '+
            '+lat_0=46.9524055555556 +lon_0=7.43958333333333 +k_0=1 +x_0=2600000 '+
            '+y_0=1200000 +ellps=bessel',
        '32633': '+proj=pipeline +step +proj=unitconvert +xy_in=deg +xy_out=rad '+
            '+step +proj=utm +zone=33 +ellps=WGS84'
    }
    def tr(self, string):
        return QCoreApplication.translate('Processing', string)

    def createInstance(self):
        return ThesisFacilitiesProcessingAlgorithm()

    def name(self):
        return 'facilities_processing'

    def displayName(self):
        return self.tr('Prepare Facilities')

    def group(self):
        return self.tr('Master Thesis')

    def groupId(self):
        return 'thesis'

    def shortHelpString(self):
        return self.tr("Prepares the geofabrik data for use in Gama simulation.")

    def initAlgorithm(self, config=None):
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT,
                self.tr('Outline layer'),
                [QgsProcessing.TypeVectorPolygon]
```

```
            )
        )
        self.addParameter(
            QgsProcessingParameterNumber(
                self.INPUT_BUFFER,
                self.tr('Outline Buffer (in metre)'),
                defaultValue=50,
                type=QgsProcessingParameterNumber.Integer
            )
        )
        self.addParameter(
            QgsProcessingParameterString(
                self.FILTER_AUTHORITY,
                'Authority Filter',
                defaultValue='town_hall'
            )
        )
        self.addParameter(
            QgsProcessingParameterString(
                self.FILTER_DOCTOR,
                'Doctor Filter',
                defaultValue='doctors, hospital, clinic'
            )
        )
        self.addParameter(
            QgsProcessingParameterString(
                self.FILTER_KINDERGARTEN,
                'Kindergarten Filter',
                defaultValue='kindergarten'
            )
        )
        self.addParameter(
            QgsProcessingParameterString(
                self.FILTER_RECREATION,
                'Recreation Filter',
                defaultValue='arts_centre,artwork,attraction,bar,' +
                    'biergarten,cafe,castle,cinema,community_centre,' +
                    'dog_park,fast_food,guesthouse,hostel,hotel,' +
                    'ice_rink,library,nightclub,park,playground,' +
                    'pub,restaurant,sports_centre,stadium,theatre,' +
                    'theme_park,tourist_info,viewpoint,zoo'
            )
        )
        self.addParameter(
            QgsProcessingParameterString(
                self.FILTER_SCHOOL,
                'School Filter',
                defaultValue='school'
            )
        )
        self.addParameter(
            QgsProcessingParameterString(
                self.FILTER_SHOP,
                'Shop Filter',
                defaultValue='bakery,beauty_shop,beverages,bicycle_shop,' +
                    'bookshop,butcher,car_dealership,chemist,clothes,' +
                    'computer_shop,convenience,department_store,' +
                    'doityourself,florist,furniture_shop,garden_centre,' +
                    'general,gift_shop,greengrocer,hairdresser,' +
                    'jeweller,kiosk,laundry,mall,mobile_phone_shop,' +
                    'newsagent,optician,outdoor_shop,pharmacy,' +
                    'shoe_shop,sports_shop,stationery,supermarket,' +
                    'toy_shop,travel_agent,video_shop'
            )
        )
        self.addParameter(
            QgsProcessingParameterString(
                self.FILTER_UNIVERSITY,
                'University Filter',
                defaultValue='university'
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT_POI_PT,
                self.tr('OSM POIs Point'),
                types=[QgsProcessing.TypeVectorPoint]
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT_POI_PY,
                self.tr('OSM POIs Polygon'),
                types=[QgsProcessing.TypeVectorPolygon]
            )
        )
        self.addParameter(
            QgsProcessingParameterVectorDestination(
                self.OUTPUT1,
                self.tr('All POIs')
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSink(
                self.OUTPUT2,
                self.tr('Facilities')
            )
        )

    def processAlgorithm(self, parameters, context, feedback):
        source = self.parameterAsSource(
            parameters,
            self.INPUT,
```

```
    context
)
if source is None:
    raise QgsProcessingException(
        self.invalidSourceError(parameters, self.INPUT))

# Output source CRS
feedback.pushInfo('CRS is {}'.format(source.sourceCrs().authid()))
self.TARGET_CRS = source.sourceCrs().authid()[5:]
if self.TARGET_CRS not in self.CRS_OPERATION:
    feedback.pushInfo('Source CRS is not supported. Only following are '+
        'supported: EPSG:'+', EPSG:'.join(list(self.CRS_OPERATION.keys())))
    raise QgsProcessingException(
        self.invalidSourceError(parameters, self.INPUT))

# Create buffer from the outline layer
params = {
    'INPUT': parameters[self.INPUT],
    'DISTANCE': parameters[self.INPUT_BUFFER],
    'SEGMENTS': 5,
    'END_CAP_STYLE': 0,
    'JOIN_STYLE': 0,
    'MITER_LIMIT': 2,
    'DISSOLVE': False,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
outlineBuffer = processing.run(
    'native:buffer',
    params, context=context, feedback=feedback)['OUTPUT']

# Clip OSM point layer
feedback.pushInfo('Prepare OSM point layer')
params = {
    'INPUT': parameters[self.INPUT_POI_PT],
    'OVERLAY': outlineBuffer,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
poiPtOutput = processing.run(
    'native:clip',
    params, context=context, feedback=feedback)['OUTPUT']

if feedback.isCanceled():
    return {}

# Reproject OSM point layer
params = {
    'INPUT': poiPtOutput,
    'TARGET_CRS':QgsCoordinateReferenceSystem('EPSG:'+self.TARGET_CRS),
    'OPERATION': self.CRS_OPERATION[self.TARGET_CRS],
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
poiPtOutput = processing.run(
    'native:reprojectlayer',
    params, context=context, feedback=feedback)['OUTPUT']
# rename layer to get predictable name
poiPtOutput.setName('pois_p')

if feedback.isCanceled():
    return {}

# Clip OSM polygon layer
feedback.pushInfo('Prepare OSM polygon layer')
params = {
    'INPUT': parameters[self.INPUT_POI_PY],
    'OVERLAY': outlineBuffer,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
poiPyOutput = processing.run(
    'native:clip',
    params, context=context, feedback=feedback)['OUTPUT']

if feedback.isCanceled():
    return {}

# Reproject OSM polygon layer
params = {
    'INPUT': poiPyOutput,
    'TARGET_CRS':QgsCoordinateReferenceSystem('EPSG:'+self.TARGET_CRS),
    'OPERATION': self.CRS_OPERATION[self.TARGET_CRS],
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
poiPyOutput = processing.run(
    'native:reprojectlayer',
    params, context=context, feedback=feedback)['OUTPUT']

if feedback.isCanceled():
    return {}

# Dissolve OSM polygon layer
params = {
    'INPUT': poiPyOutput,
    'FIELD': ['osm_id'],
    'SEPARATE_DISJOINT': False,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
poiPyOutput = processing.run(
    'native:dissolve',
    params, context=context, feedback=feedback)['OUTPUT']

if feedback.isCanceled():
    return {}

# centroids of OSM polygon layer
```

```
params = {
    'INPUT': poiPyOutput,
    'ALL_PARTS': False,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
poiCentOutput = processing.run(
    'native:centroids',
    params, context=context, feedback=feedback)['OUTPUT']
poiCentOutput.setName('pois_a')

if feedback.isCanceled():
    return {}

# Merge OSM layers
feedback.pushInfo('Merge OSM layers')
params = {
    'LAYERS': [poiPtOutput, poiCentOutput],
    'CRS':  QgsCoordinateReferenceSystem('EPSG:2056'),
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
allPoisOutput = processing.run(
    'native:mergevectorlayers',
    params, context=context, feedback=feedback)['OUTPUT']

if feedback.isCanceled():
    return {}

# Delete the path field
params = {
    'INPUT': allPoisOutput,
    'COLUMN': ['path'],
    'OUTPUT': self.parameterAsOutputLayer(parameters, self.OUTPUT1, context)
}
poiOutput = processing.run('native:deletecolumn',
    params, context=context, feedback=feedback)['OUTPUT']

# Create fields
facilityFields = QgsFields();
facilityFields.append(QgsField('id', QVariant.Int))
facilityFields.append(QgsField('type', QVariant.String, '', 16))
facilityFields.append(QgsField('osm_id', QVariant.String, '', 12))
facilityFields.append(QgsField('fclass', QVariant.String, '', 40))
facilityFields.append(QgsField('name', QVariant.String, '', 100))
(facilities, facilitiesId) = self.parameterAsSink(
    parameters,
    self.OUTPUT2,
    context,
    facilityFields,
    allPoisOutput.wkbType(),
    allPoisOutput.sourceCrs()
)
# prepare filters
types = {
    'authority': [],
    'doctor': [],
    'kindergarten': [],
    'recreation': [],
    'school': [],
    'shop': [],
    'university': []
}

# Loop through all types and get the associated input parameter
for key in types:
    # Split input string
    f = re.split('\\W', parameters[getattr(self, 'FILTER_'+key.upper())])
    # Filter empty strings
    types[key] = list(filter(None, f))

if feedback.isCanceled():
    return {}

# Compute the number of steps to display within the progress bar and
# get features from source
total = 100.0 / allPoisOutput.featureCount() if allPoisOutput.featureCount() else 0
features = allPoisOutput.getFeatures()
currentId = 0

for current, feature in enumerate(features):
    # Stop the algorithm if cancel button has been clicked
    if feedback.isCanceled():
        return {}

    type = ''
    for cType in types:
        if feature['fclass'] in types[cType]:
            type = cType
            break

    if type != '':
        currentId += 1
        f = QgsFeature()
        f.setFields(facilityFields)
        f['id'] = currentId
        f['type'] = type
        f['osm_id'] = feature['osm_id']
        f['fclass'] = feature['fclass']
        f['name'] = feature['name']
        # add geometry
        f.setGeometry(feature.geometry())

        # Add a feature to facilities
        facilities.addFeature(f, QgsFeatureSink.FastInsert)
```

```
            # Update the progress bar
            feedback.setProgress(int(current * total))

        return {
           self.OUTPUT1: poiOutput,
           self.OUTPUT2: facilitiesId
        }
```

Anhang B: Processing Script für Workplaces

https://firesoft.ch/unigis/velo/workplaces_processingscript.py

```
# -*- coding: utf-8 -*-

"""
***************************************************************************
*                                                                         *
*   This program is free software; you can redistribute it and/or modify  *
*   it under the terms of the GNU General Public License as published by  *
*   the Free Software Foundation; either version 2 of the License, or     *
*   (at your option) any later version.                                   *
*                                                                         *
***************************************************************************
"""
from PyQt5.QtCore import QVariant
from qgis.PyQt.QtCore import QCoreApplication
from qgis.core import (QgsFeature,
                       QgsField,
                       QgsFields,
                       QgsProcessing,
                       QgsFeatureSink,
                       QgsProcessingException,
                       QgsProcessingAlgorithm,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterFeatureSink,
                       QgsProcessingParameterFile,
                       QgsProcessingParameterNumber,
                       QgsProcessingParameterVectorDestination,
                       QgsProject,
                       QgsRectangle)
from qgis import processing
import math

class ThesisWorkplacesProcessingAlgorithm(QgsProcessingAlgorithm):
    # Constants used to refer to parameters and outputs. They will be
    # used when calling the algorithm from another algorithm, or when
    # calling from the QGIS console.

    INPUT = 'INPUT'
    INPUT_BUFFER = 'outline_buffer'
    INPUT_CSV = 'csv'

    OUTPUT = 'OUTPUT'

    FIELD_MAP = {
        'employees': 'B08EMPT' # Beschäftigte Total
    }

    def tr(self, string):
        return QCoreApplication.translate('Processing', string)

    def createInstance(self):
        return ThesisWorkplacesProcessingAlgorithm()

    def name(self):
        return 'workplaces_processing'

    def displayName(self):
        return self.tr('Prepare Workplaces')

    def group(self):
        return self.tr('Master Thesis')

    def groupId(self):
        return 'thesis'

    def shortHelpString(self):
        return self.tr("Prepares the BFS data for use in Gama simulation.")

    def initAlgorithm(self, config=None):
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT,
                self.tr('Outline layer'),
                [QgsProcessing.TypeVectorPolygon]
            )
        )
        self.addParameter(
            QgsProcessingParameterNumber(
                self.INPUT_BUFFER,
                self.tr('Outline Buffer (in metre)'),
                defaultValue=50,
                type=QgsProcessingParameterNumber.Integer
            )
        )
        self.addParameter(
            QgsProcessingParameterFile(
                self.INPUT_CSV,
                self.tr('BFS STATENT file (csv)'),
```

```
            extension='csv'
        )
    )
    self.addParameter(
        QgsProcessingParameterFeatureSink(
            self.OUTPUT,
            self.tr('Workplaces')
        )
    )

def processAlgorithm(self, parameters, context, feedback):
    source = self.parameterAsSource(
        parameters,
        self.INPUT,
        context
    )

    if source is None:
        raise QgsProcessingException(
            self.invalidSourceError(parameters, self.INPUT))

    # Output source CRS
    feedback.pushInfo('CRS is {}'.format(source.sourceCrs().authid()))

    feedback.pushInfo('Import CSV')
    # Create points from CSV
    params = {
        'INPUT': parameters[self.INPUT_CSV],
        'XFIELD': 'E_KOORD',
        'YFIELD': 'N_KOORD',
        'ZFIELD':'',
        'MFIELD':'',
        'TARGET_CRS': source.sourceCrs(),
        'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
    }
    allPointsOutput = processing.run(
        'native:createpointslayerfromtable',
        params, context=context, feedback=feedback)['OUTPUT']

    if feedback.isCanceled():
        return {}

    # Create buffer from the outline layer
    feedback.pushInfo('Create buffer from the outline layer')
    params = {
        'INPUT': parameters[self.INPUT],
        'DISTANCE': parameters[self.INPUT_BUFFER],
        'SEGMENTS': 5,
        'END_CAP_STYLE': 0,
        'JOIN_STYLE': 0,
        'MITER_LIMIT': 2,
        'DISSOLVE': False,
        'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
    }
    outlineBuffer = processing.run(
        'native:buffer',
        params, context=context, feedback=feedback)['OUTPUT']

    if feedback.isCanceled():
        return {}

    # Clip point layer
    feedback.pushInfo('Remove points outside the outline buffer')
    params = {
        'INPUT': allPointsOutput,
        'OVERLAY': outlineBuffer,
        'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
    }
    pointsOutput = processing.run(
        'native:clip',
        params, context=context, feedback=feedback)['OUTPUT']

    if feedback.isCanceled():
        return {}

    oExtent = outlineBuffer.sourceExtent()
    # Round grid extent to 100 meters (ceil and floor)
    gridExtent = '{},{},{},{}'.format(
        math.floor(oExtent.xMinimum()/100)*100,
        math.ceil(oExtent.xMaximum()/100)*100,
        math.floor(oExtent.yMinimum()/100)*100,
        math.ceil(oExtent.yMaximum()/100)*100)
    # create grid
    params = {
        'TYPE': 2,
        'EXTENT': gridExtent+' [' + source.sourceCrs().authid() +']',
        'HSPACING': 100,
        'VSPACING': 100,
        'HOVERLAY': 0,
        'VOVERLAY': 0,
        'CRS': source.sourceCrs(),
        'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
    }
    gridOutput = processing.run(
        'native:creategrid',
        params, context=context, feedback=feedback)['OUTPUT']

    if feedback.isCanceled():
        return {}

    # Extract by location
    # remove cells outside outline
    params = {
```

```
            'INPUT': gridOutput,
            'PREDICATE': [0,4],
            'INTERSECT': parameters[self.INPUT],
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        gridOutput = processing.run(
            'native:extractbylocation',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        # Create fields
        workplaceFields = QgsFields();
        workplaceFields.append(QgsField('id', QVariant.Int))
        workplaceFields.append(QgsField('employees', QVariant.Int))
        (workplaces, workplacesId) = self.parameterAsSink(
            parameters,
            self.OUTPUT,
            context,
            workplaceFields,
            source.wkbType(),
            source.sourceCrs()
        )

        # Compute the number of steps to display within the progress bar and
        # get features from source
        total = 100.0 / gridOutput.featureCount() if gridOutput.featureCount() else 0
        features = gridOutput.getFeatures()
        currentId = 0

        for current, feature in enumerate(features):
            # Stop the algorithm if cancel button has been clicked
            if feedback.isCanceled():
                return {}
            # use bottom left corner to select point
            search = QgsRectangle(feature['left']-1, feature['bottom']-1,
                feature['left']+1, feature['bottom']+1)

            pointsOutput.selectByRect(search)
            selection = pointsOutput.selectedFeatures()
            if len(selection) == 1:
                currentId += 1
                f = QgsFeature()
                f.setFields(workplaceFields)
                f['id'] = currentId
                f['employees'] = selection[0][self.FIELD_MAP['employees']]
                # add grid geometry
                f.setGeometry(feature.geometry())

                # Add a feature to workplaces
                workplaces.addFeature(f, QgsFeatureSink.FastInsert)

            # Update the progress bar
            feedback.setProgress(int(current * total))

        return {
           self.OUTPUT: workplacesId
        }
```

Anhang C: Processing Script für Homes

https://firesoft.ch/unigis/velo/homes_processingscript.py

```
# -*- coding: utf-8 -*-

"""
***************************************************************************
*                                                                         *
*   This program is free software; you can redistribute it and/or modify  *
*   it under the terms of the GNU General Public License as published by  *
*   the Free Software Foundation; either version 2 of the License, or     *
*   (at your option) any later version.                                   *
*                                                                         *
***************************************************************************
"""
from PyQt5.QtCore import QVariant
from qgis.PyQt.QtCore import QCoreApplication
from qgis.core import (QgsFeature,
                       QgsField,
                       QgsFields,
                       QgsProcessing,
                       QgsFeatureSink,
                       QgsProcessingException,
                       QgsProcessingAlgorithm,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterFeatureSink,
                       QgsProcessingParameterFile,
                       QgsProcessingParameterNumber,
                       QgsProcessingParameterVectorDestination,
                       QgsProject,
                       QgsRectangle)
from qgis import processing
import math

class ThesisHomesProcessingAlgorithm(QgsProcessingAlgorithm):
    # Constants used to refer to parameters and outputs. They will be
    # used when calling the algorithm from another algorithm, or when
    # calling from the QGIS console.
```

```
INPUT = 'INPUT'
INPUT_BUFFER = 'outline_buffer'
INPUT_CSV = 'csv'
INPUT_YEAR = 'year'

OUTPUT = 'OUTPUT'

# jj = Jahr der Daten, auf 2 Ziffern kodiert (z.B. 2013 = 13)
FIELD_MAP = {
    'residents': 'BjjBTOT', # Ständige Wohnbevölkerung, Total
    # Ständige männliche Wohnbevölkerung, Total
    'm_tot': 'BjjBMTOT',
    'm_below_5':'BjjBM01', # 0 - 4 jährig
    'm_5_9':    'BjjBM02', # 5 - 9 jährig
    'm_10_14':  'BjjBM03', # 10 - 14 jährig
    'm_15_19':  'BjjBM04', # 15 - 19 jährig
    'm_20_24':  'BjjBM05', # 20 - 24 jährig
    'm_25_29':  'BjjBM06', # 25 - 29 jährig
    'm_30_34':  'BjjBM07', # 30 - 34 jährig
    'm_35_39':  'BjjBM08', # 35 - 39 jährig
    'm_40_44':  'BjjBM09', # 40 - 44 jährig
    'm_45_49':  'BjjBM10', # 45 - 49 jährig
    'm_50_54':  'BjjBM11', # 50 - 54 jährig
    'm_55_59':  'BjjBM12', # 55 - 59 jährig
    'm_60_64':  'BjjBM13', # 60 - 64 jährig
    'm_65_69':  'BjjBM14', # 65 - 69 jährig
    'm_70_74':  'BjjBM15', # 70 - 74 jährig
    'm_75_79':  'BjjBM16', # 75 - 79 jährig
    'm_80_84':  'BjjBM17', # 80 - 84 jährig
    'm_85_89':  'BjjBM18', # 85 - 89 jährig
    'm_over_90':'BjjBM19', # 90 jährig und älter
    # Ständige weibliche Wohnbevölkerung, Total
    'f_tot': 'BjjBWTOT',
    'f_below_5':'BjjBW01', # 0 - 4 jährig
    'f_5_9':    'BjjBW02', # 5 - 9 jährig
    'f_10_14':  'BjjBW03', # 10 - 14 jährig
    'f_15_19':  'BjjBW04', # 15 - 19 jährig
    'f_20_24':  'BjjBW05', # 20 - 24 jährig
    'f_25_29':  'BjjBW06', # 25 - 29 jährig
    'f_30_34':  'BjjBW07', # 30 - 34 jährig
    'f_35_39':  'BjjBW08', # 35 - 39 jährig
    'f_40_44':  'BjjBW09', # 40 - 44 jährig
    'f_45_49':  'BjjBW10', # 45 - 49 jährig
    'f_50_54':  'BjjBW11', # 50 - 54 jährig
    'f_55_59':  'BjjBW12', # 55 - 59 jährig
    'f_60_64':  'BjjBW13', # 60 - 64 jährig
    'f_65_69':  'BjjBW14', # 65 - 69 jährig
    'f_70_74':  'BjjBW15', # 70 - 74 jährig
    'f_75_79':  'BjjBW16', # 75 - 79 jährig
    'f_80_84':  'BjjBW17', # 80 - 84 jährig
    'f_85_89':  'BjjBW18', # 85 - 89 jährig
    'f_over_90':'BjjBW19', # 90 jährig und älter
    'm_below_15': '',
    'f_below_15': ''
}

def tr(self, string):
    return QCoreApplication.translate('Processing', string)

def createInstance(self):
    return ThesisHomesProcessingAlgorithm()

def name(self):
    return 'homes_processing'

def displayName(self):
    return self.tr('Prepare Homes')

def group(self):
    return self.tr('Master Thesis')

def groupId(self):
    return 'thesis'

def shortHelpString(self):
    return self.tr("Prepares the BFS data for use in Gama simulation.")

def initAlgorithm(self, config=None):
    self.addParameter(
        QgsProcessingParameterFeatureSource(
            self.INPUT,
            self.tr('Outline layer'),
            [QgsProcessing.TypeVectorPolygon]
        )
    )
    self.addParameter(
        QgsProcessingParameterNumber(
            self.INPUT_BUFFER,
            self.tr('Outline Buffer (in metre)'),
            defaultValue=50,
            type=QgsProcessingParameterNumber.Integer
        )
    )
    self.addParameter(
        QgsProcessingParameterFile(
            self.INPUT_CSV,
            self.tr('BFS STATPOP file (csv)'),
            extension='csv'
        )
    )
    self.addParameter(
        QgsProcessingParameterNumber(
            self.INPUT_YEAR,
```

```
                self.tr('Year'),
                defaultValue=2022,
                type=QgsProcessingParameterNumber.Integer
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSink(
                self.OUTPUT,
                self.tr('Homes')
            )
        )

    def processAlgorithm(self, parameters, context, feedback):
        source = self.parameterAsSource(
            parameters,
            self.INPUT,
            context
        )
        if source is None:
            raise QgsProcessingException(
                self.invalidSourceError(parameters, self.INPUT))

        # Output source CRS
        feedback.pushInfo('CRS is {}'.format(source.sourceCrs().authid()))

        # apply year to field map
        year = parameters[self.INPUT_YEAR]
        if year > 99:
            year = year - 2000
        for key in self.FIELD_MAP:
            self.FIELD_MAP[key] = self.FIELD_MAP[key].replace('jj', str(year))

        feedback.pushInfo('Import CSV')
        # Create points from CSV
        params = {
            'INPUT': parameters[self.INPUT_CSV],
            'XFIELD': 'E_KOORD',
            'YFIELD': 'N_KOORD',
            'ZFIELD':'',
            'MFIELD':'',
            'TARGET_CRS': source.sourceCrs(),
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        allPointsOutput = processing.run(
            'native:createpointslayerfromtable',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        # Create buffer from the outline layer
        feedback.pushInfo('Create buffer from the outline layer')
        params = {
            'INPUT': parameters[self.INPUT],
            'DISTANCE': parameters[self.INPUT_BUFFER],
            'SEGMENTS': 5,
            'END_CAP_STYLE': 0,
            'JOIN_STYLE': 0,
            'MITER_LIMIT': 2,
            'DISSOLVE': False,
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        outlineBuffer = processing.run(
            'native:buffer',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        # Clip point layer
        feedback.pushInfo('Remove points outside the outline buffer')
        params = {
            'INPUT': allPointsOutput,
            'OVERLAY': outlineBuffer,
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        pointsOutput = processing.run(
            'native:clip',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        oExtent = outlineBuffer.sourceExtent()
        # Round grid extent to 100 meters (ceil and floor)
        gridExtent = '{},{},{},{}'.format(
            math.floor(oExtent.xMinimum()/100)*100,
            math.ceil(oExtent.xMaximum()/100)*100,
            math.floor(oExtent.yMinimum()/100)*100,
            math.ceil(oExtent.yMaximum()/100)*100)
        # create grid
        params = {
            'TYPE': 2,
            'EXTENT': gridExtent+' [' + source.sourceCrs().authid() +']',
            'HSPACING': 100,
            'VSPACING': 100,
            'HOVERLAY': 0,
            'VOVERLAY': 0,
            'CRS': source.sourceCrs(),
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        gridOutput = processing.run(
            'native:creategrid',
```

```
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        # Extract by location
        # remove cells outside outline
        params = {
            'INPUT': gridOutput,
            'PREDICATE': [0,4],
            'INTERSECT': parameters[self.INPUT],
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        gridOutput = processing.run(
            'native:extractbylocation',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        # Create fields
        workplaceFields = QgsFields();
        workplaceFields.append(QgsField('id', QVariant.Int))
        for key in self.FIELD_MAP:
            workplaceFields.append(QgsField(key, QVariant.Int))
        (homes, homesId) = self.parameterAsSink(
            parameters,
            self.OUTPUT,
            context,
            workplaceFields,
            source.wkbType(),
            source.sourceCrs()
        )

        # Compute the number of steps to display within the progress bar and
        # get features from source
        total = 100.0 / gridOutput.featureCount() if gridOutput.featureCount() else 0
        features = gridOutput.getFeatures()
        currentId = 0

        for current, feature in enumerate(features):
            # Stop the algorithm if cancel button has been clicked
            if feedback.isCanceled():
                return {}
            # use bottom left corner to select point
            search = QgsRectangle(feature['left']-1, feature['bottom']-1,
                feature['left']+1, feature['bottom']+1)

            pointsOutput.selectByRect(search)
            selection = pointsOutput.selectedFeatures()
            if len(selection) == 1:
                currentId += 1
                f = QgsFeature()
                f.setFields(workplaceFields)
                f['id'] = currentId
                for key in self.FIELD_MAP:
                    if self.FIELD_MAP[key] != '':
                        f[key] = int(selection[0][self.FIELD_MAP[key]])

                f['m_below_15'] = f['m_below_5'] + f['m_5_9'] + f['m_10_14']
                f['f_below_15'] = f['f_below_5'] + f['f_5_9'] + f['f_10_14']
                # add grid geometry
                f.setGeometry(feature.geometry())

                # Add a feature to homes
                homes.addFeature(f, QgsFeatureSink.FastInsert)

            # Update the progress bar
            feedback.setProgress(int(current * total))

        return {
           self.OUTPUT: homesId
        }
```

Anhang D: R-Script für Erwerbsstatus Daten

https://firesoft.ch/unigis/velo/employment_status_probabilities.R

```
library(dplyr)
library(stringr)
library(data.table)
library(openxlsx)
# Working Directory setzten
setwd('./daten/')

# Schulbesuchsquoten der 3-31-Jährigen
# https://www.bfs.admin.ch/bfs/de/home/statistiken/kataloge-datenbanken/tabellen.assetdetail.24130072.html
fileNameSchool <- 'je-d-15.02.00.03.xlsx'
# Arbeitsmarktstatus nach Geschlecht, Nationalität, Altersgruppen, Familientyp
fileNameEmploymentStatus <- 'je-d-03.02.00.02.01.xlsx'

xls <- read.xlsx(xlsxFile = fileNameSchool, startRow=6, sheet=1)

# Datatable für Arbeitsmarktstatus vorbereiten
employmentStatus <- data.table(type=character())
# Spaltenname für Frauen und Männer ergänzen
for (g in c('m', 'f')){
  for (age in seq(0, 65, by=5)){
    if(age<5){
      attrName <- sprintf('%s_below_5', g)
```

```
    }else{
      attrName <- sprintf('%s_%s_%s', g, age, age+4)
    }
    employmentStatus <- employmentStatus[,(attrName):=numeric()]
  }
}
# Schüler Zeile vorbereiten
newRow<-list('pupil')
# Daten für Schüler aus XLS lesen
for (g in c('m', 'f')){
  if(g=='f'){
    rows<-c(17,21) # Frauen Obligatorische Schule und Sekundarstufe II
  }else{
    rows<-c(2,6) # Männer Obligatorische Schule und Sekundarstufe II
  }
  for (age in seq(0, 65, by=5)){
    if(age<5){
      newRow <- c(newRow, sum(xls[ rows, c('3','4')])/5)
    }else if(age<30){
      newRow <- c(newRow, sum(xls[ rows, sprintf('%s',seq(age,age+4))])/5)
    }else{
      newRow <- c(newRow, 0)
    }
  }
}
# Datatable um Schüler Zeile ergänzen
employmentStatus<-rbind(employmentStatus,newRow)

# Studenten Zeile vorbereiten
newRow<-list('student')
# Daten für Studenten aus XLS lesen
for (g in c('m', 'f')){
  if(g=='f'){
    rows<-c(27) # Frauen Tertiärstufe
  }else{
    rows<-c(12) # Männer Tertiärstufe
  }
  for (age in seq(0, 65, by=5)){
    if(age<5){
      newRow <- c(newRow, sum(xls[ rows, c('3','4')])/5)
    }else if(age<30){
      newRow <- c(newRow, sum(xls[ rows, sprintf('%s',seq(age,age+4))])/5)
    }else{
      newRow <- c(newRow, 0)
    }
  }
}
# Datatable um Studenten Zeile ergänzen
employmentStatus<-rbind(employmentStatus,newRow)

xls <- read.xlsx(xlsxFile = fileNameEmploymentStatus, startRow=50, sheet=5)
employedRow<-list('employed')
unemployedRow<-list('unemployed')
inactiveRow<-list('inactive')
for (g in c('m', 'f')){
  if(g=='f'){
    startRow <- 38 # Frauen 15-24 Jahre
  }else{
    startRow <- 1 # Männer 15-24 Jahre
  }
  for (age in seq(0, 65, by=5)){
    if(age < 15){
      employedRow <- c(employedRow, 0)
      unemployedRow <- c(unemployedRow, 0)
      inactiveRow <- c(inactiveRow, 0)
    }else{
      if(age < 25){
        addRow <- 0 # 15-24 Jahre
      }else if(age < 40){
        addRow <- 5 # 25-39 Jahre
      }else if(age < 55){
        addRow <- 10 # 40-54 Jahre
      }else if(age < 65){
        addRow <- 15 # 55-64 Jahre
      }else{
        addRow <- 20 # 65 Jahre und älter
      }
      t <- as.numeric(xls[startRow+addRow,'2022'])   # Total
      e <- as.numeric(xls[startRow+addRow+2,'2022']) # Erwerbstätige
      u <- as.numeric(xls[startRow+addRow+3,'2022']) # Erwerbslose gemäss ILO
      i <- as.numeric(xls[startRow+addRow+4,'2022']) # Nichterwerbspersonen
      # Felder ergänzen
      employedRow <- c(employedRow, (100/t*e))
      unemployedRow <- c(unemployedRow, (100/t*u))
      inactiveRow <- c(inactiveRow, (100/t*i))
    }
  }
}
# Datatables um jeweilige Zeile ergänzen
employmentStatus<-rbind(employmentStatus,employedRow)
employmentStatus<-rbind(employmentStatus,unemployedRow)
employmentStatus<-rbind(employmentStatus,inactiveRow)
# Werte von 0.2 und niedriger auf 0 setzen
employmentStatus[employmentStatus<=0.2]<-0

# CSV erstellen
write.table(employmentStatus,
  file = 'employment_status_probabilities.csv',
  row.names=FALSE,col.names=TRUE, sep=';')
```

Anhang E: Processing Script für Network & Intersections

https://firesoft.ch/unigis/velo/network_processingscript.py

```
# -*- coding: utf-8 -*-

"""
***************************************************************************
*                                                                         *
*   This program is free software; you can redistribute it and/or modify  *
*   it under the terms of the GNU General Public License as published by  *
*   the Free Software Foundation; either version 2 of the License, or     *
*   (at your option) any later version.                                   *
*                                                                         *
***************************************************************************
"""
from PyQt5.QtCore import QVariant
from qgis.PyQt.QtCore import QCoreApplication
from qgis.core import (QgsFeature,
                       QgsField,
                       QgsFields,
                       QgsGeometry,
                       QgsGeometryEngine,
                       QgsProcessing,
                       QgsFeatureSink,
                       QgsHstoreUtils,
                       QgsPointXY,
                       QgsProcessingException,
                       QgsProcessingAlgorithm,
                       QgsProcessingParameterProviderConnection,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterFeatureSink,
                       QgsProcessingParameterFile,
                       QgsProcessingParameterNumber,
                       QgsProcessingParameterRasterLayer,
                       QgsProcessingParameterVectorDestination,
                       QgsCoordinateReferenceSystem,
                       QgsProject,
                       QgsRectangle,
                       QgsWkbTypes)
from qgis import processing
import math, json
import networkx as nx

class ThesisNetworkProcessingAlgorithm(QgsProcessingAlgorithm):
    # Constants used to refer to parameters and outputs. They will be
    # used when calling the algorithm from another algorithm, or when
    # calling from the QGIS console.

    INPUT = 'INPUT'
    INPUT_BUFFER = 'outline_buffer'
    INPUT_PGR = 'pgr'
    INPUT_OSM = 'osm'
    INPUT_DB =  'osm_db'
    INPUT_DEM = 'dem'
    INPUT_COVER = 'land_cover'

    OUTPUT1 = 'OUTPUT1'
    OUTPUT2 = 'OUTPUT2'
    OUTPUT3 = 'OUTPUT3'

    TARGET_CRS = '2056' # 2056: LV95, 32633: WGS 84 / UTM zone 33N
    CRS_OPERATION = {
        '2056': '+proj=pipeline +step +proj=unitconvert +xy_in=deg +xy_out=rad '+
            '+step +proj=push +v_3 +step +proj=cart +ellps=WGS84 +step '+
            '+proj=helmert +x=-674.374 +y=-15.056 +z=-405.346 +step +inv '+
            '+proj=cart +ellps=bessel +step +proj=pop +v_3 +step +proj=somerc '+
            '+lat_0=46.9524055555556 +lon_0=7.43958333333333 +k_0=1 +x_0=2600000 '+
            '+y_0=1200000 +ellps=bessel',
        '32633': '+proj=pipeline +step +proj=unitconvert +xy_in=deg +xy_out=rad '+
            '+step +proj=utm +zone=33 +ellps=WGS84'
    }

    def tr(self, string):
        return QCoreApplication.translate('Processing', string)

    def createInstance(self):
        return ThesisNetworkProcessingAlgorithm()

    def name(self):
        return 'network_processing'

    def displayName(self):
        return self.tr('Prepare Network')

    def group(self):
        return self.tr('Master Thesis')

    def groupId(self):
        return 'thesis'

    def shortHelpString(self):
        return self.tr("Prepares the OSM and other data for use in Gama simulation.")

    def initAlgorithm(self, config=None):
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT,
                self.tr('Outline layer'),
                [QgsProcessing.TypeVectorPolygon]
```

```
            )
        )
        self.addParameter(
            QgsProcessingParameterNumber(
                self.INPUT_BUFFER,
                self.tr('Outline Buffer (in metre)'),
                defaultValue=200,
                type=QgsProcessingParameterNumber.Integer
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT_PGR,
                self.tr('osm2pgrouting: Ways (ways)'),
                [QgsProcessing.TypeVectorLine]
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT_OSM,
                self.tr('osm2pgrouting: OSM Tags (osm_ways)'),
                [QgsProcessing.TypeVectorLine]
            )
        )
        self.addParameter(
            QgsProcessingParameterProviderConnection(
                self.INPUT_DB,
                self.tr('Database connection for relation table'),
                'postgres'
            )
        )
        self.addParameter(
            QgsProcessingParameterRasterLayer(
                self.INPUT_DEM,
                self.tr('DEM Raster'),
                [QgsProcessing.TypeRaster]
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT_COVER,
                self.tr('Land cover (AV Bodenbedeckung)'),
                [QgsProcessing.TypeVectorPolygon]
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSink(
                self.OUTPUT1,
                self.tr('Network')
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSink(
                self.OUTPUT2,
                self.tr('Ways with Tags')
            )
        )
        self.addParameter(
            QgsProcessingParameterFeatureSink(
                self.OUTPUT3,
                self.tr('Intersections')
            )
        )

    def processAlgorithm(self, parameters, context, feedback):
        # Retrieve the feature source and sink. The 'dest_id' variable is used
        # to uniquely identify the feature sink, and must be included in the
        # dictionary returned by the processAlgorithm function.
        source = self.parameterAsSource(
            parameters,
            self.INPUT,
            context
        )
        sourcePgr = self.parameterAsSource(
            parameters,
            self.INPUT_PGR,
            context
        )
        sourceOsm = self.parameterAsLayer(
            parameters,
            self.INPUT_OSM,
            context
        )
        dbConnection = self.parameterAsConnectionName(
            parameters,
            self.INPUT_DB,
            context
        )
        sourceDem = self.parameterAsRasterLayer(
            parameters,
            self.INPUT_DEM,
            context
        )

        if source is None:
            raise QgsProcessingException(
                self.invalidSourceError(parameters, self.INPUT))
        if sourcePgr is None:
            raise QgsProcessingException(
                self.invalidSourceError(parameters, self.INPUT_PGR))

        # Add network to hstore tags from pgsql table osm_ways
        feedback.pushInfo('Add network tag from cycle network via relation')
```

```
# https://wiki.openstreetmap.org/wiki/Switzerland/CycleNetwork
sql = "UPDATE osm_ways SET tags = tags || hstore('network', subquery.route) "
sql+= "FROM (SELECT (ARRAY_AGG(hstore(r.tags)->'network' ORDER BY "
sql+= "array_position(ARRAY['icn','ncn','rcn','lcn'], "
sql+= "hstore(r.tags)->'network')))[1] AS route, w.osm_id "
sql+= "FROM osm_ways AS w INNER JOIN planet_osm_rels AS r ON "
sql+= "'network' = ANY(r.tags) AND 'w'||w.osm_id = ANY(r.members) "
sql+= "GROUP BY w.osm_id) AS subquery WHERE osm_ways.osm_id=subquery.osm_id;"
# Reproject OSM network layer
params = {
    'DATABASE': dbConnection,
    'SQL': sql
}
sqlResult = processing.run(
    'native:postgisexecutesql',
    params, context=context, feedback=feedback)

if feedback.isCanceled():
    return {}

# Create buffer from the outline layer
params = {
    'INPUT': parameters[self.INPUT],
    'DISTANCE': parameters[self.INPUT_BUFFER],
    'SEGMENTS': 5,
    'END_CAP_STYLE': 0,
    'JOIN_STYLE': 0,
    'MITER_LIMIT': 2,
    'DISSOLVE': False,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
outlineBuffer = processing.run(
    'native:buffer',
    params, context=context, feedback=feedback)['OUTPUT']

# Clip OSM Network layer
feedback.pushInfo('Prepare OSM network layer')
params = {
    'INPUT': parameters[self.INPUT_PGR],
    'OVERLAY': outlineBuffer,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
osmOutput = processing.run(
    'native:clip',
    params, context=context, feedback=feedback)['OUTPUT']

if feedback.isCanceled():
    return {}

# Reproject OSM network layer
params = {
    'INPUT': osmOutput,
    'TARGET_CRS': QgsCoordinateReferenceSystem('EPSG:'+self.TARGET_CRS),
    'OPERATION': self.CRS_OPERATION[self.TARGET_CRS],
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
osmOutput = processing.run(
    'native:reprojectlayer',
    params, context=context, feedback=feedback)['OUTPUT']

if feedback.isCanceled():
    return {}

# Clip land cover layer
feedback.pushInfo('Prepare land cover layer')
params = {
    'INPUT': parameters[self.INPUT_COVER],
    'METHOD': 0,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
coverOutput = processing.run(
    'native:fixgeometries',
    params, context=context, feedback=feedback)['OUTPUT']
params = {
    'INPUT': coverOutput,
    'OVERLAY': outlineBuffer,
    'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
}
coverOutput = processing.run(
    'native:clip',
    params, context=context, feedback=feedback)['OUTPUT']
if feedback.isCanceled():
    return {}

feedback.pushInfo('Add tags to ways')
waysFields = sourcePgr.fields()
# add landuse_area and landuse_percent
waysFields.append(QgsField('tags', QVariant.String))

(ways, waysId) = self.parameterAsSink(
    parameters,
    self.OUTPUT2,
    context,
    waysFields,
    sourcePgr.wkbType(),
    source.sourceCrs()
)

# Compute the number of steps to display within the progress bar and
# get features from source
total = 100.0 / osmOutput.featureCount() if osmOutput.featureCount() else 0
features = osmOutput.getFeatures()
currentId = 0
```

```
        for current, feature in enumerate(features):
            # Stop the algorithm if cancel button has been clicked
            if feedback.isCanceled():
                return {}

            f = QgsFeature()
            f.setFields(waysFields)
            f.setAttributes(feature.attributes() + [None])
            # add geometry
            f.setGeometry(feature.geometry())

            sourceOsm.selectByExpression('osm_id={}'.format(f['osm_id']))
            selection = sourceOsm.selectedFeatures()
            if len(selection) > 0:
                f.setAttribute('tags', json.dumps(selection[0]['tags']))

            # Add a feature to ways
            ways.addFeature(f, QgsFeatureSink.FastInsert)
            # Update the progress bar
            feedback.setProgress(int(current * total))

        if feedback.isCanceled():
            return {}

        # Extract by Expression
        feedback.pushInfo('Remove additional unnecessary streets')
        # https://wiki.openstreetmap.org/wiki/Key:highway
        hFilter = [
                'proposed', 'construction' # Lifecycle
                'raceway','corridor','platform',
                'elevator','via_ferrata',
                # Autobahn und Autostrasse sind nur für Motorfahrzeugen welche mindestens
                # 80 km/h fahren
                # https://de.wikipedia.org/wiki/Autobahn_und_Autostrasse_(Schweiz)
                'motorway','motorway_link','trunk','trunk_link'
        ]
        params = {
            'INPUT': waysId, # parameters[self.INPUT_OSM],  waysId,
            'EXPRESSION': 'map_get(from_json("tags"), \'highway\') NOT IN (\'' +
                "','".join(hFilter) + '\')',
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        extOutput = processing.run(
            'native:extractbyexpression',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        feedback.pushInfo('Remove disconnected islands')
        # Prepare lines for NetworkX
        params = {
            'INPUT': extOutput,
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        singelpartOutput = processing.run('native:multiparttosingleparts',
            params, context=context, feedback=feedback)['OUTPUT']

        # https://gis.stackexchange.com/questions/184319/finding-disconnected-islands-in-road-network-layer-using-
qgis
        # https://github.com/AfriGIS-South-Africa/disconnected-islands/blob/master/disconnected_islands.py
        G = nx.Graph()
        # construct undirected graph
        for feat in singelpartOutput.getFeatures():
            line = feat.geometry().asPolyline()
            for i in range(len(line)-1):
                G.add_edges_from([((line[i][0], line[i][1]),
                    (line[i+1][0], line[i+1][1]), {'fid': feat['gid']})])

        # evaluate on connected components
        connectedComponents = list(G.subgraph(c) for c in nx.connected_components(G))
        # gather edges and components to which they belong
        fidComp = {}
        for i, graph in enumerate(connectedComponents):
            for edge in graph.edges(data=True):
                fidComp[edge[2].get('fid', None)] = i

        disconnectedIds = list(dict(filter(lambda x: x[1]>0 ,fidComp.items())).keys())
        params = {
            'INPUT': extOutput,
            'EXPRESSION': 'gid NOT IN ('
                  + ','.join('{0}'.format(n) for n in disconnectedIds) + ')',
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        extOutput = processing.run(
            'native:extractbyexpression',
            params, context=context, feedback=feedback)['OUTPUT']

        feedback.pushInfo('Create network layer')
        # Create fields
        networkFields = QgsFields();
        networkFields.append(QgsField('id', QVariant.Int))
        networkFields.append(QgsField('osm_id', QVariant.Int))

        networkFields.append(QgsField('road_name', QVariant.String, '', 254))
        networkFields.append(QgsField('basetype', QVariant.String, '', 254))
        networkFields.append(QgsField('brunnel', QVariant.Int))
        networkFields.append(QgsField('restric_ft', QVariant.Int))
        networkFields.append(QgsField('restric_tf', QVariant.Int))
        networkFields.append(QgsField('oneway_ft', QVariant.Int))
        networkFields.append(QgsField('oneway_tf', QVariant.Int))
        networkFields.append(QgsField('bic_inf_ft', QVariant.String, '', 254))
```

```
networkFields.append(QgsField('bic_inf_tf', QVariant.String, '', 254))
networkFields.append(QgsField('mit_vol_ft', QVariant.Int))
networkFields.append(QgsField('mit_vol_tf', QVariant.Int))
networkFields.append(QgsField('d_route_ft', QVariant.String, '', 254))
networkFields.append(QgsField('d_route_tf', QVariant.String, '', 254))
networkFields.append(QgsField('road_categ', QVariant.String, '', 254))
networkFields.append(QgsField('max_sp_ft', QVariant.Int))
networkFields.append(QgsField('max_sp_tf', QVariant.Int))
networkFields.append(QgsField('ad_edge_ft', QVariant.Int))
networkFields.append(QgsField('ad_edge_tf', QVariant.Int))
networkFields.append(QgsField('parking_ft', QVariant.String, '', 254))
networkFields.append(QgsField('parking_tf', QVariant.String, '', 254))
networkFields.append(QgsField('pavement', QVariant.String, '', 254))
networkFields.append(QgsField('width_lane', QVariant.Double))
networkFields.append(QgsField('gradient_f', QVariant.Int))
networkFields.append(QgsField('gradient_t', QVariant.Int))
networkFields.append(QgsField('rails', QVariant.String, '', 254))
networkFields.append(QgsField('n_lanes_ft', QVariant.Double))
networkFields.append(QgsField('n_lanes_tf', QVariant.Double))
networkFields.append(QgsField('land_use', QVariant.String, '', 254))

(network, networkId) = self.parameterAsSink(
    parameters,
    self.OUTPUT1,
    context,
    networkFields,
    sourcePgr.wkbType(),
    source.sourceCrs()
)

# Compute the number of steps to display within the progress bar and
# get features from source
total = 100.0 / extOutput.featureCount() if extOutput.featureCount() else 0
features = extOutput.getFeatures()
coverFeatures = [feature for feature in coverOutput.getFeatures()]
currentId = 0

for current, feature in enumerate(features):
    # Stop the algorithm if cancel button has been clicked
    if feedback.isCanceled():
        return {}

    tags = json.loads(feature['tags'])

    currentId += 1
    f = QgsFeature()
    f.setFields(networkFields)
    f['id'] = currentId
    f['osm_id'] = feature['osm_id']
    f['road_name'] = feature['name']

    # Only basetype 6-stairway is used in the simulation.
    basetypeList = []
    # Most steps are already filtered in osm2pgrouting, unless they have a
    # tracktype tag (mapconfig_for_bicycles.xml)
    if tags.get('highway') == 'steps':
        basetypeList.append('6')

    f['basetype'] = ';'.join(basetypeList)

    f['brunnel'] = 1 if tags.get('bridge') else 2 if tags.get('tunnel') else 0

    f['restric_ft'] = 0
    f['restric_tf'] = 0
    # https://wiki.openstreetmap.org/wiki/Key:oneway
    f['oneway_ft'] = 0
    f['oneway_tf'] = 0
    if ((tags.get('oneway') == 'yes' and tags.get('oneway:bicycle') != 'no')
            or tags.get('oneway:bicycle') == 'yes'):
        f['oneway_ft'] = 1
        f['oneway_tf'] = 0
        if(tags.get('foot') == 'no'):
            f['restric_tf'] = 2
        else:
            f['restric_tf'] = 1
    elif ((tags.get('oneway') == '-1' and tags.get('oneway:bicycle') != 'no')
            or tags.get('oneway:bicycle') == '-1'):
        f['oneway_ft'] = 0
        f['oneway_tf'] = 1
        if(tags.get('foot') == 'no'):
            f['restric_ft'] = 2
        else:
            f['restric_ft'] = 1
    # https://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Access_restrictions#Switzerland
    if(tags.get('bicycle') == 'no'
            or ((tags.get('highway') in ['pedestrian', 'bridleway', 'footway']
                and (tags.get('bicycle') != 'yes'
                or tags.get('bicycle') != 'designated')))):
        if(tags.get('foot') == 'no'
                or (tags.get('highway') == 'bridleway'
                    and tags.get('foot') != 'yes')):
            f['restric_ft'] = 2
            f['restric_tf'] = 2
        else:
            f['restric_ft'] = 1
            f['restric_tf'] = 1

    # https://wiki.openstreetmap.org/wiki/Key:cycleway
    # https://wiki.openstreetmap.org/wiki/Key:cycleway:right
    f['bic_inf_ft'] = 'no'
    f['bic_inf_tf'] = 'no'
    if tags.get('highway') == 'cycleway':
        if f['restric_ft'] == 0:
```

```
            f['bic_inf_ft'] = 'bicycle_way'
        if f['restric_tf'] == 0:
            f['bic_inf_tf'] = 'bicycle_way'
    # https://wiki.openstreetmap.org/wiki/Switzerland/Map_Features#Cycle_and_Foot_Ways
    elif ((tags.get('highway') in ['footway', 'path', 'track']
                and tags.get('bicycle') == 'designated')
            or (tags.get('highway') == 'pedestrian'
                and tags.get('bicycle') == 'yes')):
        if f['restric_ft'] == 0:
            f['bic_inf_ft'] = 'mixed_way'
        if f['restric_tf'] == 0:
            f['bic_inf_tf'] = 'mixed_way'
    elif tags.get('cycleway'):
        # https://wiki.openstreetmap.org/wiki/Key:cycleway#Cycle_lanes
        if tags.get('cycleway') == 'opposite_lane':
            if f['oneway_ft'] == 1:
                f['bic_inf_tf'] = 'bicycle_lane'
            else:
                f['bic_inf_ft'] = 'bicycle_lane'
        # https://wiki.openstreetmap.org/wiki/Key:cycleway#Cycle_tracks
        elif tags.get('cycleway') == 'opposite_track':
            if f['oneway_ft'] == 1:
                f['bic_inf_tf'] = 'bicycle_way'
            else:
                f['bic_inf_ft'] = 'bicycle_way'
        else:
            f['bic_inf_ft'] = tags.get('cycleway')
            f['bic_inf_tf'] = f['bic_inf_ft']
    if tags.get('cycleway:both'):
        f['bic_inf_ft'] = tags.get('cycleway:both')
        f['bic_inf_tf'] = f['bic_inf_ft']
    if tags.get('cycleway:right'):
        f['bic_inf_ft'] = tags.get('cycleway:right')
    if tags.get('cycleway:left'):
        f['bic_inf_tf'] = tags.get('cycleway:left')

    for bicInf in ['bic_inf_ft', 'bic_inf_tf']:
        if f[bicInf] == 'track':
            f[bicInf] = 'bicycle_way'
        elif f[bicInf]  in ['lane', 'opposite', 'yes']:
            f[bicInf] = 'bicycle_lane'
        elif f[bicInf] == 'share_busway':
            f[bicInf] = 'bus_lane'
        elif f[bicInf] == 'separate':
            f[bicInf] = 'no'

    # Only ncn and rcn are recommended as network tag, but lcn is also used for
    # swiss cycle routes
    # https://wiki.openstreetmap.org/wiki/Switzerland/CycleNetwork
    route = 'no'
    if tags.get('network') == 'icn':
        route = 'international'
    elif tags.get('network') == 'ncn':
        route = 'national'
    elif tags.get('network') == 'rcn':
        route = 'regional'
    elif tags.get('network') == 'lcn':
        route = 'local'
    f['d_route_ft'] = route
    f['d_route_tf'] = route

    roadCategory = None
    # https://wiki.openstreetmap.org/wiki/DE:Tag:highway%3Dliving_street#Verwandte_Tags
    if tags.get('highway') in ['living_street', 'pedestrian']:
        roadCategory = 'calmed'
    # https://wiki.openstreetmap.org/wiki/Switzerland/Map_Features#Cycle_and_Foot_Ways
    elif (tags.get('highway') == 'cycleway'
            or (tags.get('highway') == 'footway'
                and tags.get('bicycle') == 'designated')
            or tags.get('motor_vehicle') == 'no'
            or (tags.get('motorcar') == 'no'
                and tags.get('motorcycle') == 'no')
            or (tags.get('highway') == 'pedestrian'
                and tags.get('bicycle') == 'yes')
            or (tags.get('highway') == 'path'
                and tags.get('bicycle') == 'designated')):
        roadCategory = 'no_mit'
    elif tags.get('highway') in ['primary', 'primary_link']:
        roadCategory = 'primary'
    elif tags.get('highway') in ['secondary', 'secondary_link', 'tertiary',
            'tertiary_link']:
        roadCategory = 'secondary'
    elif tags.get('highway') == 'residential':
        roadCategory = 'residential'
    elif tags.get('highway') in ['service', 'unclassified', 'track']:
        roadCategory = 'service'
    elif tags.get('highway') == 'path':
        roadCategory = 'path'
    f['road_categ'] = roadCategory

    f['max_sp_ft'] = feature['maxspeed_forward']
    f['max_sp_tf'] = feature['maxspeed_backward']

    f['rails'] = 'yes' if tags.get('railway') else 'no'

    # https://wiki.openstreetmap.org/wiki/Street_parking
    # https://wiki.openstreetmap.org/wiki/Key:parking:lane
    p_left = None
    p_right = None
    if tags.get('parking:both'):
        p_left = tags.get('parking:both')
        p_right = tags.get('parking:both')
    elif tags.get('parking:lane:both'):
```

```
    p_left = tags.get('parking:lane:both')
    p_right = tags.get('parking:lane:both')
if tags.get('parking:right'):
    p_right = tags.get('parking:right')
elif tags.get('parking:lane:right'):
    p_right = tags.get('parking:lane:right')
if tags.get('parking:left'):
    p_left = tags.get('parking:left')
elif tags.get('parking:lane:left'):
    p_left = tags.get('parking:lane:left')
p_restriction = ['no', 'no_parking', 'no_standing', 'no_stopping']
f['parking_ft']='yes' if p_right and p_right not in p_restriction else 'no'
f['parking_tf']='yes' if p_left and p_left not in p_restriction else 'no'

# https://wiki.openstreetmap.org/wiki/Key:surface
pavementMap = {
    'asphalt': ['asphalt', 'chipseal', 'concrete', 'concrete:plates'],
    'gravel': ['gravel', 'fine_gravel', 'compacted'],
    'soft': ['unpaved', 'ground'],
    'cobble': ['paving_stones', 'sett', 'unhewn_cobblestone',
        'cobblestone', 'brick', 'paving_stones', 'pebblestone']
}
surface = tags.get('surface')
if surface:
    for pType in pavementMap:
        if surface in pavementMap[pType]:
            f['pavement'] = pType
            break

gradient = 0
# calculate gradient only if not a tunnel or bridge
if f['brunnel'] == 0:
    # get start and end point of line geometry
    parts = feature.geometry().constGet()
    start = parts[0][0]
    end = parts[-1][-1]
    # get elevation
    startEle, res = sourceDem.dataProvider().sample(QgsPointXY(start), 1)
    endEle, res = sourceDem.dataProvider().sample(QgsPointXY(end), 1)
    # calculate slope
    slope = (endEle - startEle) / feature.geometry().length()*100
    # gradient category according to classification for up- and downhill:
    # * -1.5 % <“0”<1,5 %; 1,5 % <“1”< 3 %; 3 % <“2”< 6 %; 6 % <“3”< 12 %;
    # “4” > 12 %; -1,5 % >“-1”> -3 %; -3 % >“-2”> -6 %; -6 % >“-3”> -12 %;
    # “-4” < -12 %
    if abs(slope) < 1.5:
        gradient = 0
    elif abs(slope) < 3:
        if slope < 0:
            gradient = -1
        else:
            gradient = 1
    elif abs(slope) < 6:
        if slope < 0:
            gradient = -2
        else:
            gradient = 2
    elif abs(slope) < 12:
        if slope < 0:
            gradient = -3
        else:
            gradient = 3
    else:
        if slope < 0:
            gradient = -4
        else:
            gradient = 4

f['gradient_f'] = gradient
f['gradient_t'] = gradient * (-1)

f['n_lanes_ft'] = 1
f['n_lanes_tf'] = 1
if f['oneway_ft'] == 1:
    f['n_lanes_tf'] = 0
elif f['oneway_tf'] == 1:
    f['n_lanes_ft'] = 0
lanes = 1
# https://wiki.openstreetmap.org/wiki/Key:lanes
if tags.get('lanes'):
    lanes = int(tags.get('lanes'))
    if f['oneway_ft'] == 1:
        f['n_lanes_ft'] = lanes
    elif f['oneway_tf'] == 1:
        f['n_lanes_tf'] = lanes
    else:
        f['n_lanes_ft'] = math.ceil(lanes/2)
        f['n_lanes_tf'] = f['n_lanes_ft']
        if tags.get('lanes:forward'):
            f['n_lanes_ft'] = int(tags.get('lanes:forward'))
            if tags.get('lanes:both_ways'):
                f['n_lanes_ft'] = f['n_lanes_ft']
                f['n_lanes_ft']+= int(tags.get('lanes:both_ways'))
        if tags.get('lanes:backward'):
            f['n_lanes_tf'] = int(tags.get('lanes:backward'))
            if tags.get('lanes:both_ways'):
                f['n_lanes_tf'] = f['n_lanes_tf']
                f['n_lanes_tf']+= int(tags.get('lanes:both_ways'))
            if not tags.get('lanes:forward'):
               f['n_lanes_ft'] = lanes - int(tags.get('lanes:backward'))
        elif tags.get('lanes:forward'):
            f['n_lanes_tf'] = lanes - int(tags.get('lanes:forward'))
```

```
            lanes = max(1, lanes); # sicherstellen dass der Wert mindestens 1 ist
            if tags.get('width'):
                try:
                    f['width_lane'] = float(tags.get('width')) / lanes
                except:
                    feedback.pushInfo('{} width: {}'.format(
                        f['osm_id'],tags.get('width')))
            else:
                width_lane = None
                # calculate lane width only if not a tunnel or bridge
                if f['brunnel'] == 0:
                    fGE = QgsGeometry.createGeometryEngine(
                        feature.geometry().constGet())
                    fGE.prepareGeometry();
                    subset = [poly for poly in coverFeatures if fGE.intersects(
                        poly.geometry().constGet())]
                    if len(subset) > 0:
                        coverFeature = None
                        maxLength = 0
                        for p in subset:
                            i = fGE.intersection(p.geometry().constGet())
                            length = i.length()
                            # Ignore short distances (less than 2 meters)
                            if 2 < length and length > maxLength:
                                coverFeature = p
                                maxLength = length
                        if coverFeature is not None:
                            # Calculating average width of polygon
                            # https://gis.stackexchange.com/questions/20279/calculating-average-width-of-polygon/
181801#20282
                            p = coverFeature.geometry().length()
                            try:
                                    width = (p - math.sqrt(
                                        p**2 - 16*coverFeature.geometry().area())
                                        ) / 4
                            except:
                                feedback.pushInfo('{} p: {} a: {}'.format(
                                    f['osm_id'],p, coverFeature.geometry().area()))
                            if width < fGE.length():
                                f['width_lane'] = width / lanes

            # add grid geometry
            f.setGeometry(feature.geometry())

            # Add a feature to networks
            network.addFeature(f, QgsFeatureSink.FastInsert)

            # Update the progress bar
            feedback.setProgress(int(current * total))

        feedback.pushInfo('Generate intersections')
        # Create fields
        intersectionsFields = QgsFields();
        intersectionsFields.append(QgsField('id', QVariant.Int))
        (intersections, intersectionsId) = self.parameterAsSink(
            parameters,
            self.OUTPUT3,
            context,
            intersectionsFields,
            QgsWkbTypes.Point,
            source.sourceCrs()
        )
        currentId = 0
        # Extract network by brunnel and generat intersections
        for brunnel in [0, 1, 2]:
            feedback.pushInfo('brunnel: {}'.format(brunnel))
            # Extract by Attribute
            params = {
                'FIELD': 'brunnel',
                'INPUT': networkId,
                'OPERATOR': 0,  # =
                'VALUE': brunnel,
                'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
            }
            intOutput = processing.run(
                'native:extractbyattribute',
                params, context=context, feedback=feedback)['OUTPUT']

            if feedback.isCanceled():
                return {}

            # https://nkaza.github.io/post/intersection-density-from-osm-using-qgis-r/
            # https://gis.stackexchange.com/questions/321985/find-intersections-of-roads-in-qgis
            # http://www.qgistutorials.com/en/docs/3/calculating_intersection_density.html
            # Dissolve all lines without attribute
            params = {
                'INPUT': intOutput,
                'FIELD': [],
                'SEPARATE_DISJOINT': True,
                'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
            }
            intOutput = processing.run(
                "native:dissolve",
                params, context=context, feedback=feedback)['OUTPUT']

            if feedback.isCanceled():
                return {}
            # Multiparts to singleparts
            params = {
                'INPUT': intOutput,
                'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
            }
            intOutput = processing.run(
```

```
            'native:multiparttosingleparts',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}
        # Line Intersections
        params = {
            'INPUT': intOutput,
            'INTERSECT': intOutput,
            'INPUT_FIELDS': ['id'],
            'INTERSECT_FIELDS': [],
            'INTERSECT_FIELDS_PREFIX': '',
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        intOutput = processing.run(
            'native:lineintersections',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}
        # Remove duplicate geometries
        params = {
            'INPUT': intOutput,
            'OUTPUT': QgsProcessing.TEMPORARY_OUTPUT
        }
        intOutput = processing.run(
            'native:deleteduplicategeometries',
            params, context=context, feedback=feedback)['OUTPUT']

        if feedback.isCanceled():
            return {}

        features = intOutput.getFeatures()

        for current, feature in enumerate(features):
            # Stop the algorithm if cancel button has been clicked
            if feedback.isCanceled():
                return {}

            currentId += 1
            f = QgsFeature()
            f.setFields(intersectionsFields)
            f['id'] = currentId

            # add grid geometry
            f.setGeometry(feature.geometry())

            # Add a feature to intersections
            intersections.addFeature(f, QgsFeatureSink.FastInsert)

            if feedback.isCanceled():
                return {}

    return {
       self.OUTPUT1: networkId,
       self.OUTPUT2: waysId,
       self.OUTPUT3: intersectionsId,
    }
```

Anhang F: Bicycle Model Version 2.1

https://firesoft.ch/unigis/velo/bicycle_model.gaml

```
/***
* Name: Bicycle model v.2.1
* Author: Dana Kaziyeva, Gudrun Wallentin, Martin Loidl, Thomas Stadelmann
* Description: The purpose of the model is to generate disaggregated traffic flow distribution of cyclists at the
* regional scale level. The human decision-making is based on derived assumptions from the mobility survey data.
* The model's result is the emergent bicycle traffic flow pattern at the high spatial and temporal level of detail.
* To use it for another study area: change input data: fileHomePlaces, fileWorkPlaces, fileFacilities,
* fileCountingStations, fileRoads, fileIntersections, fileCityOutline, fileRegionOutline
***/
model bicyclemodel

global {
   // Input data
   // residential data (250m resolution) with demographic attributes
   file fileHomePlaces <- shape_file("../includes/model_input/shapefiles/homes.shp");
   // employees data (100m resolution) with the attribute of the number of registered employees
   file fileWorkPlaces <- file("../includes/model_input/shapefiles/workplaces.shp");
   // facilities dataset
   file fileFacilities <- file("../includes/model_input/shapefiles/facilities.shp");
   // bicycle counting stations dataset
   file fileCountingStations <- file("../includes/model_input/shapefiles/counting_stations.shp");
   // network dataset with "null" values represented as "-9999"
   file fileRoads <- shape_file("../includes/model_input/shapefiles/network.shp");
   // street intersections dataset
   file fileIntersections <- file("../includes/model_input/shapefiles/intersections.shp");
   // city outline dataset
   file fileCityOutline <- file("../includes/model_input/shapefiles/outline_city.shp");
   // region outline dataset
   file fileRegionOutline <- file("../includes/model_input/shapefiles/outline_region.shp");
   // activity probabilities by position
   matrix activityMatrix <- matrix(file("../includes/model_input/csv_files/activity_probabilities.csv"));
   // departure time probabilities by activity type and by hour
   matrix timeMatrix <- matrix(file("../includes/model_input/csv_files/time_probabilities.csv"));
   // mode probabilities by activity type and spatial extent: city/region
   matrix modeMatrix <- matrix(file("../includes/model_input/csv_files/mode_probabilities.csv"));
   // work duration probabilities by gender
   matrix durationMatrix <- matrix(file("../includes/model_input/csv_files/work_duration_probabilities.csv"));
```

```
// observed counting data at counting stations for validation. Time interval should be the same as the
// "countingStationTimeInterval"
matrix countsMatrix <- matrix(
   csv_file("../includes/model_input/csv_files/hourly_counts_stations.csv", ';', '"', false)
);
// employment status probabilities by gender and age group
matrix employmentStatusMatrix <- matrix(
   csv_file("../includes/model_input/csv_files/employment_status_probabilities.csv", ';', '"', false)
);

// Output data path names
// the heatmap of bicycle traffic volume on a network
string heatmapFileName <- "../includes/model_output/[simulation]/heatmap.shp";
// the number of moving cyclists by trip purpose
string activeCyclistsFileName <- "../includes/model_output/[simulation]/active_cyclists.csv";
// the number of traversed cyclists at counting stations
string countsFileName <- "../includes/model_output/[simulation]/counts.csv";
// bicycle trips with travel information
string tripsFileName <- "../includes/model_output/[simulation]/trips.txt";
// residential data with added employment states
string homesFileName <- "../includes/model_output/[simulation]/homes.shp";
// the heatmap of bicycle traffic volume on a network
string unaccessibleFileName <- "../includes/model_output/[simulation]/unaccessible_location.csv";

// Model parameters
// spatial extent
geometry shape <- envelope(fileRegionOutline) + envelope(fileHomePlaces) + envelope(fileWorkPlaces);
geometry cityOutline; //city boundaries
float step <- 1 #mn; //simulation step: 1 minute
float simulationStartTime; //machine time at the beginning of simulation run (excluding initialization time)
string crs <- 'EPSG:2056'; //coordinate reference system EPSG:2056 (LV95) / EPSG:32633 (WGS 84 / UTM zone 33N)
// the user-defined time interval to save the number of traverses on a network, in cycles (= minutes)
int networkTimeInterval;
// the user-defined time interval to save the number of traverses at stations, in cycles (= minutes)
int countingStationTimeInterval;
// the user-defined time interval to save the number of actively moving cyclists, in cycles (= minutes)
int activeCylistsTimeInterval;

// Network variables
graph theGraph; //bidirectional network graph composed of road species
map<road, float> perimeterWeights; //perimeter weights for roads
map<road, float> cyclingWeights; //road weights that define routing behaviour of cyclists
// user-defined routing algorithm for cyclists: "shortest path", "safest path". Deafault is "safest path".
string routingAlgorithm;

// Attribute weights needed for network assessment and calculation of safety index
float bicycleInfrastructureWeight; //weight for bicycle infrastructure
float mitVolumeWeight; //weight for motorized traffic volume attribute
float designatedRouteWeight; //weight for designated route attribute
float roadCategoryWeight; //weight for road category attribute
float maxSpeedWeight; //weight for maximal speed attribute
float adjacentEdgeWeight; //weight for adjacent edge attribute
float parkingWeight; //weight for parking attribute
float pavementWeight; //weight for pavement attribute
float widthLaneWeight; //weight for width lane attribute
float gradientWeight; //weight for gradient attribute
float railsWeight; //weight for rails attribute
float numberLaneWeight; //weight for lane number attribute
float landuseWeight; //weight for landuse attribute
float designatedRouteAdjusted; //adjusted weight for constraction status attribute
float railsAdjusted; //adjusted weight for rails attribute
float pavementAdjusted; //adjusted weight for pavement
float gradientAdjusted; //adjusted weight for gradient
float bridgeValue; //bridge value
float pushValue; //push value

// Activity probabilities depending on employment_status and activity position (ordinal number of activity in
// activity chain, 0-7). Activity types: "home","other_place","work","business",
// "education"("school","university"),"shop","authority","doctor","recreation","bringing"
map<string, map<int, map<string, float>>> activityProbabilities;

// Starting time probabilities by activity type and actvity position (0-7). The distribution of the departure
// time probabilities is represented with 0-100% for every hour in the 24 hours range
map<string, map<int, map<int, float>>> timeProbabilities;
map<string, map<int, float>> timeThresholds;

//Mode probabilities by activity type and spatial extent (city/region). Modes: "walk","bike","car",
// "car_passenger","public_transport","other_mode"
map<string, map<bool, map<string, float>>> modeProbabilities;

//Work duration probabilities by gender
map<string, map<map<string, int>, float>> workDurationProbabilities;
point stationChartSize;
bool repeatedFlag <- false;

// Initialzation
init {
   heatmapFileName <- replace(heatmapFileName, '[simulation]', name);
   activeCyclistsFileName <- replace(activeCyclistsFileName, '[simulation]', name);
   countsFileName <- replace(countsFileName, '[simulation]', name);
   tripsFileName <- replace(tripsFileName, '[simulation]', name);
   homesFileName <- replace(homesFileName, '[simulation]', name);
   unaccessibleFileName <- replace(unaccessibleFileName, '[simulation]', name);
   float initializationStartTime <- machine_time; //the initialization starts
   // Create the city outline
   cityOutline <- geometry(fileCityOutline);
   remove fileCityOutline from: self;
   do prepareHomes;
   // Create species
   do createRoads;
   do createIntersections;
   do createFacilities;
   do createCountingStations;
```

```
    do createPersons;

    // Import decision-making rules and create initial mobility schedules for persons
    do prepareProbabilities;
    do calculateFirstActivity;

    // Create the "trips" file
    save "self;gender;age;activityId;activityType;startingTime;endingTime;durationTime;travelTime;mode;speed;"
       +"tripLength;tripCityShare;intersections;trip_geom;" to: tripsFileName format: "text" rewrite: false;
    write "duration of initialization: " + (machine_time - initializationStartTime) / 1000; //initialization time
}

action prepareHomes {
    float time_prepareHomes <- machine_time;
    list<geometry> homes;
    map<string, list<int>> possibleHomes; // Hash with arrays of IDs per age/gender group
    // Attributes lists
    list<string> mAttNames <- ['m_below_5', 'm_5_9', 'm_10_14', 'm_15_19', 'm_20_24', 'm_25_29', 'm_30_34',
       'm_35_39', 'm_40_44', 'm_45_49', 'm_50_54', 'm_55_59', 'm_60_64', 'm_65_69'];
    list<string> fAttNames <- ['f_below_5', 'f_5_9', 'f_10_14', 'f_15_19', 'f_20_24', 'f_25_29', 'f_30_34',
       'f_35_39', 'f_40_44', 'f_45_49', 'f_50_54', 'f_55_59', 'f_60_64', 'f_65_69'];
    list<string> mPensAttNames <- ['m_65_69', 'm_70_74', 'm_75_79', 'm_80_84', 'm_85_89', 'm_over_90'];
    list<string> fPensAttNames <- ['f_65_69', 'f_70_74', 'f_75_79', 'f_80_84', 'f_85_89', 'f_over_90'];
    list<string> statusAttNames <- ['m_below_15', 'f_below_15', 'm_employed', 'm_unemploy', 'm_pension',
       'm_students', 'm_inactive', 'm_emp_unk', 'f_employed', 'f_unemploy', 'f_pension', 'f_students',
       'f_inactive', 'f_emp_unk', 'pupils', 'students'];
    loop home over: fileHomePlaces {
       // prepare home attributs
       home.attributes['id'] <- int(home.attributes['id']);
       home.attributes['residents'] <- int(home.attributes['residents']);
       home.attributes['m_tot'] <- int(home.attributes['m_tot']);
       home.attributes['f_tot'] <- int(home.attributes['f_tot']);
       home.attributes['m_pupils'] <- 0;
       home.attributes['f_pupils'] <- 0;
       loop att over: statusAttNames {
          home.attributes[att] <- 0;
       }
       loop att over: mAttNames + fAttNames {
          // Initialize if empty
          if possibleHomes[att] = nil {
             possibleHomes[att] <- [];
          }
          // Add the ID as many times as the age/gender group contains people
          loop times: int(home.attributes[att]) {
             possibleHomes[att] <- possibleHomes[att] + [int(home.attributes['id'])];
          }
       }

       homes <- homes + [home];
    }
    homes <- calculateAttribute(homes, possibleHomes, 'm_employed', mAttNames, 'm_tot');
    homes <- calculateAttribute(homes, possibleHomes, 'f_employed', fAttNames, 'f_tot');
    homes <- calculateAttribute(homes, possibleHomes, 'm_pupils', mAttNames, 'm_tot');
    homes <- calculateAttribute(homes, possibleHomes, 'f_pupils', fAttNames, 'f_tot');
    homes <- calculateAttribute(homes, possibleHomes, 'm_students', mAttNames, 'f_tot');
    homes <- calculateAttribute(homes, possibleHomes, 'f_students', fAttNames, 'f_tot');
    homes <- calculateAttribute(homes, possibleHomes, 'm_unemploy', mAttNames, 'm_tot');
    homes <- calculateAttribute(homes, possibleHomes, 'f_unemploy', fAttNames, 'f_tot');
    // Remove group _65_69 as these are already included in PensAttNames
    remove 'm_65_69' from: mAttNames;
    remove 'f_65_69' from: fAttNames;
    loop home over: homes {
       home.attributes['pupils'] <- int(home.attributes['m_pupils']) + int(home.attributes['f_pupils']);
       home.attributes['students'] <- int(home.attributes['m_students']) + int(home.attributes['f_students']);
       int inactive <- 0;
       // Calculate male inactive and pension values
       if int(home.attributes['m_tot']) > 0 {
          // The total minus all assigned employment attributes
          inactive <- max(0, int(home.attributes['m_tot']) - int(home.attributes['m_pupils'])
             - int(home.attributes['m_students']) - int(home.attributes['m_employed'])
             - int(home.attributes['m_unemploy']));
          if (inactive > 0) {
             int pensioner <- 0;
             int below65 <- 0;
             // Add up all people under 65
             loop pensAtt over: mAttNames {
                below65 <- below65 + int(home.attributes[pensAtt]);
             }
             // Add up all people aged 65 and over
             loop pensAtt over: mPensAttNames {
                pensioner <- pensioner + int(home.attributes[pensAtt]);
             }
             // inactive multiplied the ratio of people under 65 and 65 and older
             home.attributes['m_inactive'] <- round(inactive * 1 / (below65 + pensioner) * below65);
             home.attributes['m_pension'] <- inactive - int(home.attributes['m_inactive']);
          }
       }
       // Calculate female inactive and pension values
       if int(home.attributes['f_tot']) > 0 {
          // The total minus all assigned employment attributes
          inactive <- max(0, int(home.attributes['f_tot']) - int(home.attributes['f_pupils'])
             - int(home.attributes['f_students']) - int(home.attributes['f_employed'])
             - int(home.attributes['f_unemploy']));
          if (inactive > 0) {
             int pensioner <- 0;
             int below65 <- 0;
             // Add up all people under 65
             loop pensAtt over: fAttNames {
                below65 <- below65 + int(home.attributes[pensAtt]);
             }
             // Add up all people aged 65 and over
             loop pensAtt over: fPensAttNames {
                pensioner <- pensioner + int(home.attributes[pensAtt]);
```

```
            }
            // inactive multiplied the ratio of people under 65 and 65 and older
            home.attributes['f_inactive'] <- round(inactive * 1 / (below65 + pensioner) * below65);
            home.attributes['f_pension'] <- inactive - int(home.attributes['f_inactive']);
         }
      }
   }
   // Save homes with added Attributes as shape File
   save homes to: homesFileName format: "shp" crs: crs attributes:
      ['id', 'residents'] + ['m_tot'] + mAttNames + mPensAttNames + ['f_tot']
      + fAttNames + fPensAttNames + statusAttNames;
   remove employmentStatusMatrix from: self;
   remove fileHomePlaces from: self;
   // replace residential data with new homes shape file
   self.fileHomePlaces <- shape_file(homesFileName);
   write "Homes are prepared: " + length(fileHomePlaces) + ". Execution time: "
      + (machine_time - time_prepareHomes) / 1000 + " sec";
}
list<geometry> calculateAttribute (list<geometry> homes, map<string, list<int>> possibleHomes, string attName,
   list attNames, string totAtt) {
   int rowNo;
   list<string> addCountAtt <- [];
   // Set row by attribute name for employmentStatusMatrix
   switch (attName) {
      match_one ['m_employed', 'f_employed'] {
         rowNo <- 3;
      }
      match_one ['m_pupils', 'f_pupils'] {
         rowNo <- 1;
      }
      match_one ['m_students', 'f_students'] {
         rowNo <- 2;
      }
      match_one ['m_unemploy', 'f_unemploy'] {
         rowNo <- 4;
         // Values from pupils and studens are included, as a pupil or student can also be classified as
         // employed, but only in very exceptional cases as unemployed.
         if (attName = 'm_unemploy') {
            addCountAtt <- ['m_pupils', 'm_students'];
         } else {
            addCountAtt <- ['f_pupils', 'f_students'];
         }
      }
   }
   // loop over age group attributes of the current gender
   loop att over: attNames {
      possibleHomes[att] <- shuffle(possibleHomes[att]);
      // Set column by attribute possition in attNames for employmentStatusMatrix
      int columnNo <- (attNames index_of att) + 1;
      int i <- 0;
      int j <- 0;
      int l <- length(possibleHomes[att]);
      // Calculate the number of people for the current employment status from the age/gender group
      int times <- min(l, round(l / 100 * float(employmentStatusMatrix[columnNo, rowNo])));
      // Loop over the array of the current age/gender group until either "times" or the end is reached
      loop while: i < times and j < l {
         int homeId <- possibleHomes[att][j];
         geometry home <- homes first_with (each.attributes['id'] = homeId);
         int addCount <- 0;
         // Include additional attributes (used only for unemployed)
         loop addAttName over: addCountAtt {
            addCount <- addCount + int(home.attributes[addAttName]);
         }
         // Add as long as total is not reached
         if int(home.attributes[totAtt]) > (int(home.attributes[attName]) + addCount) {
            home.attributes[attName] <- int(home.attributes[attName]) + 1;
            i <- i + 1;
         }
         j <- j + 1;
      }
   }
   return homes;
}
// Create "facility" species
action createFacilities {
   create facility from: fileHomePlaces with: [facilityPopulation::int(read("residents"))] {
      facilityType <- "other_place"; //"other_place" facilities are homes of persons
      if facilityPopulation = 0 {
         do die;
      }
   }
   create facility from: fileWorkPlaces with: [facilityPopulation::int(read("employees"))] {
      facilityType <- "work";
      if facilityPopulation = 0 {
         do die;
      }
   }
   create facility from: fileWorkPlaces with: [facilityPopulation::int(read("employees"))] {
      facilityType <- "business";
      if facilityPopulation = 0 {
         do die;
      }
   }
   create facility from: fileFacilities with: [facilityType::string(read("type"))] {
      if length(road overlapping (self)) > 0 {
         // select nearby location, because GAMA generates an error when calculating routes from points that
         // intersect link vertices of the graph
         location <- {location.x + 0.000001, location.y, 0.0};
      }

   }
   remove fileFacilities from: self;
}
```

```
// Create "road" species.
action createRoads {
// The attributes of network file are characterized by direction: the "ft" is from-to direction, the "tf" is
// to-from direction.
   loop link over: fileRoads {
      float safetyInd; // safety index
      float safetyInd_opposite; //safety index of opposite way
      int restrict; //restriction level, explained in the "calculateSafetyIndex" section
      int restrict_opposite; //restriction level of opposite way, explained in the "calculateSafetyIndex" section
      geometry geom; // link geometry

      // Calculate a safety index and geometry according to a link direction
      if int(link.attributes['oneway_ft']) = 1 and int(link.attributes['oneway_tf']) = 0 {
         // if a link has a single direction "FT"
         safetyInd <- calculateSafetyIndex(int(link.attributes['link_id']), int(link.attributes['brunnel']),
            string(link.attributes['basetype']), string(link.attributes['bic_inf_ft']),
            int(link.attributes['mit_vol_ft']), string(link.attributes['d_route_ft']),
            string(link.attributes['road_categ']), int(link.attributes['max_sp_ft']),
            int(link.attributes['ad_edge_ft']), string(link.attributes['parking_ft']),
            string(link.attributes['pavement']), int(link.attributes['width_lane']),
            int(link.attributes['grad_ft']), string(link.attributes['rails']),int(link.attributes['n_lanes_ft']),
            string(link.attributes['land_use']), int(link.attributes['restric_ft']));
         restrict <- int(link.attributes['restric_ft']);
         geom <- polyline(link.points);
      } else if int(link.attributes['oneway_ft']) = 0 and int(link.attributes['oneway_tf']) = 1 {
         // if a link has a single direction "TF"
         safetyInd <- calculateSafetyIndex(int(link.attributes['link_id']), int(link.attributes['brunnel']),
            string(link.attributes['basetype']), string(link.attributes['bic_inf_tf']),
            int(link.attributes['mit_vol_tf']), string(link.attributes['d_route_tf']),
            string(link.attributes['road_categ']), int(link.attributes['max_sp_tf']),
            int(link.attributes['ad_edge_tf']), string(link.attributes['parking_tf']),
            string(link.attributes['pavement']), int(link.attributes['width_lane']),
            int(link.attributes['grad_tf']), string(link.attributes['rails']),int(link.attributes['n_lanes_tf']),
            string(link.attributes['land_use']), int(link.attributes['restric_tf']));
         restrict <- int(link.attributes['restric_tf']);
         geom <- polyline(reverse(link.points)); //reverse a link geometry to have an opposite direction
      } else {
         // If a link has both  "FT" and "TF" directions
         safetyInd <- calculateSafetyIndex(int(link.attributes['link_id']), int(link.attributes['brunnel']),
            string(link.attributes['basetype']), string(link.attributes['bic_inf_ft']),
            int(link.attributes['mit_vol_ft']), string(link.attributes['d_route_ft']),
            string(link.attributes['road_categ']), int(link.attributes['max_sp_ft']),
            int(link.attributes['ad_edge_ft']), string(link.attributes['parking_ft']),
            string(link.attributes['pavement']), int(link.attributes['width_lane']),
            int(link.attributes['grad_ft']), string(link.attributes['rails']),int(link.attributes['n_lanes_ft']),
            string(link.attributes['land_use']), int(link.attributes['restric_ft']));
         restrict <- int(link.attributes['restric_ft']);
         geom <- polyline(link.points);
         safetyInd_opposite
            <- calculateSafetyIndex(int(link.attributes['link_id']),int(link.attributes['brunnel']),
            string(link.attributes['basetype']), string(link.attributes['bic_inf_tf']),
            int(link.attributes['mit_vol_tf']), string(link.attributes['d_route_tf']),
            string(link.attributes['road_categ']), int(link.attributes['max_sp_tf']),
            int(link.attributes['ad_edge_tf']), string(link.attributes['parking_tf']),
            string(link.attributes['pavement']), int(link.attributes['width_lane']),
            int(link.attributes['grad_tf']),string(link.attributes['rails']),int(link.attributes['n_lanes_tf']),
            string(link.attributes['land_use']), int(link.attributes['restric_tf']));
         restrict_opposite <- int(link.attributes['restric_tf']);
      }
      // Create roads
      create road number: 1 {
         id <- int(link.attributes['id']);
         safetyIndex <- safetyInd;
         weight <- (1 + safetyInd) * 5 - 4; //link weight based on a safety index
         restriction <- restrict;
         shape <- geom;

         // Create an opposite road if a link is bothways
         if safetyInd_opposite != nil {
            create road number: 1 {
               id <- int(link.attributes['id']) * -1; //assign a negative id
               restriction <- restrict_opposite;
               safetyIndex <- safetyInd_opposite;
               weight <- (1 + safetyInd_opposite) * 5 - 4;
               // reverse a link geometry to have an opposite direction
               shape <- polyline(reverse(myself.shape.points));
               if restriction != 2 {
                  myself.oppositeRoad <- self; //assign a road as opposite to an original road
               }
            }
         }
      }
   }
   ask road {
      if restriction = 2 {
         do die;
      }
   } // delete links restricted for cycling and pushing

   // Calculate the network graph depending on the user-defined parameter of the routing algorithm for cyclists
   if routingAlgorithm <= "safest path" {
      cyclingWeights <- road as_map (each::(each.weight * each.shape.perimeter));
   } else { //"shortest path"
      cyclingWeights <- road as_map (each::each.shape.perimeter);
   }
   perimeterWeights <- road as_map (each::each.shape.perimeter);
   theGraph <- directed(as_edge_graph(road) with_weights cyclingWeights);
   remove fileRoads from: self;
}

/* The network assessment model calculates a safety index for every road. The required road attributes are:
 * brunnel - tunnel or bridge: "0"-no, "1"-yes
```

```
 * baseType - type of lane usage:
 *    1-roadway, 2-bicycle path, 4-rail, 5-traffic island, 6-stairway, 7-side walk, 8-parking lane,
 *    11-driving lane, 12-waterway, 13-uphill, 14-right turn lane, 21-protected pedestrian crossing,
 *    22-bicycle crossing, 23-protected pedestrian and bicycle crossing, 24-tunnel, 25-bridge,
 *    31-bike path with adjoining walkway, 32-multipurpose lanes, 33-bicycle lanes, 34-busway,
 *    35-bicycle lane against the one-way, 36-pedestrian and bicycle path
 * bicycleInfrastructure - type of bicycle infrastructure:
 *    "bicycle_way" - physically separated bicycle lane, "bicycle_lane" - bicycle lane adjacent to motorized
 *    lane, "mixed_way" - a lane for bicycles and motorized vehicles, "no" - a motorized lane,
 *    no bicycles allowed
 * mitVolume - daily traffic volume of motorized vehicles per segment (24h)
 * designatedRoute:
 *    "planning"- road segments where planning authorities want bicyclists to ride; usually not available in
 *       standard data sets, must be obtain in workshops etc.
 *    "national" - highest category of designated routes, often along major rivers (in Austria e.g.
 *       Tauernradweg, Donauradweg etc.)
 *    "regional" - designated routes with major, regional impact, often realized as thematic routes (in Salzburg
 *       e.g. Mozartradweg)
 *    "local " - designated routes within municipalities/towns, often sponsored by local businesses (in Salzburg
 *       e.g. Raiffeisenradweg)
 *    "no" - no designated routes or planning intents
 * roadCategory:
 *    "primary" -  highest category of roads which are traversable by bicyclists (highways are excluded!). Mostly
 *        maintained by national/federal authorities and numbered (in Austria with prefix B),
 *    "secondary" - next highest category of roads. Mostly maintained by regional authorities and numbered (in
 *        Austria with prefix L). Within cities major roads which are not maintained by national/federal
 *        authorities should be of this category,
 *    "residential" - municipal roads which don't belong to one of the 2 higher categories,
 *    "service" - all kinds of access and small connector roads where bicycles are permitted (e.g.
 *        Verbindungsweg,Zufahrt, Stichstraße etc.),
 *    "calmed" - roads with any kind of limited MIT access but bicycle permission (Begegnungszone, Wohnstraße,
 *        Anrainerstraßen, Wirtschaftswege etc.),
 *    "no_mit" - any roads with restricted MIT access but bicycle permission (pedestrian zone with bicycle
 *        permission, cycleway etc.),
 *    "path " - paths where cycling is either not permitted or not possible (although it is not explicitly
 *        restricted)
 * maxSpeed - maximum speed allowed by regulations
 * adjacentEdge - number of adjacent edges at the crossings
 * parking - on-street parking: "yes","no"
 * pavement: "asphalt" - paved road, "gravel" - road with compacted gravel, "soft" - uncompacted path with soft
 *    underground, "cobble" - road with cobble stones
 * widthLane - lane width in meters
 * gradient - gradient category according to classification for upslope and downhill:
 *    -1.5 % <"0"<1,5 %; 1,5 % <"1"< 3 %; 3 % <"2"< 6 %; 6 % <"3"< 12 %; "4" > 12 %; -1,5 % >"-1"> -3 %;
 *    -3 % >"-2"> -6 %; -6 % >"-3"> -12 %; "-4" < -12 %
 * rails - railway on the road: "yes","no"
 * numberLane - number of lanes
 * landuse:
 *    "green" - areas that are not sealed and are "green" (open meadows, wood, pastures, parks etc.) or in
 *       "natural" condition (incl. water bodies etc.),
 *    "residential" - areas that are loosely covered with buildings (small towns and villages, single-family
 *        houses etc.),
 *    "built" - areas that are densely covered with buildings without/with little green spaces (cities,
 *        apartment buildings, multi-story buildings etc.),
 *    "commercial" - areas that are mainly covered by large commercial buildings (business parks etc.)
 * oneway - availability of ways in one or both directions. If both directions are "0", then both ways. If
 *    one of directions is "0" and another one is "1", then one way
 * restriction:
 *    "0" - not restricted
 *    "1" - restricted for motorized vehicles, allowed to push bike,
 *    "2" - restricted for every type of mode
 */
float calculateSafetyIndex (int linkId, int brunnel, string baseType, string bicycleInfrastructure,
   int mitVolume, string designatedRoute, string roadCategory, int maxSpeed, int adjacentEdge, string parking,
   string pavement, int widthLane, int gradient, string rails, int numberLane, string landuse, int restriction) {
   list<float> indicators;
   list<float>
   weights <- [bicycleInfrastructureWeight, mitVolumeWeight, designatedRouteWeight, roadCategoryWeight,
      maxSpeedWeight, adjacentEdgeWeight, parkingWeight, pavementWeight, widthLaneWeight, gradientWeight,
      railsWeight, numberLaneWeight, landuseWeight];

   //Calculate inidicators
   switch bicycleInfrastructure {
      match "bicycle_way" { add 0.0 to: indicators; }
      match "mixed_way" { add 0.1 to: indicators; }
      match "bicycle_lane" { add 0.25 to: indicators; }
      match "bus_lane" { add 0.25 to: indicators; }
      match "shared_lane" { add 0.5 to: indicators; }
      match "undefined" { add 0.8 to: indicators; }
      match "no" { add 1.0 to: indicators; }
      default { add -9999.0 to: indicators; }
   }

   if mitVolume >= 0 {
      if mitVolume = 0 {
         add 0.0 to: indicators;
      } else if mitVolume < 500 {
         add 0.05 to: indicators;
      } else if mitVolume < 2500 {
         add 0.25 to: indicators;
      } else if mitVolume < 7500 {
         add 0.5 to: indicators;
      } else if mitVolume < 15000 {
         add 0.75 to: indicators;
      } else if mitVolume < 25000 {
         add 0.85 to: indicators;
      } else if mitVolume >= 25000 {
         add 1 to: indicators;
      }
   } else {
      add -9999.0 to: indicators;
   }
```

```
switch designatedRoute {
   match "planning" { add 0.0 to: indicators; }
   match "national" { add 0.1 to: indicators; }
   match "regional" { add 0.15 to: indicators; }
   match "local" { add 0.2 to: indicators; }
   match "no" { add 1.0 to: indicators; }
   default { add -9999.0 to: indicators; }
}

switch roadCategory {
   match "primary" { add 1.0 to: indicators; }
   match "secondary" { add 0.8 to: indicators; }
   match "residential" { add 0.2 to: indicators; }
   match "service" { add 0.15 to: indicators; }
   match "calmed" { add 0.1 to: indicators; }
   match "no_mit" { add 0.0 to: indicators; }
   match "path" { add 1.0 to: indicators; }
   default { add -9999.0 to: indicators; }
}
if maxSpeed >= 0 {
   if maxSpeed = 0 {
      add 0.0 to: indicators;
   } else if maxSpeed < 30 {
      add 0.1 to: indicators;
   } else if maxSpeed < 50 {
      add 0.15 to: indicators;
   } else if maxSpeed < 60 {
      add 0.4 to: indicators;
   } else if maxSpeed < 70 {
      add 0.6 to: indicators;
   } else if maxSpeed < 80 {
      add 0.7 to: indicators;
   } else if maxSpeed < 100 {
      add 0.8 to: indicators;
   } else if maxSpeed >= 100 {
      add 1.0 to: indicators;
   }
} else {
   add -9999.0 to: indicators;
}
if adjacentEdge >= 0 {
   if adjacentEdge <= 2 {
      add 0.0 to: indicators;
   } else if adjacentEdge = 3 {
      add -9999.0 to: indicators;
   } else if adjacentEdge = 4 {
      add -9999.0 to: indicators;
   } else if adjacentEdge = 5 {
      add -9999.0 to: indicators;
   } else if adjacentEdge = 6 {
      add -9999.0 to: indicators;
   } else if adjacentEdge >= 7 {
      add 1 to: indicators;
   }
} else {
   add -9999.0 to: indicators;
}
switch parking {
   match "yes" { add 1.0 to: indicators; }
   match "no" { add 0.0 to: indicators; }
   default { add -9999.0 to: indicators; }
}
switch pavement {
   match "asphalt" { add 0.0 to: indicators; }
   match "gravel" { add 0.25 to: indicators; }
   match "soft" { add 0.6 to: indicators; }
   match "cobble" { add 1.0 to: indicators; }
   default { add -9999.0 to: indicators; }
}
if widthLane >= 0 {
   if widthLane > 5 {
      add 0.0 to: indicators;
   } else if widthLane > 4 {
      add 0.1 to: indicators;
   } else if widthLane > 3 {
      add 0.15 to: indicators;
   } else if widthLane > 2 {
      add 0.5 to: indicators;
   } else if widthLane <= 2 {
      add 1.0 to: indicators;
   }
} else {
   add -9999.0 to: indicators;
}
switch gradient {
   match 4 { add 1.0 to: indicators; }
   match 3 { add 0.75 to: indicators; }
   match 2 { add 0.6 to: indicators; }
   match 1 { add 0.5 to: indicators; }
   match 0 { add 0.1 to: indicators; }
   match -1 { add 0.0 to: indicators; }
   match -2 { add 0.05 to: indicators; }
   match -3 { add 0.65 to: indicators; }
   match -4 { add 1.0 to: indicators; }
   default { add -9999.0 to: indicators; }
}
switch rails {
   match "yes" { add 1.0 to: indicators; }
   match "no" { add 0.0 to: indicators; }
   default { add -9999.0 to: indicators; }
}
if numberLane >= 0 {
   if numberLane <= 1 {
```

```
            add 0.0 to: indicators;
        } else if numberLane <= 2 {
            add 0.5 to: indicators;
        } else if numberLane <= 3 {
            add 0.8 to: indicators;
        } else if numberLane <= 4 {
            add 0.9 to: indicators;
        } else if numberLane > 4 {
            add 1.0 to: indicators;
        }
    } else {
        add -9999.0 to: indicators;
    }
    switch landuse {
        match "green" { add 0.0 to: indicators; }
        match "residential" { add 0.25 to: indicators; }
        match "built_area" { add 0.8 to: indicators; }
        match "commercial" { add 1.0 to: indicators; }
        default { add -9999.0 to: indicators; }
    }

    // Adjust weights
    if (gradient = 4 or gradient = 3 or gradient = -3 or gradient = -4)
        and (pavement = "gravel" or pavement = "soft" or pavement = "cobble") {
        weights[7] <- pavementAdjusted;
        weights[9] <- gradientAdjusted;
        float sum <- 0.0;
        loop weightIndex from: 0 to: length(weights) - 1 {
            if weightIndex != 7 and weightIndex != 9 {
                sum <- sum + weights[weightIndex];
            }
        }
        loop weightIndex1 from: 0 to: length(weights) - 1 {
            if weightIndex1 != 7 and weightIndex1 != 9 {
                weights[weightIndex1] <- (weights[weightIndex1] / sum) * (1 - weights[7] - weights[9]);
            }
        }
    }
    // Calculate a safety index
    float sum1 <- 0.0;
    loop indicatorIndex from: 0 to: length(indicators) - 1 {
        if indicators[indicatorIndex] != -9999 {
            sum1 <- sum1 + weights[indicatorIndex];
        }
    }
    loop indicatorIndex1 from: 0 to: length(indicators) - 1 {
        if indicators[indicatorIndex1] != -9999 {
            weights[indicatorIndex1] <- weights[indicatorIndex1] / sum1;
        } else {
            weights[indicatorIndex1] <- 0.0;
        }
    }
    float safetyIndex <- 0.0;
    loop indicatorIndex2 from: 0 to: length(indicators) - 1 {
        safetyIndex <- safetyIndex + indicators[indicatorIndex2] * weights[indicatorIndex2];
    }
    // Convert a basetype attribute
    container baseTypeList;
    if baseType != '' {
        baseTypeList <- baseType split_with ";";
        loop baseTypeIndex from: 0 to: length(baseTypeList) - 1 {
            if baseTypeList[baseTypeIndex] = "*" {
                baseTypeList[baseTypeIndex] <- "None";
            } else {
                baseTypeList[baseTypeIndex] <- int(baseTypeList[baseTypeIndex]);
            }
        }
    }
    // Weight a safety index depending on indicators
    if linkId = 901425318 { //Staatsbrücke bridge
        safetyIndex <- bridgeValue + 1;
    } else if baseType != '' and baseTypeList contains 6 { // Stairs
        safetyIndex <- pushValue * 1.5;
    } else if (gradient > 1 or gradient < -1) and restriction = 1 { // Slope with push requirement
        safetyIndex <- pushValue + abs(gradient) / 1.5;
    } else if brunnel = 1 and restriction = 0 { // Bridges
        safetyIndex <- bridgeValue;
    } else if brunnel = 0 and restriction = 1 {
        safetyIndex <- pushValue;
    } else if brunnel = 1 and restriction = 1 { // Bridges with push requirement
        safetyIndex <- bridgeValue + (pushValue / 1.5);
    }
    return safetyIndex with_precision 4;
}

// Create "intersection" species
action createIntersections {
    create intersection from: fileIntersections;
}

// Create "counting stations" species
action createCountingStations {
    create countingStation from: fileCountingStations
        with: [stationName::string(read("stat_name")), stationKey::string(read("key"))] {
        // import the observed counts data to each counting station for visualization
        list firstRow <- countsMatrix row_at 0;
        if (stationKey = nil) {
            stationKey <- stationName;
        }
        // search column index
        int stationColumn <- firstRow index_of stationKey;
        if (stationColumn < 0) {
            // prepend "real_" for compatibility with the data of version 2.0
```

```
            stationKey <- "real_" + stationKey;
            stationColumn <- firstRow index_of stationKey;
        }
        loop hourRow from: 0 to: countsMatrix.rows - 1 {
            add int(countsMatrix[stationColumn, hourRow]) at: int(countsMatrix[0, hourRow]) to: self.observedCounts;
        }
    }
    int roundedCount <- ceil(length(countingStation) / 3) * 3;
    stationChartSize <- {0.3, 0.9 / (roundedCount * 0.3)};
    remove fileCountingStations from: self;
    remove countsMatrix from: self;
}

// Create "person" species
action createPersons {
    float time_createPersons <- machine_time;
    list<int> femaleByAge; // the list of female population by an age group with "5 years" increment
    list<int> maleByAge; // the list of male population by an age group with "5 years" increment
    list<person> createdPersons; // the list of created persons at each residential cell
    int ageMin; // age group minimum limit
    int ageMax; // age group maximum limit
    loop home over: fileHomePlaces { // loop the cells of the residential data
        // Import the number of residents by an age group for every cell
        femaleByAge<-[
            int(home.attributes["f_below_5"]), int(home.attributes["f_5_9"]), int(home.attributes["f_10_14"]),
            int(home.attributes["f_15_19"]), int(home.attributes["f_20_24"]), int(home.attributes["f_25_29"]),
            int(home.attributes["f_30_34"]), int(home.attributes["f_35_39"]), int(home.attributes["f_40_44"]),
            int(home.attributes["f_45_49"]), int(home.attributes["f_50_54"]), int(home.attributes["f_55_59"]),
            int(home.attributes["f_60_64"]), int(home.attributes["f_65_69"]), int(home.attributes["f_70_74"]),
            int(home.attributes["f_75_79"]), int(home.attributes["f_80_84"]), int(home.attributes["f_85_89"])];
        maleByAge<-[
            int(home.attributes["m_below_5"]), int(home.attributes["m_5_9"]), int(home.attributes["m_10_14"]),
            int(home.attributes["m_15_19"]), int(home.attributes["m_20_24"]), int(home.attributes["m_25_29"]),
            int(home.attributes["m_30_34"]), int(home.attributes["m_35_39"]), int(home.attributes["m_40_44"]),
            int(home.attributes["m_45_49"]), int(home.attributes["m_50_54"]), int(home.attributes["m_55_59"]),
            int(home.attributes["m_60_64"]), int(home.attributes["m_65_69"]), int(home.attributes["m_70_74"]),
            int(home.attributes["m_75_79"]), int(home.attributes["m_80_84"]), int(home.attributes["m_85_89"])];
        // Swiss data ends with residents over 90 (f_over_90 and m_over_90)
        if home.attributes["f_over_90"] != nil {
            femaleByAge <- femaleByAge + [int(home.attributes["f_over_90"])];
            maleByAge <- maleByAge + [int(home.attributes["f_over_90"])];
        } else if home.attributes["f_over_100"] != nil {
            femaleByAge <- femaleByAge + [int(home.attributes["f_90_94"]), int(home.attributes["f_95_99"]),
                int(home.attributes["f_over_100"])];
            maleByAge <- maleByAge + [int(home.attributes["m_90_94"]), int(home.attributes["m_95_99"]),
                int(home.attributes["m_over_100"])];
        }
        createdPersons <- [];

        // Create persons within a cell
        ageMin <- 0; // the first age group is 0-4. These variables are updated with an increment of 5 years.
        ageMax <- 4;
        int ageGroupCounter <- 0;
        loop while: ageGroupCounter < length(femaleByAge) {
            // Create female persons
            create person number: femaleByAge[ageGroupCounter] {
                age <- rnd(ageMin, ageMax); // randomly assign an age between min and max age values
                gender <- "female";
                // randomly assign a home location within a cell geometry
                homeLocation <- any_location_in(polygon(home.points));
                add self to: createdPersons;
            }
            // Create male persons
            create person number: maleByAge[ageGroupCounter] {
                age <- rnd(ageMin, ageMax); //randomly assign an age between min and max age values
                gender <- "male";
                // randomly assign a home location within a cell geometry
                homeLocation <- any_location_in(polygon(home.points));
                add self to: createdPersons;
            }
            // Increment an age group
            ageMin <- ageMin + 5;
            ageMax <- ageMax + 5;
            ageGroupCounter <- ageGroupCounter + 1;
            if ((ageGroupCounter + 1) = length(femaleByAge)) {
                ageMax <- max([ageMax, 104]); // Ensure that the last maxAge is at least 104
            }
        }

        // Calculate employment statuses
        do assignEmploymentStatus("below_15",int(home.attributes["m_below_15"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "male" and each.age >= 0 and each.age <= 14)));
        do assignEmploymentStatus("below_15",int(home.attributes["f_below_15"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "female" and each.age >= 0 and each.age <= 14)));
        do assignEmploymentStatus("pensioner",int(home.attributes["m_pension"]),reverse(createdPersons
            where(each.employmentStatus = "" and each.gender= "male") sort_by (each.age)));
        do assignEmploymentStatus("pensioner",int(home.attributes["f_pension"]),reverse(createdPersons
            where(each.employmentStatus = "" and each.gender= "female") sort_by (each.age)));
        do assignEmploymentStatus("pupils_students_over_15",int(home.attributes["m_students"]),createdPersons
            where (each.employmentStatus = "" and each.gender = "male" and each.age >= 15) sort_by(each.age));
        do assignEmploymentStatus("pupils_students_over_15",int(home.attributes["f_students"]),createdPersons
            where (each.employmentStatus = "" and each.gender = "female" and each.age >= 15) sort_by(each.age));
        do assignEmploymentStatus("employed",int(home.attributes["m_employed"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "male" and each.age >= 15)));
        do assignEmploymentStatus("employed",int(home.attributes["f_employed"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "female" and each.age >= 15)));
        do assignEmploymentStatus("unemployed",int(home.attributes["m_unemploy"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "male" and each.age >= 15)));
        do assignEmploymentStatus("unemployed",int(home.attributes["f_unemploy"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "female" and each.age >= 15)));
        do assignEmploymentStatus("inactive",int(home.attributes["m_inactive"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "male" and each.age >= 15)));
        do assignEmploymentStatus("inactive",int(home.attributes["f_inactive"]),shuffle(createdPersons
```

```
            where (each.employmentStatus = "" and each.gender = "female" and each.age >= 15)));
        do assignEmploymentStatus("undefined",int(home.attributes["m_emp_unk"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "male" and each.age >= 15)));
        do assignEmploymentStatus("undefined",int(home.attributes["f_emp_unk"]),shuffle(createdPersons
            where (each.employmentStatus = "" and each.gender = "female" and each.age >= 15)));

        // Remove persons under 6 years old from the simulation, because of their inability to carry out activities
        // and travel on their own. Remove persons whose employment status is unknown.
        ask createdPersons where (each.age < 6 or each.employmentStatus = "undefined") {
            remove self from: createdPersons;
            do die;
        }
        // Update employment statuses by assigning education statuses (pupil, student).
        // be Pupils and students over 15 can be registered as "employed", thats why some of "employed" persons
        // will reassigned with "pupil"/"student" statuses, persons "below_15" will be reassigned as "pupils".
        do assignEmploymentStatus("pupil",int(home.attributes["pupils"]),reverse(
            createdPersons where (each.employmentStatus = "below_15") sort_by (each.age))+
            createdPersons where (each.employmentStatus = "pupils_students_over_15") sort_by (each.age) +
            createdPersons where (each.employmentStatus = "employed") sort_by (each.age));
        do assignEmploymentStatus("student",int(home.attributes["students"]),
            createdPersons where (each.employmentStatus = "pupils_students_over_15" or
                each.employmentStatus = "employed"
            ) sort_by (each.age));
    }

    // Remove persons from the simulation, whose education statuses were not defined due to unknown values
    ask person where (each.employmentStatus = "below_15" or each.employmentStatus = "pupils_students_over_15") {
        do die;
    }
    ask person where (each.employmentStatus = "") {
        do die;
    }
    remove fileHomePlaces from: self;
    write "Persons are created: " + person count (true) + ". Execution time: "
        + (machine_time - time_createPersons) / 1000 + " sec";
}

// Assign an employment status. "emStName" - employment status type, "numberEmpSt" - required number of people by
// employment status, "the_persons" - suitable persons
action assignEmploymentStatus (string empStName, int numberEmpSt, list<person> the_persons) {
    person rndPerson;
    loop while: numberEmpSt != 0 {
        rndPerson <- first(the_persons);
        rndPerson.employmentStatus <- empStName;
        the_persons <- the_persons - rndPerson;
        numberEmpSt <- numberEmpSt - 1;
    }
}

// Prepare probability distributions of activity types, departure times and modes
action prepareProbabilities {
// Calculate the activity type probabilities depending on employment status and activity position
// The matrix columns (1-12) represent activity types. The rows (0-7) represent activity positions
    int persons_total <- length(person); //total number of the population
    // loop through employment statuses
    loop emp_status over: ["employed", "unemployed", "inactive", "pensioner", "pupil", "student"] {
        map<int, map<string, float>> probabilitiesById; // list of probabilities by activity position
        loop actPosRow from: 0 to: activityMatrix.rows - 1 { // loop through the rows of activity positions
            map<string, float> probabilities; // list of probabilities by activity type
            // Import probabilities for the total population from the file
            loop actTypeColumn from: 0 to: activityMatrix.columns-1{ // loop through the columns of activity type
                switch actTypeColumn{ // add probability values to the "probabilities" list
                    match 0{add float(activityMatrix[actTypeColumn,actPosRow]) at: "home" to: probabilities;}
                    match 1{add float(activityMatrix[actTypeColumn,actPosRow]) at: "other_place" to: probabilities;}
                    match 2{add float(activityMatrix[actTypeColumn,actPosRow]) at: "work" to: probabilities;}
                    match 3{add float(activityMatrix[actTypeColumn,actPosRow]) at: "business" to: probabilities;}
                    match 4{
                        add float(activityMatrix[actTypeColumn,actPosRow]) at: "school" to: probabilities;
                        add float(activityMatrix[actTypeColumn,actPosRow]) at: "university" to: probabilities;
                    }
                    match 5{add float(activityMatrix[actTypeColumn,actPosRow]) at: "shop" to: probabilities;}
                    match 6{add float(activityMatrix[actTypeColumn,actPosRow]) at: "authority" to: probabilities;}
                    match 7{add float(activityMatrix[actTypeColumn,actPosRow]) at: "doctor" to: probabilities;}
                    match 8{add float(activityMatrix[actTypeColumn,actPosRow]) at: "recreation" to: probabilities;}
                    match 9{add float(activityMatrix[actTypeColumn,actPosRow]) at: "bringing" to: probabilities;}
                    match 10{add float(activityMatrix[actTypeColumn,actPosRow]) at: "none" to: probabilities;}
                }
            }
            // Recalculate probabilities depending on employment status
            float unessential_probabilities <- 100.0 - probabilities["work"] - probabilities["business"]
                - probabilities["school"]; // summ of the probabilities of non-utalititarian activities
            switch emp_status {
                match "employed" {
                    float new_work_probability <- persons_total * probabilities["work"] / length(
                        person where (each.employmentStatus = emp_status)
                    ); // new "work" probability for the amount of employed persons
                    float new_business_probability <- persons_total * probabilities["business"] / length(
                        person where (each.employmentStatus = emp_status)
                    ); // new "business" probability for the amount of employed persons
                    // proportionalize the "work" and "business" probabilities to be under 100 in total
                    if new_work_probability + new_business_probability > 100.0 {
                        new_work_probability <- new_work_probability * 100.0 / (new_work_probability
                            + new_business_probability);
                        new_business_probability <- 100.0 - new_work_probability;
                    }
                    loop activity over: probabilities.keys {
                        switch activity {
                            match "work" { probabilities[activity] <- new_work_probability; }
                            match "business" { probabilities[activity] <- new_business_probability; }
                            match "school" { probabilities[activity] <- 0.0; }
                            match "university" { probabilities[activity] <- 0.0; }
                            default {
                                probabilities[activity] <- (100.0 - new_work_probability - new_business_probability)
```

```
                        * probabilities[activity] / unessential_probabilities;
                }
            }
        }
        match "pupil" {
            // new "school" probability for the amount of pupils and students
            float new_school_probability <- persons_total * probabilities["school"] / length(
                person where (each.employmentStatus = "pupil" or each.employmentStatus = "student"));
            loop activity over: probabilities.keys {
                switch activity {
                    match "work" { probabilities[activity] <- 0.0; }
                    match "business" { probabilities[activity] <- 0.0; }
                    match "school" { probabilities[activity] <- new_school_probability; }
                    match "university" { probabilities[activity] <- 0.0; }
                    default {
                        probabilities[activity] <- (100.0 - new_school_probability) * probabilities[activity]
                            / unessential_probabilities;
                    }
                }
            }
        }
        match "student" {
            // new "university" probability for the amount of pupils and students
            float new_university_probability <- persons_total * probabilities["university"] / length(
                person where (each.employmentStatus = "pupil" or each.employmentStatus = "student"));
            loop activity over: probabilities.keys {
                switch activity {
                    match "work" { probabilities[activity] <- 0.0; }
                    match "business" { probabilities[activity] <- 0.0; }
                    match "school" { probabilities[activity] <- 0.0; }
                    match "university" { probabilities[activity] <- new_university_probability; }
                    default {
                        probabilities[activity] <- (100.0 - new_university_probability) * probabilities[activity]
                            / unessential_probabilities;
                    }
                }
            }
        }
        default {
            loop activity over: probabilities.keys {
                switch activity {
                    match "work" { probabilities[activity] <- 0.0; }
                    match "business" { probabilities[activity] <- 0.0; }
                    match "school" { probabilities[activity] <- 0.0; }
                    match "university" { probabilities[activity] <- 0.0; }
                    default {
                        probabilities[activity] <- 100.0 * probabilities[activity] / unessential_probabilities;
                    }
                }
            }
        }
    }
    // add the "probabilities" to the "probabilitiesById" list
    add probabilities at: actPosRow to: probabilitiesById;
  }
  // add the "probabilitiesById" to the final "activityProbabilities" list
  add probabilitiesById at: emp_status to: activityProbabilities;
}
remove activityMatrix from: self;

// Calculate accumulative departure time probabilities (0-100%). The matrix columns (2-26) represent 24 hours.
// The rows (1-64) represent activity type and position in activity chain.
int activityCounter <- 0; // variable for looping through activity types
loop actTypeRow from: 0 to: 8 { // loop through rows of activity types
  map<int, map<int, float>> probabilitiesByPosition; // list of probabilities by activity position
  map<int, float> thresholdsByPosition; // list of threshold probabilities by activity position
  // loop through rows of activity positions
  loop actPositionRow from: 0 + activityCounter to: 6 + activityCounter {
    float newTimeProbability; // accumulative probability from 0 to 100%
    map<int, float> probabilities; // list of prababilities by hour
    loop hourColumn from: 2 to: timeMatrix.columns - 1 { // loop through hours
      if float(timeMatrix[hourColumn, actPositionRow]) > 0.0 {
        newTimeProbability <- newTimeProbability + float(timeMatrix[hourColumn, actPositionRow]);
        // add the probability value to the "probabilities" list
        add newTimeProbability at: hourColumn - 1 to: probabilities;
      }
    }
    // add the "probabilities" to the "probabilitiesByPosition" list
    add probabilities at: actPositionRow - activityCounter + 1 to: probabilitiesByPosition;
    // set 0.0 for all thresholds. They will be updated during the simulation
    add 0.0 at: actPositionRow - activityCounter to: thresholdsByPosition;
  }
  // add the "probabilitiesByPosition" to the final "timeProbabilities" list and the
  // "thresholdsByPosition" to the final "timeThresholds" list
  switch actTypeRow {
    match 0 {
      add probabilitiesByPosition at: "home" to: timeProbabilities;
      add thresholdsByPosition at: "home" to: timeThresholds;
    }
    match 1 {
      add probabilitiesByPosition at: "other_place" to: timeProbabilities;
      add thresholdsByPosition at: "other_place" to: timeThresholds;
    }
    match 2 {
      add probabilitiesByPosition at: "work" to: timeProbabilities;
      add thresholdsByPosition at: "work" to: timeThresholds;
    }
    match 3 {
      add probabilitiesByPosition at: "business" to: timeProbabilities;
      add thresholdsByPosition at: "business" to: timeThresholds;
    }
    match 4 {
```

```
            add probabilitiesByPosition at: "shop" to: timeProbabilities;
            add thresholdsByPosition at: "shop" to: timeThresholds;
        }
        match 5 {
            add probabilitiesByPosition at: "authority" to: timeProbabilities;
            add thresholdsByPosition at: "authority" to: timeThresholds;
        }
        match 6 {
            add probabilitiesByPosition at: "doctor" to: timeProbabilities;
            add thresholdsByPosition at: "doctor" to: timeThresholds;
        }
        match 7 {
            add probabilitiesByPosition at: "recreation" to: timeProbabilities;
            add thresholdsByPosition at: "recreation" to: timeThresholds;
        }
        match 8 {
            add probabilitiesByPosition at: "bringing" to: timeProbabilities;
            add thresholdsByPosition at: "bringing" to: timeThresholds;
        }
    }
    activityCounter <- activityCounter + 7; //update the activityCounter
}
remove timeMatrix from: self;

// Calculate mode type probabilities. The matrix columns (2-7) represent mode. The rows (0-21) represent
// activity type and extent (city/region).
activityCounter <- 0; //variable for looping through activity types
loop actTypeRow from: 0 to: 10 { //loop through the rows of activity types
    map<bool, map<string, float>> probabilitiesByExtend; //list of probabilities by spatial extent
    // loop through through the rows of spatial extents
    loop extendRow from: 0 + activityCounter to: 1 + activityCounter {
        float newModeProbability; // accumulative probability from 0 to 100%
        map<string, float> probabilities; // list of probabilities by mode
        loop modeColumn from: 2 to: modeMatrix.columns - 1 { // loop through the columns of modes
            if float(modeMatrix[modeColumn, extendRow]) > 0.0 {
                newModeProbability <- newModeProbability + float(modeMatrix[modeColumn, extendRow]);
                switch modeColumn { // add probability values to the "probabilities" list
                    match 2 {
                        add newModeProbability at: "walk" to: probabilities;
                    }
                    match 3 {
                        add newModeProbability at: "bike" to: probabilities;
                    }
                    match 4 {
                        add newModeProbability at: "car" to: probabilities;
                    }
                    match 5 {
                        add newModeProbability at: "car_passenger" to: probabilities;
                    }
                    match 6 {
                        add newModeProbability at: "public_transport" to: probabilities;
                    }
                    match 7 {
                        add newModeProbability at: "other_mode" to: probabilities;
                    }
                }
            }
        }
        if last(probabilities) < 100.0 {
            probabilities[probabilities index_of last(probabilities)] <- 100.0;
        } // set a sharp (100.0) limit for the last probability in the list
        switch extendRow - activityCounter { //add the "probabilities" to the "probabilitiesByExtend" list
            match 0 {
                add probabilities at: true to: probabilitiesByExtend;
            }
            match 1 {
                add probabilities at: false to: probabilitiesByExtend;
            }
        }
    }
    switch actTypeRow { // add the "probabilitiesByExtend" to the final "modeProbabilities" list
        match 0 {
            add probabilitiesByExtend at: "home" to: modeProbabilities;
        }
        match 1 {
            add probabilitiesByExtend at: "other_place" to: modeProbabilities;
        }
        match 2 {
            add probabilitiesByExtend at: "work" to: modeProbabilities;
        }
        match 3 {
            add probabilitiesByExtend at: "business" to: modeProbabilities;
        }
        match 4 {
            add probabilitiesByExtend at: "school" to: modeProbabilities;
        }
        match 5 {
            add probabilitiesByExtend at: "university" to: modeProbabilities;
        }
        match 6 {
            add probabilitiesByExtend at: "shop" to: modeProbabilities;
        }
        match 7 {
            add probabilitiesByExtend at: "authority" to: modeProbabilities;
        }
        match 8 {
            add probabilitiesByExtend at: "doctor" to: modeProbabilities;
        }
        match 9 {
            add probabilitiesByExtend at: "recreation" to: modeProbabilities;
        }
        match 10 {
            add probabilitiesByExtend at: "bringing" to: modeProbabilities;
```

```
            }
        }
        activityCounter <- activityCounter + 2; //update the activityCounter variable
    }
    remove modeMatrix from: self;

    // Calculate accumulative probabilities of duration of work activity. The matrix columns (2,3) represent
    // gender. The columns (1,2) represent min/max working hours. The rows represent probabilities by gender
    // and working hours.
    loop theGenderCol from: 2 to: 3 { // loop through the columns of gender
        float newDurationProbability; // accumulative probability from 0 to 100%
        map<map<string, int>, float> probabilities; // list of probabilities by working hours
        loop theRows from: 0 to: durationMatrix.rows - 1 { // loop through the rows of working hours
            // min/max working hours in minutes per day (initial values are in hours per week)
            map<string, int> workingHours;
            add int(durationMatrix[0, theRows]) / 5 * 60 at: "min" to: workingHours;
            add int(durationMatrix[1, theRows]) / 5 * 60 at: "max" to: workingHours;
             // accumulative probability from 0 to 100%
            newDurationProbability <- newDurationProbability + float(durationMatrix[theGenderCol, theRows]);
            // add probability values to the "probabilities" list
            add newDurationProbability at: workingHours to: probabilities;
        }

        if last(probabilities) < 100.0 {
            probabilities[probabilities index_of last(probabilities)] <- 100.0;
        } // set a sharp (100.0) limit for the last probability in the list
        switch theGenderCol { // add the "probabilities" to the final "workDurationProbabilities" list
            match 2 {
                add probabilities at: "male" to: workDurationProbabilities;
            }
            match 3 {
                add probabilities at: "female" to: workDurationProbabilities;
            }
        }
    }
    remove durationMatrix from: self;
}

// Calculate initial travel schedules
action calculateFirstActivity {
    ask person {
        // Locate a person at an initial activity
        activityId <- 0;
        activityType <- calculateActivityType();
        location <- calculateTarget();

        // Calculate the next activity
        activityId <- activityId + 1;
        activityType <- calculateActivityType();
        startingTime <- lastActivity = false ? calculateStartingTime() : 0;
        durationTime <- lastActivity = false ? calculateDurationTime() : 0;
        endingTime <- 9999; // cannot use null value abbreviation "nil", because it equals 0 for int
        mode <- calculateMode();
        sourceLocation <- location;
        targetLocation <- lastActivity = false ? calculateTarget() : nil;
        if lastActivity = true {
            if location = homeLocation {
                do die;
            } else {
                activityType <- "home";
                startingTime <- cycle;
                targetLocation <- homeLocation;
            }
        }
    }
}

// The simulation starts
reflex startSimulation when: cycle = 0 {
    simulationStartTime <- machine_time;
}

// Update starting time probabilities with time thresholds. Persons can depart anytime from next hour
reflex updateTimeThresholds when: every(60 #cycle) {
    loop activityT over: timeProbabilities.keys {
        loop activityP over: timeProbabilities[activityT].keys {
            if timeProbabilities[activityT][activityP][int(cycle / 60) + 1] != nil {
                timeThresholds[activityT][activityP] <- timeProbabilities[activityT][activityP][int(cycle / 60) + 1];
            }
            timeProbabilities[activityT][activityP] >- timeProbabilities[activityT][activityP][int(cycle / 60) + 1];
        }
    }
}

// Save moving cyclists by activity type every "activeCylistsTimeInterval" to the "active_cyclists" file
reflex saveActiveCyclists when: every(activeCylistsTimeInterval #cycle) {
    save [cycle, person count (each.status = "moving"), person count (each.status = "moving"
        and each.activityType = "home"), person count (each.status = "moving" and each.activityType = "work"),
        person count (each.status = "moving" and each.activityType = "business"),
        person count (each.status = "moving" and each.activityType = "school"),
        person count (each.status = "moving" and each.activityType = "university"),
        person count (each.status = "moving" and each.activityType = "authority"),
        person count (each.status = "moving" and each.activityType = "doctor"),
        person count (each.status = "moving" and each.activityType = "recreation"),
        person count (each.status = "moving" and each.activityType = "shop"),
        person count (each.status = "moving" and each.activityType = "bringing"),
        person count (each.status = "moving" and each.activityType = "other_place")]
        to: activeCyclistsFileName format: csv rewrite: false;
}

//Save the heatmap with registered cyclists on the network by hour
reflex saveHeatmap when:cycle = 1441{
    save road where(each.id >= 0) to: heatmapFileName format:"shp" crs:crs rewrite:true attributes:[
```

```
            "id"::id,
            "safetyIndex"::safetyIndex,
            "cyclistsByIntervalInMin"::cyclistsByInterval,
            "cyclists"::cyclistsTotal
        ];
    }

    //The simulation stops at 24:00
    reflex stopSimulation when: cycle = 1441 {
        write "Total simulation time: " + (machine_time - simulationStartTime) / 1000 / 60 + " min";
        if repeatedFlag {
            do die;
        } else {
            do pause;
        }

    }
}

////////////////////////////////////////////////SPECIES DECLARATION//////////////////////////////////////////////

/* Person species represent travelling objects that carry out activities
 * 1) The activity assignment is based on probabilities and assumptions about activity type, departure and duration
 *    times, mode, speed, min/max travelling distance restrictions and target location
 * 2) Persons on bicycles that traverse city outline start to move at starting times. Other persons are teleported
 *    to target location at starting times.
 * 3) A movement ends at target location
 * 4) An activity continues until a duration time is over
 * 5) A next activity assignment occurs at an ending time of current activity in an iterative manner until the last
 *    activity */
species person skills: [moving] {
    int age; // 0-104
    string gender; // "male" and "female"
    // "below_15", "pupil", "employed_pupil", "student", "employed_student","employed", "unemployed", "pensioner",
    // "inactive"(economically inactive, not registered as unemployed),"undefined"
    string employmentStatus <- "";
    point homeLocation; // home location
    string status <- "staying"; // "staying" or "moving"
    int activityId; //every activity gets the ordinal number in a sequence (chain) of performed activities
    // "home","work","business","school","university","shop","authority","doctor","recreation","bringing"(bringing
    // children to kindergarden or children and old persons to hospital), "other_place"(any place else)
    string activityType;
    bool lastActivity <- false; // "true" if the activity is the last one in the simulation
    int startingTime; // time to start travelling, in min
    int endingTime; // time to stop the activity, in min
    int durationTime; // time to spend at the activity location, in min
    string mode <- ""; // transport mode: "walk","bike","car","car_passenger","public_transport","other_mode"
    float minDistance <- 0.00; // minimum allowed distance to travel with the mode, in meters
    // maximum allowed distance to travel with the mode, in meters. Default value is 1000000.00 for "car",
    // "car_passenger", "public_transport", "other_mode"
    float maxDistance <- 1000000.00;
    point sourceLocation; // departure location
    point targetLocation; // target location
    aspect base {
        draw triangle(20) depth: rnd(2) color: #white; //person shape
    }

    //Calculate a next activity
    reflex calculateActivity when: cycle = endingTime {
        endingTime <- 9999;
        activityId <- activityId + 1;
        activityType <- calculateActivityType();
        startingTime <- lastActivity = false ? calculateStartingTime() : 0;
        durationTime <- lastActivity = false ? calculateDurationTime() : 0;
        mode <- location = homeLocation ? calculateMode() : mode; //mode change is possible only at home
        sourceLocation <- location;
        targetLocation <- lastActivity = false ? calculateTarget() : nil;
        // if a selected activity is the last one, a person travels home or leaves the simulation world
        if lastActivity = true {
            if location = homeLocation {
                do die;
            } else {
                activityType <- "home";
                startingTime <- cycle;
                targetLocation <- homeLocation;
            }
        }
    }

    // Calculate an activity type
    string calculateActivityType {
    // Calculate a probability distribution from 0.0% to 100.0%
        map<string, float> myActivityProbabilities <- copy(activityProbabilities[employmentStatus][activityId]);
        // a probability of a recent activity is excluded from probabilities
        myActivityProbabilities[activityType] <- 0.0;
        float newActivityProbability;
        float probabilitiesSum <- sum(myActivityProbabilities);
        loop index over: myActivityProbabilities.keys {
            newActivityProbability <- newActivityProbability + myActivityProbabilities[index] * 100 / probabilitiesSum;
            myActivityProbabilities[index] <- newActivityProbability;
        }

        // Calculate an activity type according to a probability distribution
        float rndNumberActType <- rnd(last(myActivityProbabilities));
        switch rndNumberActType {
            match_between [0.0, myActivityProbabilities["home"]] {
                return "home";
                break;
            }
            match_between [myActivityProbabilities["home"], myActivityProbabilities["other_place"]] {
                return "other_place";
                break;
            }
```

```
        match_between [myActivityProbabilities["other_place"], myActivityProbabilities["work"]] {
            return "work";
            break;
        }
        match_between [myActivityProbabilities["work"], myActivityProbabilities["business"]] {
            return "business";
            break;
        }
        match_between [myActivityProbabilities["business"], myActivityProbabilities["school"]] {
            return "school";
            break;
        }
        match_between [myActivityProbabilities["school"], myActivityProbabilities["university"]] {
            return "university";
            break;
        }
        match_between [myActivityProbabilities["university"], myActivityProbabilities["shop"]] {
            return "shop";
            break;
        }
        match_between [myActivityProbabilities["shop"], myActivityProbabilities["authority"]] {
            return "authority";
            break;
        }
        match_between [myActivityProbabilities["authority"], myActivityProbabilities["doctor"]] {
            return "doctor";
            break;
        }
        match_between [myActivityProbabilities["doctor"], myActivityProbabilities["recreation"]] {
            return "recreation";
            break;
        }
        match_between [myActivityProbabilities["recreation"], myActivityProbabilities["bringing"]] {
            return "bringing";
            break;
        }
        // if an activity type is null, then a person leaves the simulation or finishes its daily schedule
        // by going home first.
        default {
            lastActivity <- true;
            return "home";
            break;
        }
    }
}

// Calculate a starting time
int calculateStartingTime {
    int earliestTime;
    int latestTime;
    switch activityType {
        // departure times for "school" and "university" activities are randomly selected between min and max times
        match "school" {
            earliestTime <- 420;
            latestTime <- 480;
            if cycle <= earliestTime {
                return rnd(earliestTime, latestTime);
            } else if cycle <= latestTime {
                return rnd(cycle, latestTime);
            } else {
                lastActivity <- true;
                return 9999;
            }
        }
        match "university" {
            earliestTime <- 480;
            latestTime <- 1080;
            if cycle <= earliestTime {
                return rnd(earliestTime, latestTime);
            } else if cycle <= latestTime {
                return rnd(cycle, latestTime);
            } else {
                lastActivity <- true;
                return 9999;
            }
        }
        default {
            // select distribution of probabilities depending on activityType and activityId
            map<int, float> startTimeProbability <- timeProbabilities[activityType][activityId];
            if length(startTimeProbability) = 0 {
                lastActivity <- true;
                return 9999;
            } // finish daily travels and go home
            float timeThreshold <- timeThresholds[activityType][activityId];
            float rndNumberStartTime <- rnd(timeThreshold, max(startTimeProbability));
            loop k over: startTimeProbability.keys {
                if rndNumberStartTime >= timeThreshold and rndNumberStartTime <= startTimeProbability[k] {
                    earliestTime <- 60 * (k - 1);
                    latestTime <- 60 * k;
                    break;
                } else {
                    timeThreshold <- startTimeProbability[k];
                }
            }
            return rnd(earliestTime, latestTime);
        }
    }
}

// Calculate a duration time depending on the type of the next activity
int calculateDurationTime {
    int shortestDuration; // min duration
    int longestDuration; // max duration
```

```
    int leftTime; // time left before a facility closes

    // Select min/max durations
    switch activityType {
        match "home" {
            shortestDuration <- 30;
            longestDuration <- 60;
            leftTime <- 1380;
        }
        // A duration for "pupil" depends on an age
        match "school" {
            switch age {
                match_between [6, 7] {
                    shortestDuration <- 300;
                    longestDuration <- 300;
                    leftTime <- 960;
                }
                match_between [8, 9] {
                    shortestDuration <- 420;
                    longestDuration <- 420;
                    leftTime <- 960;
                }
                match_between [10, 13] {
                    shortestDuration <- 480;
                    longestDuration <- 480;
                    leftTime <- 960;
                }
                default {
                    shortestDuration <- 360;
                    longestDuration <- 480;
                    leftTime <- 960;
                }
            }
        }
        match "university" {
            shortestDuration <- 120;
            longestDuration <- 360;
            leftTime <- 1200;
        }
        // A duration for "employed" depends on a gender and a probability distribution of working hours
        match "work" {
            map<map<string, int>, float> workDurationProbability <- workDurationProbabilities[gender];
            float durationThreshold <- 0.0;
            float rndNumberWorkDuration <- rnd(100.0);
            loop k over: workDurationProbability.keys {
                if rndNumberWorkDuration >= durationThreshold and rndNumberWorkDuration <=workDurationProbability[k]{
                    shortestDuration <- k["min"];
                    longestDuration <- k["max"];
                    return rnd(shortestDuration, longestDuration);
                } else {
                    durationThreshold <- workDurationProbability[k];
                }
            }
        }
        match "business" {
            shortestDuration <- 15;
            longestDuration <- 180;
            leftTime <- 1080;
        }
        match "recreation" {
            shortestDuration <- 60;
            longestDuration <- 180;
            leftTime <- 1380;
        }
        match "shop" {
            shortestDuration <- 15;
            longestDuration <- 120;
            leftTime <- 1080;
        }
        match "other_place" {
            shortestDuration <- 15;
            longestDuration <- 180;
            leftTime <- 1380;
        }
        match "authority" {
            shortestDuration <- 30;
            longestDuration <- 30;
            leftTime <- 1080;
        }
        match "doctor" {
            shortestDuration <- 60;
            longestDuration <- 60;
            leftTime <- 1080;
        }
        match "bringing" {
            shortestDuration <- 15;
            longestDuration <- 60;
            leftTime <- 1080;
        }
    }
    // Calculate a duration
    leftTime <- leftTime - startingTime; // left time until a facility is closed
    if leftTime >= longestDuration {
        return rnd(shortestDuration, longestDuration);
    } else if leftTime >= shortestDuration {
        return rnd(shortestDuration, leftTime);
    } else {
        lastActivity <- true;
        return 9999;
    }
}

// Calculate a transport mode depending on a type of the next activity and location
```

```
string calculateMode {
   map<string, float> modeProbability <- modeProbabilities[activityType][location overlaps cityOutline];
   float modeThreshold <- 0.0;
   float rndNumberMode <- rnd(100.00);
   loop k over: modeProbability.keys {
      if rndNumberMode >= modeThreshold and rndNumberMode <= modeProbability[k] {
         switch k {
            match "walk" {
            // Calculate minimum and maximum travel distances by mode based on distance probability distribution
            // for "bike" and "walk"
               switch rnd(100.00) {
                  match_between [0.0, 70.9908735332464] {
                     minDistance <- 0.00;
                     maxDistance <- 1000.00;
                     break;
                  }
                  match_between [70.9908735332464, 100.00] {
                     minDistance <- 1000.00;
                     maxDistance <- 5000.00;
                     break;
                  }
               }
               //Calculate a speed
               speed <- rnd(0.7, 2.0); //0.7-2.0 m/s
            }
            match "bike" {
               switch rnd(100.00) {
                  match_between [0.0, 72.9672650475185] {
                     minDistance <- 0.00;
                     maxDistance <- 2000.00;
                     break;
                  }
                  match_between [72.9672650475185, 100.00] {
                     minDistance <- 2000.00;
                     maxDistance <- 8000.00;
                     break;
                  }
               }
               speed <- rnd(1.6, 5.5); //1.6-5.5 m/s
            }
            match "other_mode" {
               minDistance <- 0.00;
               maxDistance <- 1000000.0;
               speed <- rnd(2.4, 13.6);
            }
            default {
               minDistance <- 0.00;
               maxDistance <- 1000000.0;
               speed <- rnd(4.9, 14.9);
            }
         }
         return k;
      } else {
         modeThreshold <- modeProbability[k];
      }
   }
}

// Calculate a target location depending on activity type and allowed travel distances (min/max)
point calculateTarget {
   facility rndFacility;
   switch activityType {
      match "home" {
         return homeLocation;
      }
      match "work" {
         // Select a work facility where the number of employees (facilityPopulation) is greater than 0
         rndFacility <- shuffle(facility) first_with (each.facilityType = "work" and each.facilityPopulation > 0
             and between(distance_to(self, each.location), minDistance, maxDistance));
         // Select a work facility without distance restrictions, if there is no facility within allowed
         // distances
         if rndFacility = nil {
            rndFacility <-shuffle(facility) first_with (each.facilityType = "work"
               and each.facilityPopulation > 0);
         }
         // decrease number of potential work places
         rndFacility.facilityPopulation <- rndFacility.facilityPopulation - 1;
         return any_location_in(rndFacility);
      }
      match "bringing" {
         rndFacility <- shuffle(facility) first_with ((each.facilityType = "kindergarten"
            or each.facilityType = "doctor")
               and between(distance_to(self, each.location), minDistance, maxDistance));
         if rndFacility = nil {
            rndFacility <- shuffle(facility) first_with (each.facilityType = "kindergarten"
               or each.facilityType = "doctor");
         }
         return any_location_in(rndFacility);
      }
      default {
         rndFacility <- shuffle(facility) first_with (each.facilityType = activityType
            and between(distance_to(self, each.location), minDistance, maxDistance));
         if rndFacility = nil {
            rndFacility <- shuffle(facility) first_with (each.facilityType = activityType);
         }
         return any_location_in(rndFacility);
      }
   }
}

// Decide to move or teleport
reflex start when: cycle = startingTime {
   if mode = "bike" { //if a mode is "bike"
```

```
            path trip <- path_between(theGraph, sourceLocation, targetLocation);
            if (trip != nil) {
               // if the start and end of a trip are not on the opposite roads
               if abs((first(trip.edges) as road).id) != abs((last(trip.edges) as road).id)
                  and first(trip.segments) != last(trip.segments) {
                  // if a trip or the parts of a trip occur in the city
                  if trip.shape overlaps cityOutline {
                     status <- "moving"; //move along the network
                  }
               }
            } else {
               do saveUnaccessibleLocations;
            }
         }
         // if trip is not by bike or does not intersect the city or is very short and uses two links
         if status = "staying" {
            location <- targetLocation; //don't move along the network but transfer to a target location
            // update a duration time
            durationTime <- durationTime + int((distance_to(sourceLocation, targetLocation) / speed) / 60.0);
         }
      }

      // Move
      reflex move when: status = "moving" {
         path pathFollowed <- goto(on: theGraph, target: targetLocation, move_weights: perimeterWeights,
            return_path: true);
         if pathFollowed != nil {
            // Register a person at traversed completely roads
            loop segment over: pathFollowed.segments {
               road traversedRoad <- road(pathFollowed agent_from_geometry segment);
               if point(theGraph target_of (traversedRoad)) overlaps segment {
                  traversedRoad.cyclistsRoad <- traversedRoad.cyclistsRoad + 1;
               }
            }
            // Register a person at traversed counting station
            ask countingStation overlapping (pathFollowed.shape) {
               cyclistsStation <- cyclistsStation + 1;
            }

         }
      }

      // Stop at a target location
      reflex stop when: location = targetLocation {
         do saveTrip;
         status <- "staying";
         startingTime <- 9999;
         targetLocation <- nil;
         sourceLocation <- nil;
         if activityId = 7 or lastActivity = true or endingTime >= 1440 {
            do die;
         } // remove a person from simulation
      }

      // Save trip information to "trips" file
      action saveTrip {
         geometry tripGeom; // trip geometry
         float tripLength; // travelled distance, in meters
         float tripCityShare; // percentage of trip within city, in %
         int tripTravelTime; // travel time, in min
         int intersections; // number of intersections along a trip
         if status = "moving" { // if a trip is made by "bike" and intersect the city
            path trip <- path_between(theGraph, sourceLocation, targetLocation);
            try {
               tripGeom <- trip.shape;
               tripLength <- tripGeom.perimeter;
               tripCityShare <- (tripGeom inter cityOutline).perimeter * 100 / tripLength;
               tripTravelTime <- cycle - startingTime;
               intersections <- length(intersection overlapping (tripGeom));
               tripGeom <- CRS_transform(tripGeom);
            } catch {
               do saveUnaccessibleLocations;
            }
         }
         if lastActivity = true {
            durationTime <- 1440 - cycle;
         } // update a duration time
         endingTime <- cycle + durationTime;
         // save traip information as txt with ";" as a delimeter, since gama turn ; to , when saving as csv file
         save string(self) + ";" + gender + ";" + string(age) + ";" + string(activityId) + ";" + activityType
            + ";" + string(startingTime) + ";" + string(endingTime) + ";" + string(durationTime) + ";"
            + string(tripTravelTime) + ";" + mode + ";" + string(speed) + ";" + string(tripLength) + ";"
            + string(tripCityShare) + ";" + string(intersections) + ";" + string(tripGeom) + ";"
            to: tripsFileName format: "text" rewrite: false;
      }
      // Save the unaccessible locations
      action saveUnaccessibleLocations {
         point source <- CRS_transform(sourceLocation) as point;
         point target <- CRS_transform(targetLocation) as point;
         int uTime <- machine_time as int;
         if (!file_exists(unaccessibleFileName)) {
            save 'cycle' + ";" + 'self' + ";" + 'machine time' + ";" + 'starting time' + ";" + 'type' + ";" + 'point'
               to: unaccessibleFileName format: text rewrite: true;
         }

         save string(cycle) + ";" + string(self) + ";" + uTime + ";" + string(startingTime) + ";" + 'target' + ";"
            + string(target) + ";" to: unaccessibleFileName format: "text" rewrite: false;
         save string(cycle) + ";" + string(self) + ";" + uTime + ";" + string(startingTime) + ";" + 'source' + ";"
            + string(source) + ";" to: unaccessibleFileName format: "text" rewrite: false;
      }
   }

   // Facility species represent the locations by activity type
```

```
species facility schedules: [] {
   string facilityType;
   int facilityPopulation; // number of employees at "work" facilities
   aspect base {
      draw shape color: #cadetblue;
   }
}

// Road species represent directional and connected links that form street network.
species road frequency: networkTimeInterval {
   int id; // unique identifier
   float safetyIndex; // level of safety
   // routing weight: perimeter value - for the shortest path algorithm, safety index value - for the safest path
   // algorithm
   float weight;
   // "0" - not restricted, "1" - restricted for motorized vehicles, allowed to push bike,
   // "2" - restricted for every type of mode
   int restriction;
   float linkLength; //perimeter
   road oppositeRoad; //road with opposite direction
   int cyclistsRoad <- 0; //number of traversed cyclists every specified interval of time
   map<int, int> cyclistsByInterval; //list of number of traversed cyclists every specified interval of time
   int cyclistsTotal <- 0; //total amount of traversed cyclists
   aspect base {
      draw shape color:
      rgb(max([105, min([255, 105 + int(cyclistsTotal * 0.15)])]),
         max([105, min([255, 105 + int(cyclistsTotal * 0.15)])]),
         max([105, min([255, 105 + int(cyclistsTotal * 0.15)])]))
      width: 1 + cyclistsTotal * 0.005;
   }

   //Update number of traversed cyclists every specified interval of time.
   reflex updateCyclists when: every(networkTimeInterval #cycle) {
      if oppositeRoad != nil { // traverses of cyclists over an opposite road are registered by an original road
         cyclistsRoad <- cyclistsRoad + oppositeRoad.cyclistsRoad;
      }
      add cyclistsRoad at: cycle to: cyclistsByInterval;
      cyclistsTotal <- cyclistsTotal + cyclistsRoad;
      cyclistsRoad <- 0;
   }
}

// Road intersections
species intersection schedules: [];

// Counting stations that register traversing cyclists every specified interval of time
species countingStation frequency: countingStationTimeInterval {
   string stationName;
   string stationKey;
   int cyclistsStation <- 0; // number of traversed cyclists every specified interval of time
   // number of traversed cyclists every specified interval of time for the chart display
   int cyclistsStationChart <- 0;
   map<int, int> observedCounts; //number of traversed cyclists from the observed data
   init {
      shape <- circle(12, location);
   }
   aspect base {
      draw circle(0.1) color: #mediumorchid;
   }
   //Register cyclists that passed counting stations every "countingStationTimeInterval" of cycles (min)
   reflex saveCoutningData when: every(countingStationTimeInterval #cycle) {
      save [cycle, stationKey, cyclistsStation] to: countsFileName format: "csv" rewrite: false;
      cyclistsStationChart <- cyclistsStation;
      cyclistsStation <- 0;
   }
}

//////////////////////////////////////////////////////EXPERIMENT////////////////////////////////////////////////////////
experiment bicycle_model type: gui {
   parameter "interval to save cyclists on network, in min" category: "Output parameters"
     var: networkTimeInterval <- 60;
   parameter "interval to save cyclists at counting stations, in min" category: "Output parameters"
     var: countingStationTimeInterval <- 60;
   parameter "interval to save active cyclists, in min" category: "Output parameters"
     var: activeCylistsTimeInterval <- 60;
   parameter "routing algorithm" category: "Network" var: routingAlgorithm <- "safest path"
     among: ["safest path", "shortest path"];
   parameter "bicycle infrastructure weight" category: "Network" var: bicycleInfrastructureWeight <- 0.2;
   parameter "traffic volume of motorized vehicles weight" category: "Network" var: mitVolumeWeight <- 0.0;
   parameter "designated route weight" category: "Network" var: designatedRouteWeight <- 0.1;
   parameter "road category weight" category: "Network" var: roadCategoryWeight <- 0.3;
   parameter "max speed weight" category: "Network" var: maxSpeedWeight <- 0.1;
   parameter "adjacent edge weight" category: "Network" var: adjacentEdgeWeight <- 0.0;
   parameter "parking weight" category: "Network" var: parkingWeight <- 0.1;
   parameter "pavement weight" category: "Network" var: pavementWeight <- 0.1;
   parameter "lane width weight" category: "Network" var: widthLaneWeight <- 0.0;
   parameter "gradient weight" category: "Network" var: gradientWeight <- 0.1;
   parameter "rails weight" category: "Network" var: railsWeight <- 0.0;
   parameter "lanes number weigth" category: "Network" var: numberLaneWeight <- 0.0;
   parameter "landuse weight" category: "Network" var: landuseWeight <- 0.0;
   parameter "designated route adjusted weight" category: "Network" var: designatedRouteAdjusted <- 2.0;
   parameter "rails adjusted weight" category: "Network" var: railsAdjusted <- 0.6;
   parameter "pavement adjusted weight" category: "Network" var: pavementAdjusted <- 0.4;
   parameter "gradient adjusted weight" category: "Network" var: gradientAdjusted <- 0.4;
   parameter "bridge value" category: "Network" var: bridgeValue <- 3.0;
   parameter "push value" category: "Network" var: pushValue <- 3.0;
   output {
      display city_map type: opengl background: rgb(10, 40, 55) {
         species road aspect: base;
         species countingStation aspect: base;
         species person aspect: base;
      }
      display activeAgents type: java2D refresh: every(10 #cycle) {
```

```
    chart "Total number of active cyclists" type: series size: {1, 0.5} position: {0, 0} {
        data "Active cyclists" value: person count (each.status = "moving") style: line color: #black;
    }
    chart "Active cyclists by trip purpose" type: series size: {1, 0.5} position: {0, 0.5} {
        data "School" value: person count (each.status = "moving" and each.activityType = "school")
            style: line color: #mediumseagreen;
        data "University" value: person count (each.status = "moving" and each.activityType = "university")
            style: line color: #plum;
        data "Work" value: person count (each.status = "moving" and each.activityType = "work")
            style: line color: #royalblue;
        data "Recreation" value: person count (each.status = "moving" and each.activityType = "recreation")
            style: line color: #khaki;
        data "Shop" value: person count (each.status = "moving" and each.activityType = "shop")
            style: line color: #chocolate;
        data "Other activity" value: person count (each.status = "moving" and each.activityType = "other_place")
            style: line color: #darkcyan;
        data "Home" value: person count (each.status = "moving" and each.activityType = "home")
            style: line color: #cadetblue;
        data "Business" value: person count (each.status = "moving" and each.activityType = "business")
            style: line color: #maroon;
        data "Authority" value: person count (each.status = "moving" and each.activityType = "authority")
            style: line color: #darkgrey;
        data "Doctor" value: person count (each.status = "moving" and each.activityType = "doctor")
            style: line color: #coral;
        data "Bringing" value: person count (each.status = "moving" and each.activityType = "bringing")
            style: line color: #seagreen;
    }
}
display activeAgentsAtStations type: java2D refresh: every(60 #cycle) {
    chart "Active cyclists at " + countingStation(0).stationName + " per hour" type: series
        size: stationChartSize position: {0, 0} {
            data "simulated counts" value: countingStation(0).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(0).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(1).stationName + " per hour" type: series
        size: stationChartSize position: {0.3, 0} {
            data "simulated counts" value: countingStation(1).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(1).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(2).stationName + " per hour" type: series
        size: stationChartSize position: {0.6, 0} {
            data "simulated counts" value: countingStation(2).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(2).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }

    chart "Active cyclists at " + countingStation(3).stationName + " per hour" type: series
        size: stationChartSize position: {0, stationChartSize.y} {
            data "simulated counts" value: countingStation(3).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(3).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }

    chart "Active cyclists at " + countingStation(4).stationName + " per hour" type: series
        size: stationChartSize position: {0.3, stationChartSize.y} {
            data "simulated counts" value: countingStation(4).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(4).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(5).stationName + " per hour" type: series
        size: stationChartSize position: {0.6, stationChartSize.y} {
            data "simulated counts" value: countingStation(5).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(5).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(6).stationName + " per hour" type: series
        size: stationChartSize position: {0, stationChartSize.y * 2} {
            data "simulated counts" value: countingStation(6).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(6).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(7).stationName + " per hour" type: series
        size: stationChartSize position: {0.3, stationChartSize.y * 2} {
            data "simulated counts" value: countingStation(7).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(7).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(8).stationName + " per hour" type: series
        size: stationChartSize position: {0.6, stationChartSize.y * 2} {
            data "simulated counts" value: countingStation(8).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(8).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(9).stationName + " per hour" type: series
        size: stationChartSize position: {0, stationChartSize.y * 3} {
            data "simulated counts" value: countingStation(9).cyclistsStationChart
                style: line color: #goldenrod;
            data "observed counts" value: countingStation(9).observedCounts[int(cycle / 60)]
                style: line color: #gamablue;
    }
    chart "Active cyclists at " + countingStation(10).stationName + " per hour" type: series
        size: stationChartSize position: {0.3, stationChartSize.y * 3} {
```

```
                data "simulated counts" value: countingStation(10).cyclistsStationChart
                   style: line color: #goldenrod;
                data "observed counts" value: countingStation(10).observedCounts[int(cycle / 60)]
                   style: line color: #gamablue;
            }
            chart "Active cyclists at " + countingStation(11).stationName + " per hour" type: series
               size: stationChartSize position: {0.6, stationChartSize.y * 3} {
                data "simulated counts" value: countingStation(11).cyclistsStationChart
                   style: line color: #goldenrod;
                data "observed counts" value: countingStation(11).observedCounts[int(cycle / 60)]
                   style: line color: #gamablue;
            }
         }
      }
   }
}
// Repeating simulation 200 times
experiment repeated type: batch repeat: 200 keep_seed: true {
   parameter "repeated flag" category: "Batch experiment" var: repeatedFlag <- true;
   parameter "interval to save cyclists on network, in min" category: "Output parameters"
      var: networkTimeInterval <- 60;
   parameter "interval to save cyclists at counting stations, in min" category: "Output parameters"
      var: countingStationTimeInterval <- 60;
   parameter "interval to save active cyclists, in min" category: "Output parameters"
      var: activeCylistsTimeInterval <- 60;
   parameter "routing algorithm" category: "Network" var: routingAlgorithm <- "safest path"
      among: ["safest path", "shortest path"];
   parameter "bicycle infrastructure weight" category: "Network" var: bicycleInfrastructureWeight <- 0.2;
   parameter "traffic volume of motorized vehicles weight" category: "Network" var: mitVolumeWeight <- 0.0;
   parameter "designated route weight" category: "Network" var: designatedRouteWeight <- 0.1;
   parameter "road category weight" category: "Network" var: roadCategoryWeight <- 0.3;
   parameter "max speed weight" category: "Network" var: maxSpeedWeight <- 0.1;
   parameter "adjacent edge weight" category: "Network" var: adjacentEdgeWeight <- 0.0;
   parameter "parking weight" category: "Network" var: parkingWeight <- 0.1;
   parameter "pavement weight" category: "Network" var: pavementWeight <- 0.1;
   parameter "lane width weight" category: "Network" var: widthLaneWeight <- 0.0;
   parameter "gradient weight" category: "Network" var: gradientWeight <- 0.1;
   parameter "rails weight" category: "Network" var: railsWeight <- 0.0;
   parameter "lanes number weigth" category: "Network" var: numberLaneWeight <- 0.0;
   parameter "landuse weight" category: "Network" var: landuseWeight <- 0.0;
   parameter "designated route adjusted weight" category: "Network" var: designatedRouteAdjusted <- 2.0;
   parameter "rails adjusted weight" category: "Network" var: railsAdjusted <- 0.6;
   parameter "pavement adjusted weight" category: "Network" var: pavementAdjusted <- 0.4;
   parameter "gradient adjusted weight" category: "Network" var: gradientAdjusted <- 0.4;
   parameter "bridge value" category: "Network" var: bridgeValue <- 3.0;
   parameter "push value" category: "Network" var: pushValue <- 3.0;
   permanent {
   }
}
```

Anhang G: R-Script für Modell Analyse

https://firesoft.ch/unigis/velo/test.R

```
library(dplyr)
library(stringr)
library(data.table)
library(ggplot2)
library(sf)
# Working Directory setzten
#setwd("./Gama_Workspace/bicycle-model_v2.0.1/includes/")
setwd("./Gama_Workspace/bicycle-model_v2.1.0/includes/")

# Zählstationen
stations <- list()
stationShape <- st_read('model_input/shapefiles/counting_stations.shp')
stationCount<-0
for (station in stationShape$stat_name){
  stationCount<-stationCount+1
  if('key' %in% colnames(stationShape)){
    stationKey<-stationShape[stationCount,'key']$key
    stationName<-station
    # Lange Bezeichung des Xylophonweg kürzen
    if(stationName=='(K13) Luzern, Xylophonweg (Reussinsel)'){
      stationName<-'Xylophonweg'
    }
  }else{
    stationKey<-station
    # _ durch Leerschlag ersetzen und Wörter gross schreiben
    stationName<-str_to_title(str_replace(station, '_', ' '))
  }
  stations[[stationCount]] <- c(key=stationKey, name=stationName)
}

# Anzahlt durchgeführte Simulationen
runs <- c(5, 10, 15, 25, 50, 100, 200)

# Observierte Zählungen auslesen
obsFile <- 'model_input/csv_files/hourly_counts_stations.csv'
obs <- read.table(file = obsFile, header = TRUE, sep = ';')
# In allen Spalten real_ entfernen
names(obs) <- str_replace(names(obs), 'real_', '')

# Datatable für Simulationsergebnis erstellen
dt <- data.table(run = integer(),time = integer())
for (station in stations){
  dt <- dt[,(station['key']):=integer()]
}

# Datatable für Simulationsergebnis mit
# Tagestotal und Korrelationskoeffizient erstellen
```

```
dtDay <- data.table(
  run = integer(),
  station = character(),
  count = integer(),
  cor = numeric(),
  p = numeric()
)
# Datatable mit dem Mittelwerten des Simulationsergebnis
dtMean <- data.table(
  station = character(),
  cor = numeric(),
  p = numeric(),
  count = numeric(),
  cv = numeric(),
  real_count = numeric(),
  delta = numeric()
)
# Datatable mit dem Variationskoeffizient
dtCv <- data.table(
  station = character()
)
for (r in runs){
  attrName <- sprintf('cv%s', r)
  dtCv <- dtCv[,(attrName):=numeric()]
}

# Funktion um das CSV der Simulation aufzubereiten
prepareSim <- function (fileName) {
  sim <- read.table(file = fileName, header = TRUE, sep = ',')
  sim <- sim[sim$cycle > 0,] # 0 cycle entfernen
  if('stationKey' %in% colnames(sim)){
    sim <- sim[order(sim$cycle, sim$stationKey), ] # sortieren nach cycle und key
  }else{
    sim <- sim[order(sim$cycle, sim$stationName), ] # sortieren nach cycle und name
  }
  sim$cycle <- sim$cycle / 60 # Minuten in Stunden umwandeln
  names(sim)[1] <- 'time' # Spalte cycle in time umbenennen
  if('stationKey' %in% colnames(sim)){
    sim <- reshape(sim, idvar = 'time', timevar = 'stationKey',
                   direction = 'wide')
  }else{
    sim <- reshape(sim, idvar = 'time', timevar = 'stationName',
                   direction = 'wide')
  }
  # Spalten nur mit Stationsname benennen
  names(sim) <- str_replace(names(sim), 'cyclistsStation.', '')
  rownames(sim) <- 1:nrow(sim) # Zeilen neu nummerieren
  sim
}
maxRuns<-max(runs)
for (run in 0:(maxRuns-1)) {
  # Im Pfad maximale Anzahl Durchläufe und den Run Wert setzen
  fileName <- sprintf('model_output/%s/Simulation %s/counts.csv',
                      as.character(maxRuns), as.character(run))
  sim <- prepareSim(fileName)
  sim['run'] <- run
  dt <- rbind(dt, sim)
  for (station in stations){
    # Korrelationskoeffizient und p-Wert berechnen
    t <- cor.test(obs[[station['key']]], sim[[station['key']]],
                  method="pearson")
    # Summe aller täglicher Durchfahrten
    count <- sum(sim[[station['key']]])
    # Tabelle um neue Zeile ergänzen
    dtDay<-rbind(dtDay, list(run, station['name'], count, t$estimate, t$p.value))
  }
}
# Boxplot der Korrelationskoeffizienten
ggplot(dtDay, aes(y = cor, x = station)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Station", y = "Correlation")
# Boxplot der p-Werte
ggplot(dtDay, aes(y = p, x = station)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Station", y = "p-value")
# Boxplot der simulierten Durchfahrten
ggplot(dtDay, aes(y = count, x = station)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Station", y = "Simulated daily cyclists")

# Funktion für den Boxplot einer Zählstation
boxplotStation <- function(station, displayObs = TRUE){
  p <- ggplot() +
    # Simulations Ergebnis der Station als Boxplot hinzufügen
    geom_boxplot(data = dt, aes_string(y = station['key'], x = 'time', group = 'time'))
  if(displayObs){
    # Gemessene Werte der Zählstation ergänzen
    p <- p + geom_line(
      data = obs,
      aes_string(y = station['key'], x = 'time',
                 # sprintf ist wegen aes_string nötig
                 color='sprintf("observed")')) +
      # Legende definieren
      scale_colour_manual('',
                          breaks = c('observed'),
                          values = c('blue')) +
      # Legende oben Links innerhalb des Graphen positionieren
      theme(legend.justification=c(1,1), legend.position=c(1,1),
            legend.title=element_blank())
  }
```

```
  # Bessere Scalen Einteilung für Stunden (4er Schritte anstelle 5er)
  p <- p + scale_x_continuous(breaks = seq(0, 24, by = 4))
  p <- p + labs(x = 'Time, h',
                y = paste(c('Number of cyclists at',station['name']), collapse=' '))
  p
}

# alle Staionen ausgeben
for (s in stations){
  # print ist für ggblot innerhalbe einer Schleife nötig
  print(boxplotStation(s))

  # Mittelwerte Variationskoeffizient und Summe der Durchfahrten berechnen
  countM <- mean(dtDay[station == s['name'], count])
  cv200 <- sd(dtDay[station == s['name'], count]) / countM # Variationskoeffizient
  realCount <- sum(obs[[s['key']]]) # Summe der observierten Durchfahrten
  dtMean <- rbind(
    dtMean,list(
      s['name'],
      mean(dtDay[station == s['name'], cor]),
      mean(dtDay[station == s['name'], p]),
      countM,
      cv200,
      realCount,
      1/countM*(realCount-countM))
  )
}
if('schanzlgasse' %in% stationShape$stat_name){
  # Schanzlgasse nur simulierte Werte ausgeben
  boxplotStation(c(key='schanzlgasse',name='Schanzlgasse'), FALSE)
}

# Resultate aller Experimente durchgehen um den Variationskoeffizient zu berechnen
first <- TRUE
for (r in runs){
  # Tabelle für tägliche durchfahrten erstellen
  dtDayTemp <- data.table(
    run = integer(),
    station = character(),
    count = integer()
  )
  for (run in 0:(r-1)) {
    # Im Dateiname die Anzahl der Durchläufe (r) und run Wert setzten
    fileName <- sprintf('model_output/%s/Simulation %s/counts.csv', r, run)
    sim <- prepareSim(fileName)
    for (station in stations){
      # Summe aller täglicher Durchfahrten
      count <- sum(sim[[station['key']]])
      # Tabelle um neue Zeile ergänzen
      dtDayTemp<-rbind(dtDayTemp, list(run, station['name'], count))
    }
  }
  for (s in stations){
    # Mittelwerte  der Durchfahrten berechnen
    countM <- mean(dtDayTemp[station == s['name'], count])
    # Variationskoeffizient
    cv <- sd(dtDayTemp[station == s['name'], count]) / countM
    if(first){
      dtCv <- rbind(
        dtCv,list(
          s['name'],
          cv
        ), fill=TRUE
      )
    }else{
      attrName <- sprintf('cv%s', r)
      dtCv[station == s['name'], attrName] <- cv
    }
  }
  first <- FALSE
}
```