TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Rauno Raa

Sõiduradu läbiva programmi loomine ja rakendamine

Bakalaureusetöö (9 EAP)

Juhendajad: Tambet Matiisen, Edgar Sepp, Karl-Johan Pilve

Tartu 2025

Sõiduradu läbiva programmi loomine ja rakendamine

Lühikokkuvõte:

Töö eesmärk on luua programm, mis leiab etteantud geograafilise ala kõikide autoga läbitavate sõiduradade läbimiseks lühima teekonna. Loodud programmi rakendati Tartu, Tallinna ja Helsingi linnadel. Programm planeerib teekonna kõikide piirkonna sõiduradade läbimiseks võimalikult väikese aja- ja kütusekuluga. Kõikide piirkonna sõiduradade läbimine võimalikult optimaalselt on oluline näiteks isejuhtivate autode jaoks kaartide loomisel, aga ka lumekoristuse ja tänavapuhastuse tee planeerimisel. Programmi rakendati osal Tartu linnast, osal Tallinna linnast ja osal Helsingi linnast. Iga eelmainitud piirkonna jaoks leiti kõiki sõiduradu läbiv optimaalne teekond.

Võtmesõnad: Teekonna planeerimine, isejuhtiv auto, graaf, Euleri tee

CERCS: P175 Informaatika, süsteemiteooria; P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria); P110 Matemaatiline loogika, hulgateooria, kombinatoorika

Development and Application of a Program for Traversing Drivable Lanes

Abstract:

The purpose of this thesis is to create a program, which finds the shortest path to traverse all the car-drivable lanes in a given geographical area. Another purpose of the thesis is to apply the program to the cities of Tartu, Tallinn and Helsinki. The program adds the possibility to drive all the given area's car-drivable lanes with minimal time and fuel costs. Traversing all the area's car-drivable lanes as optimally as possible is beneficial for creating maps for self-driving cars and for planning the path for snow ploughing and street cleaning. The program was applied to the part of the city of Tartu, part of the city of Tallinn and part of the city of Helsinki. An optimal path, which traverses all the car-drivable lanes, was found for all the aforementioned areas.

Keywords: Path planning, self-driving car, graph, Eulerian path

CERCS: P175 Informatics, systems theory; P170 Computer science, numerical analysis, systems, control; P110 Mathematical logic, set theory, combinatories

Sisukord

Sissejuhatus

Töö eesmärk on luua programm, mis leiab etteantud geograafilise ala tänavate kõikide autoga läbitavate sõiduradade läbimiseks lühima teekonna. Eesmärk on ka loodud programmi rakendada Tartu, Tallinna ja muudel linnadel. Arendatud programm annab võimaluse läbi sõita kõik piirkonna sõidurajad võimalikult väikese aja- ja kütusekuluga. Kõikide piirkonna sõiduradade läbimine võimalikult optimaalselt on oluline näiteks lumekoristuse ja tänavapuhastuse tee planeerimisel.

Kõikide piirkonna sõiduradade optimaalne läbimine on oluline ka isejuhtivate autode arendajatele. Poggenhans jt [1] on kirjutanud, et üks meetod, mille kaudu isejuhtiv sõiduk liigelda oskab, on eelnevalt kaardistatud ala põhjal liiklemine. Ala kaardistatakse nii, et inimene sõidab erinevate sensoritega, kaameratega ja navigatsioonisüsteemiga varustatud autoga läbi kaardistatava ala kõik sõidurajad. Selle tulemusel saadakse täppiskaart (*high-definition map*). Täppiskaart sisaldab geograafilise ala kohta täpseid andmeid, mille hulka kuuluvad näiteks auto soovitatav trajektoor, auto soovitatav kiirus ning suunatulede kasutamine ehk täppiskaardi abil saab isejuhtiv auto ette põhjalikud ja täpsed andmed selle kohta, kuidas sõiduradadel ning ristmikel tuleb liigelda. Täppiskaarti võib isejuhtival autol vaja minna seetõttu, et reaalajas ei pruugi isejuhtiv auto liiklemiseks saada piisavalt infot isejuhtivas autos olevate sensorite, kaamerate ja navigatsioonisüsteemi kaudu. Peamiselt ei saa isejuhtiv auto piisavalt teavet reaalajas selle kohta, kuidas peab auto paiknema ristmikul liigeldes. Kirjeldatud meetod isejuhtivale autole teabe andmiseks on oluline suuremates linnades, kus esineb rohkem ristmikke ning ristmikud on suuremad ja rohkemate teelahkmetega.

Töö põhisisu jaguneb neljaks peatükiks. Esimeses peatükis kirjeldatakse tööga seotud mõisteid ja ülesandeid. Teises peatükis kirjeldatakse, kuidas programm töötab, missugune on programmi sisend ning kuidas vaadata ja tõlgendada programmi tagastatud tulemusi. Kolmandas peatükis tutvustatakse piirkondi, mille peal programmi rakendati ning tutvustatakse ja analüüsitakse saadud tulemusi. Neljandas peatükis tuuakse välja programmi puudused ning võimalikud edasiarendamise võimalused. Töös on kasutatud juturoboti ChatGPT [2] abi programmi lähtekoodi kommenteerimisel, lähtekoodi dokumentatsiooni koostamisel ning töö struktuuri koostamisel.

1. Teoreetiline taust

1.1 Graafid

Alljärgnev materjal on refereeritud Buldase jt õpikust [3] ning Dharwadkeri jt õpikust [4]. Graaf ehk lõplik suunamata graaf on järjestatud paar mittetühjast lõplikust tippude hulgast ja lõplikust servade hulgast. Graafi tippude hulka tähistatakse sageli sümboliga V ning graafi servade hulka tähistatakse sageli sümboliga E. Graafi struktuur ehk seos V ja E vahel määratakse mingi funktsiooni etteandmisega, mis igale servale seab vastavusse mingi tippude paari, mille elemente nimetatakse serva otstippudeks. Suunatud graaf on graaf, mille servad on suunatud. Suunatud serva nimetatakse kaareks. Graaf, mille tipupaaride vahel võib esineda mitu serva, on multigraaf. Suunatud multigraaf on graaf, mille servad on suunatud ning tipupaaride vahel võib esineda mitu samasuunalist serva. Töö programmis kasutatakse sõiduradade töötlemiseks suunatud graafe ja suunatud multigraafe.

Järgnev lõik on refereeritud Buldase jt õpikust [3] ning Dharwadkeri jt õpikust [4]. Graafi nimetatakse sidusaks, kui graafis leidub tee iga kahe tipu vahel. Suunatud graafi nimetatakse nõrgalt sidusaks, kui kaarte suundade ärajätmisel saadav graaf on sidus. Suunatud graafi nimetatakse tugevalt sidusaks, kui leidub tee igast tipust igasse teise tippu. Graafi komponentideks nimetatakse graafi mittesidusaid osasid. Suunatud graafi tipu väljundastmeks nimetatakse tipust väljuvate kaarte arvu. Suunatud graafi tipu sisendastmeks nimetatakse tippu sisenevate kaarte arvu. Suunatud graaf on tasakaalus, kui iga tipu väljundaste on võrdne tipu sisendastmega.

1.2 Euleri tee

Järgnev materjal on refereeritud Buldase jt õpikust [3] ning Dharwadkeri jt õpikust [4]. Euleri tee on tee, mis kulgeb mööda graafi kõiki servi (suunatud graafis kaari), läbides igat serva (suunatud graafis kaart) täpselt ühe korra. Töö programm leiab Euleri tee suunatud graafis või suunatud multigraafis. Suunatud graaf ja suunatud multigraaf sisaldavad Euleri teed parajasti siis, kui on täidetud kõik järgnevad tingimused:

- graaf on nõrgalt sidus või tugevalt sidus;
- graaf on tasakaalus või sisaldab täpselt ühte tippu, mille väljundaste on sisendastmest täpselt ühe võrra suurem, ja täpselt ühte tippu, mille sisendaste on väljundastmest täpselt ühe võrra suurem.

1.3 Seostamise ülesanne

Järgnev lõik on refereeritud Crouse artiklist [5]. Kahedimensiooniline seostamise ülesanne (*two-dimensional assignment problem*) ehk lineaarne seostamise ülesanne (*linear sum assignment problem*) on ülesanne, mis hõlmab igale töö tegijale (*agent*) täpselt ühe töö (*task*) määramist viisil, mis hoiaks tööde hinna ehk maksumuse (*cost*) võimalikult madala. Töö tegemise hindasid kujutatakse tavaliselt kulumaatriksina (*cost matrix*), kus iga maatriksi element tähistab vastava töö tegija poolt vastava töö tegemise hinda. Lineaarne seostamise ülesanne on üldise seostamise ülesande erijuht, kus on täidetud kõik järgnevad tingimused:

- igale töö tegijale on määratud täpselt üks töö ja igale tööle on määratud täpselt üks töö tegija;
- hinnad on määratud lineaarselt ehk tööde tegemise kogu hinna arvutamiseks piisab kokku liita kõik töö tegemise maksumused;
- igale tööle töö tegija määramine on üksteisest sõltumatu ehk ühele tööle töö tegija määramine ei muuda teiste juba määratud tööde maksumust.

Töös kasutatakse lineaarset seostamise ülesannet ja selle lahendamise algoritmi suunatud graafi optimaalsel tasakaalustamisel.

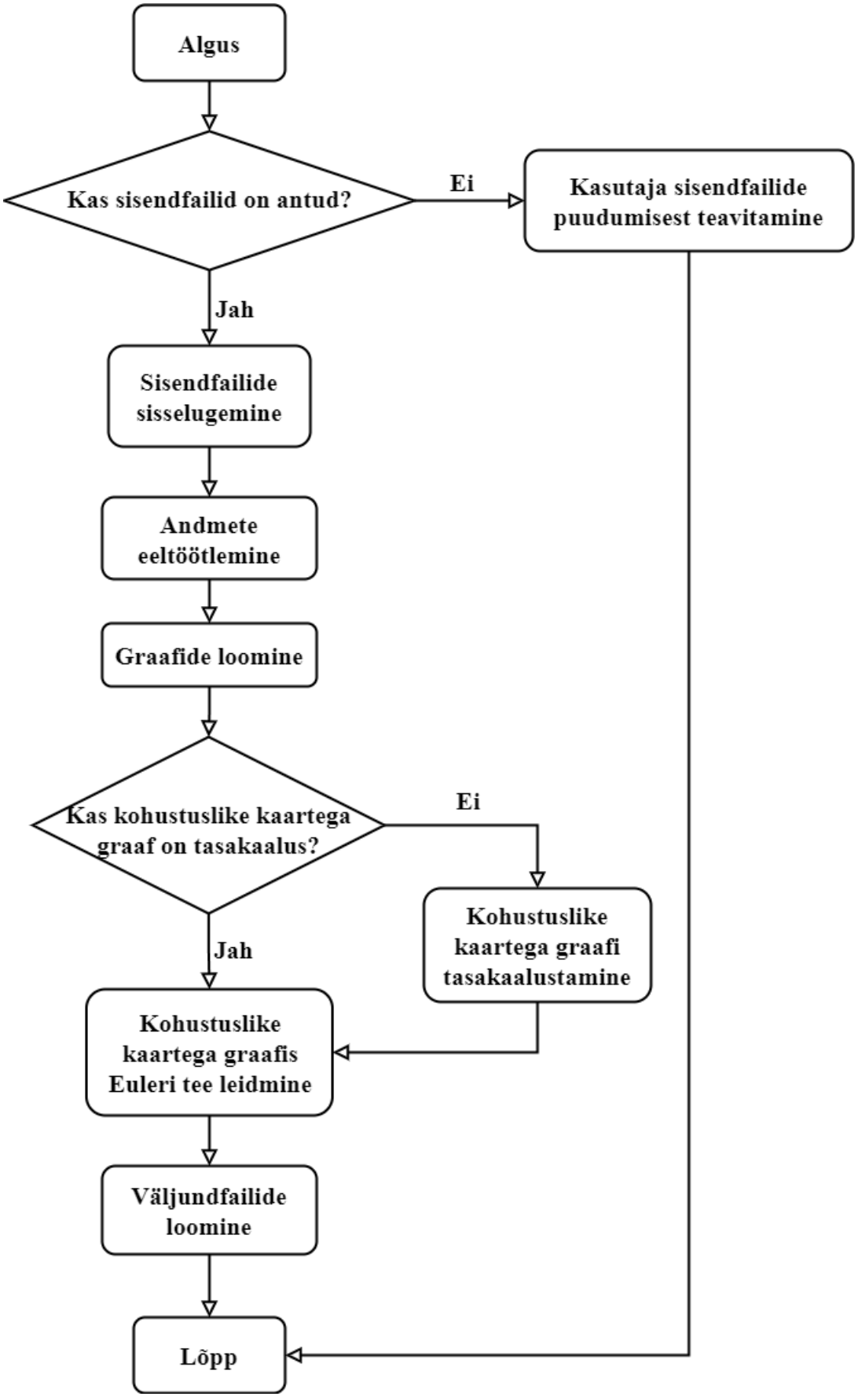
Alljärgnev materjal on refereeritud Crouse artiklist [5]. Lineaarsele seostamise ülesandele leidub tugeva polünomiaalse keerukusega lahendamise algoritm. Algoritmi nõrk polünomiaalne keerukus tähendab, et algoritmi töö aega saab defineerida mingi polünomiaalse funktsiooniga algoritmi sisendi suurusest, kuid kindlatel sisenditel võib algoritm olla väga aeglane. Tugev polünomiaalne keerukus tähendab, et algoritmi töö aega saab defineerida mingi polünomiaalse funktsiooniga algoritmi sisendi suurusest ning algoritmile ei leidu sisendit, mille puhul oleks see tunduvalt aeglasem. Töös kasutatakse seostamise ülesande lahendamisel muudetud varianti Jonker-Volgenanti algoritmist, muudetud Jonker-Volgenanti algoritm on tugeva polünomiaalse keerukusega algoritm [5,6].

2. Programmi ülevaade

Peatükis kirjeldatakse üldiselt ja detailselt programmi töövoogu. Samuti antakse ülevaade erinevatest programmiga seotud tehnoloogiatest. Lisaks tuuakse välja, mida programm sisendiks vajab ning mida teha programmi väljundiga.

2.1 Programmi üldine töövoog

Programmi ülesanne on leida etteantud geograafilise piirkonna kõikide sõiduradade läbimiseks võimalikult lühike tee. Eesmärgi saavutamiseks loeb programm sisse sisendfailid, eeltöötleb sisendfailidest loetud geograafilisi andmeid, loob eeltöödeldud andmete põhjal suunatud graafi, leiab lühima tee graafi kõikide kaarte läbimiseks ning loob väljundfailid.



Joonis 1. Programmi üldise töövoo diagramm.

Programmi üldine töövoog on näha jooniselt 1. Programmi käivitamisel tagatakse esmalt, et vajalikud sisendfailid on antud, sest vastasel juhul ei ole andmeid, mille põhjal oleks võimalik edasisi töövoo samme täita. Seega, kui sisendfaile ei leita, teavitatakse sellest kasutajat ning programm lõpetab töö. Peale sisendfailide leidmist salvestab programm sisendfailide sisu edasiseks töötlemiseks mällu. Peale failide sisselugemist viiakse mällu salvestatud andmed sobivale kujule, et nende põhjal oleks võimalik luua suunatud graaf.

Eeltöödeldud andmete põhjal luuakse kaks suunatud graafi (joonisel 1 ja edaspidi nimetatud ka kui graaf). Ühes graafis on kõik sellised kaared, mille läbimine on kohustuslik. Teises graafis on lisaks kohustuslikele kaartele ka sellised kaared, mille läbimine ei ole kohustuslik, kuid mille kaasamine võib anda lühema kõiki graafi kaari läbiva teekonna. Mõlemad graafid luuakse nii, et need oleksid tugevalt sidusad. Kuigi Euleri tee võib leiduda graafis ka siis, kui graaf on nõrgalt sidus nii, et täpselt ühest tipust ei ole väljuvaid kaari (sellisel juhul ei saa graaf enam olla tugevalt sidus) ning vastavasse tippu on täpselt üks sisenev kaar (sellise tipu sisendaste on 1 ja väljundaste on 0) ja täpselt ühe tipu väljundaste on täpselt ühe võrra suurem sisendastmest [3], siis lihtsuse mõttes tehakse graafid tugevalt sidusaks, sest tõenäoliselt ei mõjutaks ühe väljuvate kaarteta tipu kaasamine Euleri tee pikkust ning ühe kaare ehk ühe sõiduraja mitteläbimine on aktsepteeritav.

Peale graafide loomist kontrollitakse, kas kohustuslike kaartega graaf on tasakaalus või on graafis täpselt üks selline tipp, kus on väljundaste on ühe võrra suurem kui sisendaste ning täpselt üks selline tipp, kus on sisendaste ühe võrra suurem kui väljundaste. Kui suunatud graaf on tasakaalus või leiduvad kaks eelkirjeldatud tippu ja graaf on tugevalt sidus, siis see tähendab, et selles graafis peab leiduma Euleri tee [3]. Kui graaf ei ole Euleri tee leidumiseks piisavalt tasakaalus, siis on vaja graaf tasakaalustada. Graaf viiakse tasakaalustamisel olukorda, et graafis leiduks täpselt üks tipp, mille sisendaste on ühe võrra suurem kui väljundaste ning täpselt üks tipp, mille väljundaste on ühe võrra suurem kui sisendaste. Kui kohustuslike kaartega graaf on tasakaalus või Euleri tee leidumiseks piisavalt tasakaalu viidud, siis leitakse kohustuslike kaartega graafi Euleri tee.

Peale Euleri tee leidmist salvestab programm kohustuslike kaartega graafi, selles sisalduva Euleri tee, graafi kõikide kaarte pikkuse, Euleri tee pikkuse, Euleri tee ja graafi kaarte pikkuse

vahelise suhte ning programmi tööaja väljundfailidesse. Peale väljundfailide loomist lõpetab programm töö.

2.2 Programmi detailne ülevaade

Programm on loodud programmeerimiskeeles Python[1]. Python otsustati valida peamiselt seetõttu, et Pythoni jaoks on tehtud mitmeid teeke, mis lihtsustavad töö eesmärki täitva programmi loomist. Programm on avatud lähtekoodiga, et programm oleks tasuta kasutatav ning tulevikus oleks võimalik teistel inimestel programmi edasi arendada. Programmi lähtekoodi internetti üleslaadimiseks on kasutatud versioonihaldustarkvara Git[2]. Lähtekood on üles laetud lähtekoodi majutusteenusesse GitHub[3]. Lähtekoodile viitav link ja programmi käivitamise juhend on leitav lisas I. Programm on käsureapõhine, mis tähendab, et programmil ei ole graafilist kasutajaliidest. Graafilist kasutajaliidest ei arendatud programmile seetõttu, et eeldatavasti saavad programmi peamisteks kasutajateks inimesed, kellel on kogemust käsureapõhiste programmidega.

Järgnevates alapeatükkides kirjeldatakse detailselt programmi töövoo samme. Samuti tutvustatakse olulisemaid Pythoni teeke, mida programmis kasutatakse. Lisaks tutvustatakse programmiga seotud faile, failivorminguid ja rakendusi.

2.2.1 Programmi sisend

Programm nõuab kolme sisendfaili, millega pannakse paika analüüsitav geograafiline ala, kus on vaja leida teekond, mis läbiks kõiki sõiduradasid. Sisendfailid peavad olema loodud veebirakenduse osm2streets [7] abil.

Osm2streets [7] on avatud lähtekoodiga veebirakendus, mis loob avaliku andmebaasi OpenStreetMap (edaspidi OSM) [8] andmete põhjal kihid. Töö jaoks on olulised sõiduradade alade ja ristmike alade kihid, sest need sisaldavad mugavalt töödeldavaid andmeid sõiduradade kohta. Kihtide loomiseks võtab veebirakendus osm2streets [7] vastavalt kasutaja valitud geograafilisele alale sisendiks andmebaasist OSM päritud andmed XML vormingus. Rakendus osm2streets [7] lihtsustab sõiduradade ja ristmike eraldamist andmebaasist OSM, sest esiteks

[1] Python: https://www.python.org/
[2] Git: https://git-scm.com/
[3] GitHub: https://github.com/

jätab rakendus andmebaasist päritud andmetest alles ainult kasutajale sobiva geograafilise ala ning teiseks loob rakendus filtreeritud andmete põhjal eelnevalt kirjeldatud kihid, mida on mugav töödelda.

OSM [8] on avalik andmebaas, mis sisaldab terve planeedi geograafilisi andmeid. Geograafiliste andmete hulka kuuluvad näiteks tänavad, ehitised ja pargid. Andmebaasi uuendavad inimesed ja organisatsioonid kõikidest maailmajagudest. Andmebaasis sisalduvaid andmed ei ole autoriõigustega piiratud ning on vabalt kasutatavad. Andmeid on võimalik alla laadida erinevates failivormingutes.

Töö jaoks on vaja rakenduse osm2streets abil luua ja alla laadida järgmised kolm sisendfaili:

- „osm.xml“,
- „Lane polygons.geojson“,
- „Intersection polygons.geojson“.

Failid „Lane polygons.geojson“ ja „Intersection polygons.geojson“ on mõlemad Geographic JavaScript Object Notation [9] (edaspidi geojson) vorminguga. Geojson [9] on vormingul JavaScript Object Notation (edaspidi JSON) põhinev vorming, millega saab hoiustada geograafilisi andmeid.

Fail „osm.xml“ sisaldab XML vormingus andmebaasi OSM andmeid. Faili kasutatakse programmis andmete eeltöötlemisel, et edasiseks töötlemiseks alles jätta vaid suuremad ja olulisemad sõidurajad. Olulisem atribuut on `way`, mis sisaldab atribuuti `id`, mille abil on võimalik OSM andmeid siduda geojson failide andmetega. Oluline on ka atribuudis `way` sisalduv atribuut `highway`, mis määrab raja tüübi. Atribuuti `highway` kasutatakse sõiduradade filtreerimiseks. Sõiduradade filtreerimist kirjeldatakse täpsemalt andmete eeltöötlemise peatükis. Näiteid genereeritud XML failidest saab lugeda lisast II.

Fail „Lane polygons.geojson“ sisaldab andmeid erinevat tüüpi radade kohta. Näiteid genereeritud „Lane polygons.geojson“ failidest saab vaadata lisast II. Failis sisalduvad olulisemad atribuudid on järgnevad:

1) `geometry`, mis sisaldab raja kuju ja selle asukohta määravat atribuuti `coordinates` ning raja kuju tüüpi määravat atribuuti `type`. Failis „Lane polygons.geojson“ on raja kuju tüübiks alati hulknurk (*polygon*);
2) `properties`, mis sisaldab järgnevaid olulisi atribuute:
    a) `road`, mis määrab raja tänava identifikaatori ehk annab infot selle kohta, mis tänava osa antud sõidurada on;
    b) `index`, mis määrab raja indeksi ehk määrab, mitmes rada see antud tänaval on. Töö raames kirjutatud programmi juures on see kasulik peamiselt selleks, et võimaldab iga rada tänava sees eristada unikaalse väärtusega;
    c) `direction`, mis määrab sõiduraja suuna;
    d) `type`, mis määrab raja tüübi. Töö jaoks kasutatakse failist ainult sõiduraja (*driving*) tüüpi radu;
    e) `osm_way_ids`, mis määrab rajale vastava OSM identifikaatori või vastavad OSM identifikaatorid. Antud atribuudi abil on võimalik leida raja kohta failist „osm.xml“ selliseid kirjeid, mida failis „Lane polygons.geojson“ pole antud. Töö kontekstis on see kasulik sõiduradade filtreerimiseks.

Fail „Intersection polygons.geojson“ sisaldab andmeid ristmike ja tänavate kohta. Näiteid võimalikest genereeritud „Intersection polygons.geojson“ failidest saab vaadata lisast II. Vaadeldava objekti tüübi määrab ära atribuudis `properties` sisalduv atribuut `type`. Töö jaoks vaadeldakse vaid selliseid objekte, mille tüübiks on kas ristmik (*intersection*) või tänav (*road*).

Kui vaadeldav objekt on ristmik, siis olulisemad atribuudid on järgnevad:

1) `geometry`, mis sisaldab ristmiku kuju ja selle asukohta määravat atribuuti `coordinates` ning ristmiku kuju tüüpi määravat atribuuti `type`. Failis „Intersection polygons.geojson“ on ristmiku kuju tüübiks alati hulknurk (*polygon*);
2) `properties`, mis sisaldab lisaks atribuudile `type` järgnevaid olulisi atribuute:
    a) `id`, mis määrab ristmiku identifikaatori;
    b) `intersection_kind`, mis määrab ristmiku tüübi. See on oluline andmete eeltöötluse etapis, kus jäetakse alles vaid sobivat tüüpi ristmikud;
    c) `movements`, mis määrab tänava identifikaatorite paaride abil seosed, missuguselt tänavalt on võimalik ristmikul liikuda missugusele tänavale.

Kui vaadeldav objekt on tänav, siis olulisem atribuut on `properties`, mis sisaldab lisaks atribuudile `type` järgnevaid olulisi atribuute:

1) `id`, mis määrab tänava identifikaatori. Selle atribuudi väärtused vastavad faili „Lane polygons.geojson“ atribuudi `road` väärtustele. See tähendab, et atribuudi `id` põhjal on võimalik failides „Intersection polygons.geojson“ ja „Lane polygons.geojson“ tänavate kohta sisalduvat informatsiooni omavahel seostada;
2) `src_i`, mis määrab ristmiku identifikaatori abil, missugusest ristmikust tänav algab;
3) `dst_i`, mis määrab ristmiku identifikaatori abil, missuguse ristmikuga tänav lõpeb.

Programmi käivitamisel kontrollitakse esmalt, kas sisendfailid on kasutaja poolt määratud kohas olemas. Failiteede määramist tutvustatakse lähemalt lisas I. Seejärel loetakse sisendfailid mällu Pythoni nimekirjana (*list*), mis sisaldab Pythoni sõnastikke (*dictionary*). Geojson vorminguga failid loetakse mällu teegi geojson[4] abil ning XML fail loetakse sõnastikku Pythonisse sisse ehitatud teegi xml.etree.ElementTree[5] abil. XML faili sisselugemisel moodustatakse sõnastik kõikidest failis leiduvatest sõiduradadest, sõnastiku võtmeteks saavad sõiduradade identifikaatorid ning sõnastiku väärtusteks saavad omakorda sõnastikud, mille võtmeteks on vastava sõiduraja kohta käivate atribuutide nimetused ning väärtusteks on vastava sõiduraja kohta käivate atribuutide väärtused.

2.2.2 Andmete eeltöötlemine

Geojson vorminguga failide põhjal mällu loetud andmeid on vaja enne nende põhjal graafide moodustamist töödelda, sest failide esmasel sisselugemisel ei filtreeritud andmeid, mistõttu sisaldavad failide sisselugemisel moodustatud nimekirjas sisalduvad sõnastikud ebavajalikke andmeid ning andmed on kujul, mille põhjal ei saa graafi koostada. Esmalt töödeldakse failist „Lane polygons.geojson“ mällu loetud andmeid, seejärel töödeldakse failist „Intersection polygons.geojson“ mällu loetud andmeid.

Failist „Lane polygons.geojson“ mällu loetud andmete töötlemiseks ja töödeldud andmete salvestamiseks luuakse uus Pythoni sõnastik, kuhu jäetakse sobival kujul graafi moodustamiseks vajalikud andmed. Loodavas sõnastikus grupeeritakse sõidurajad tänavate

[4] Teek geojson: https://pypi.org/project/geojson/
[5] xml.etree.ElementTree: https://docs.python.org/3/library/xml.etree.elementtree.html

kaupa ehk sõnastiku võtmeteks saavad tänavate identifikaatorid ning väärtusteks saavad nimekirjad sõiduradadest, kus iga sõidurada on esitatud sõnastikuna. Iga sõiduraja sõnastik hakkab sisaldama atribuute, mis andmete töötlemise käigus leiti. Sobivate andmete leidmiseks uuritakse faili „Lane polygons.geojson“ esmasel sisselugemisel loodud sõnastike nimekirja iga sõnastikku ehk iga rada, mida fail sisaldas.

Iga raja puhul kontrollitakse esmalt, et raja tüüp oleks sõidurada ehk kontrollitakse, et vaadeldav rada oleks autoga läbitav. Selle jaoks kontrollib programm atribuudi `type` väärtust. Seejärel kontrollib programm iga sõidurajaga seotud OSM identifikaatorile vastavatest OSM andmetest (OSM andmed on faili „osm.xml“ põhjal loodud Pythoni sõnastikus), kas atribuudi `highway` väärtuseks on üks järgnevatest sõnedest [10]:

1) `motorway`, mis tähistab kiirteed;
2) `motorway_link`, mis tähistab kiirtee mahasõiduteed või pealesõiduteed;
3) `trunk`, mis tähistab riigi olulist sõiduteed või olulist maanteed, mis ei ole kiirtee, kuid ühendab näiteks suuremaid linnasid;
4) `trunk_link`, mis tähistab riigi olulise sõidutee mahasõiduteed või pealesõiduteed;
5) `primary`, mis tähistab `trunk` tüüpi sõiduteest vähem olulist sõiduteed;
6) `primary_link`, mis tähistab `primary` tüüpi sõidutee mahasõiduteed või pealesõiduteed;
7) `secondary`, mis tähistab `primary` tüüpi sõiduteest vähem olulist sõiduteed;
8) `secondary_link`, mis tähistab `secondary` tüüpi sõidutee mahasõiduteed või pealesõiduteed;
9) `tertiary`, mis tähistab `secondary` tüüpi sõiduteest vähem olulist sõiduteed;
10) `tertiary_link`, mis tähistab `tertiary` tüüpi sõidutee mahasõiduteed või pealesõiduteed;
11) `unclassified`, mis tähistab `tertiary` tüüpi teest vähem olulist sõiduteed. Seda tüüpi teed ühendavad omavahel näiteks külasid;
12) `residential`, mis tähistab tavaliselt linnasisest sõiduteed, mille ääres on eramajad.

Kui vähemalt ühe vaadeldava sõidurajaga seotud OSM identifikaatorile vastava atribuudi `highway` väärtus oli sobiv, siis on sõidurada sobiv järgmises programmi etapis graafidesse kaasamiseks, mistõttu salvestatakse vaadeldava sõiduraja andmed uude sõnastikku. Sobivate sõiduradade kohta salvestatakse uude sõnastikku järgmised andmed:

1) tänava identifikaator;
2) sõiduraja indeks;
3) sõiduraja suund;
4) teegi shapely[6] abil atribuudis `geometry` sisalduvad andmed.

Failist „Intersection polygons.geojson“ mällu loetud andmete töötlemiseks ja töödeldud andmete salvestamiseks luuakse Pythoni hulk (*set*), kuhu hakatakse lisama sobivate andmetega ristmikke. Loodav hulk hakkab sisaldama ristmiku objekte, ristmiku objektid hoiavad endas järgnevaid andmeid:

1) ristmiku identifikaator;
2) atribuudis movements sisalduvad andmed;
3) teegi shapely abil atribuudis geometry sisalduvad andmed;
4) ristmiku tüüp.

Sobivate andmete leidmiseks uuritakse faili „Intersection polygons.geojson“ esmasel sisselugemisel loodud Pythoni sõnastiku iga võti-väärtus paari. Iga paari korral vaadatakse esmalt, kas tegu on ristmiku või tänavaga. Kui vaadeldav paar on tänav, siis lisatakse vastavalt paaris sisalduvale tänava identifikaatorile igale vastava tänava identifikaatoriga sõiduradade sõnastikus olevale sõidurajale sõiduraja alguse ristmiku identifikaator ning sõiduraja lõpu ristmiku identifikaator. Kui vaadeldav paar on ristmik, siis kontrollitakse esmalt, et ristmiku tüüp ei oleks valitud geograafilise ala äär, sest edasiseks töötlemiseks on vaja võtta vaid sellised tänavad ja ristmikud, mis ei ole ala valiku tõttu ära lõigatud. Kui ristmiku tüüp on sobiv, luuakse vastav ristmiku objekt, mis lisatakse Pythoni hulka.

2.2.3 Graafide loomine

Graafid luuakse andmete eeltöötluse etapis loodud sõiduradade Pythoni sõnastiku ja ristmike objekte sisaldava hulga põhjal. Graafid koostatakse põhimõttel, et graafide tippudeks saavad ristmikud ning graafide kaarteks saavad sõidurajad.

Graafide loomiseks kasutatakse Pythoni teeki NetworkX [11]. NetworkX on avatud lähtekoodiga teek graafide koostamiseks, kuhu on sisseehitatud mitmeid graafi algoritme. Töö

[6] shapely: https://shapely.readthedocs.io/en/stable/

jaoks on teegist kõige olulisem sisseehitatud Euleri tee leidmise algoritm. Euleri tee leidmist kirjeldatakse detailsemalt järgmises peatükis.

Mõlemad koostatavad graafid on suunatud graafid. Luuakse kaks graafi: ühes graafis on sellised kaared, mis tuleb kõik kindlasti läbida, teises graafis on lisaks kindlasti läbimist vajavatele kaartele ka sellised kaared, mille läbimine ei ole kohustuslik, kuid mille kaasamine võib anda lühema Euleri tee.

Esmalt luuakse graafi tipud, milleks on ristmikud. Graafid luuakse nii, et iga sõiduraja kohta luuakse vastavat sõidurada sisaldavatel ristmikel üks tipp, et minimeerida Euleri tee leidmisel ebavajalikke reavahetusi. See saavutatakse nii, et iga tänava sõiduraja kohta duplitseeritakse vastava tänava ristmikud. Iga graafi tippu salvestatakse järgnevad andmed:

1) graafi tipu unikaalne identifikaator, mis on kolmik, mis koosneb ristmiku identifikaatorist, tänava identifikaatorist ja sõiduraja indeksist;
2) info, kas vastava sõiduraja kohta loodav tipp on ristmikule sisenemiseks või ristmikult väljumiseks;
3) tipu geograafilised koordinaadid, mis saadakse nii, et leitakse sõiduraja hulknurga ja ristmiku hulknurga üksteisele lähimat kaks punkti, millest tipu geograafilisteks koordinaatideks saab nendest kahest punktist sõiduraja hulknurga punkt (sõiduraja punkt valitakse seetõttu, et hiljem teekonna visualiseerimisel jääksid tipud vastava sõiduraja kaare peale). Kahe punkti vahelisi kaugusi arvutatakse teegi geopy[7] abil.

Seejärel luuakse graafides ristmike sisemised kaared, mis tähistavad teekondi, kuidas ristmiku sees on võimalik liikuda. Selleks vaadatakse läbi kõik ristmikud ning ristmike sees sisalduvad tipud. Iga ristmiku korral vaadatakse läbi ristmikku sisenevad tipud ja ristmikust väljuvad tipud. Mõlemas graafis moodustatakse kaar iga ristmikku siseneva tipu ja ristmikust väljuva tipu vahel, välja arvatud selliste tippude vahel, mis kuuluvad samale tänavale (millel on sama tänava identifikaator), sest sellised kaared tähistavad tagasipöördeid. Tagasipöördeid tähistavad kaared lisatakse ainult sellesse graafi, kus leiduvad ka sellised kaared, mille läbimine ei ole kohustuslik. Iga loodava kaare kohta antakse kaasa järgnevad andmed:

1) kaare lähtetipu identifikaator;
2) kaare lõpptipu identifikaator;

[7] geopy: https://geopy.readthedocs.io/en/stable/

3) kaare geograafilised koordinaadid, mis moodustatakse kaare lähtetipu koordinaatidest ja kaare lõpptipu koordinaatidest. Kuna tipud on kujutatud punktidena, siis ristmikusisesed kaared on sirgjooned;
4) kaare pikkus, mis arvutatakse kaare geograafiliste koordinaatide põhjal teegi geopy abil;
5) kaare tüüp, et oleks võimalik eristada, kas kaar on ristmikusisene või ristmikuväline. Siin etapis on iga kaar ristmikusisene kaar.

Seejärel luuakse graafides ristmikest väljas paiknevad kaared, mis tähistavad tänavate sõiduradu. Selle jaoks vaadatakse läbi kõik graafi tipud. kaar moodustatakse iga sellise tipupaari vahel, mille tänava identifikaator ja sõiduraja indeks on ühesugused. Kaare suund leitakse nii, et vaadatakse kumb tipp vaadeldavas tipupaaris on ristmikku sisenev. Iga loodava ristmikuvälise kaare kohta antakse kaasa samasugused andmed, mis antakse kaasa ristmikusisestele kaartele. Ristmikuväliste kaarte puhul leitakse kaare geograafilised koordinaadid sõiduradade sõnastikust vastavalt vaadeldava sõiduraja identifikaatorile. Selles etapis on iga loodud kaar ristmikuväline kaar.

Euleri tee leidumiseks suunatud graafis on tarvilik tingimus, et graaf on tugevalt sidus, kuid kuna loodud graafid ei pruugi olla tugevalt sidusad, siis leitakse teegi NetworkX abil mõlemas graafis suurim tugevalt sidus komponent. Edasistes programmi etappides kasutatakse graafidena vastavaid tugevalt sidusaid komponente. Teek NetworkX kasutab graafis tugevalt sidusate komponentide leidmiseks iteratiivset varianti Nuutila muudatustega Tarjani algoritmist [12,13].

Seejärel eemaldatakse graafis, mille kõikide kaarte läbimine on kohustuslik, nii palju ristmikusiseseid kaari kui võimalik, et minimeerida pöördeid. Selleks vaadatakse üle kõik graafi ristmikusisesed kaared. Igast ristmikust on lubatud eemaldada vaadeldav ristmikusisene kaar juhul, kui kaare eemaldamisel jääb kaare lähtetipu väljaminevate kaarte arv suuremaks kui üks, kaare lõpptipu sissetulevate kaarte arv suuremaks kui üks ning graaf jääb tugevalt sidusaks.

2.2.4 Graafi kõiki kaari läbiva teekonna leidmine

Euleri tee suunatud graafis leidumise teine tingimus on, et graaf peab olema tasakaalus või leidub graafis täpselt üks selline tipp, kus on väljundaste on ühe võrra suurem kui sisendaste, ning täpselt üks selline tipp, kus on sisendaste ühe võrra suurem kui väljundaste [3]. Kuna peale graafide loomist on täidetud juba Euleri tee suunatud graafis leidumise esimene tingimus, milleks on see, et graaf peab olema tugevalt sidus, siis programm proovib graafis leida Euleri teed. Kui Euleri teed polnud graafis võimalik leida, siis on teada, et graafis on Euleri tee leidumise teine tingimus täitmata ning graaf on vaja tingimuse täitmiseks piisavalt tasakaalustada.

Graafi piisavaks tasakaalustamiseks on vaja moodustada teekonnad igast tipust, kus sisendaste on rangelt suurem kui väljundaste, sellistesse tippudesse, mille väljundaste on rangelt suurem kui sisendaste. Selliste teekondade moodustamise tulemusel läheneb iga tasakaalust väljas tipp tasakaalule, sest liigse sisendastmega tippu väljuva kaare kaare lisamine suurendab tipu väljundastet ühe võrra ning liigse väljundastmega tippu siseneva kaare lisamine suurendab tipu sisendastet ühe võrra. Sellist tasakaalustamist tehakse seni, kuni graafis on täpselt üks tipp, mille väljundaste on täpselt ühe võrra suurem sisendastmest, ning üks tipp, mille sisendaste on täpselt ühe võrra suurem väljundastmest.

Teekondade moodustamiseks igast tipust, kus sisendaste on rangelt suurem kui väljundaste, sellistesse tippudesse, mille väljundaste on rangelt suurem kui sisendaste, uurib programm tippe, mis on tasakaalust väljas. Tippude astmeid vaadatakse selles graafis, mis sisaldab vaid selliseid kaari, mille läbimine on kohustuslik. Iga tipu kohta leitakse, kui palju on tipp tasakaalust väljas. Programmis moodustatakse tasakaalust väljas tippude jaoks kaks sõnastikku: ühte sõnastikku lisatakse tipud, mille väljundaste on suurem kui sisendaste, ning teise sõnastikku lisatakse tipud, mille sisendaste on suurem kui väljundaste. Mõlema sõnastiku puhul saab võtmeks tipp ning väärtuseks arv, kui palju on tipp tasakaalust väljas.

Seejärel leitakse iga suurema sisendastmega tipu kohta teekonnad igasse suurema väljundastmega tippu. Lühimaid teekondi leitakse selles graafis, mis sisaldab ka selliseid kaari, mille läbimine ei ole kohustuslik. Iga tipu teekondade arvutamiseks kasutatakse teeki NetworkX, mis kasutab Dijkstra algoritmi [14,15]. Leitud teekonnad salvestatakse Pythoni sõnastikku, kus võtmeks on paar, mille esimeseks elemendiks on on teekonna algtipp ning

teiseks elemendiks on teekonna lõpptipp, ning väärtuseks on teekond. Teekond on esitatud tipupaaride nimekirjana, kus iga tipupaar tähistab ühte kaare nii, et paari esimene element on kaare algus, paari teine element on kaare lõpp. Lisaks salvestatakse Pythoni sõnastikku ka iga leitud teekonna pikkused. Teekonna pikkuste sõnastiku puhul on võtmeks samuti tippude paar, mille esimeseks elemendiks on teekonna algtipp ning teiseks elemendiks on teekonna lõpptipp ning väärtuseks on teekonna pikkus.

Leitud andmete põhjal on võimalik kohustuslike kaartega graaf tasakaalustada, kui lisada kohustuslike kaartega graafi leitud lühimad teed nii, et igast suurema sisendastmega tipust lisatakse graafi vastavalt nii palju teekondi, kui palju on vastava suurema sisendastmega tipu sisendaste suurem väljundastmest. Lisateekonnad on vaja moodustada viisil, et graafi igat kaart vähemalt ühe korra läbiv tee pikeneks peale lisateekondade lisamist võimalikult vähe. Optimaalset lisateekondade moodustamist saab vaadelda kui seostamise ülesannet: iga suurema sisendastmega tipp on seostamise ülesande mõttes kui töö tegija, iga suurema väljundastmega tipp on seostamise ülesande mõttes kui töö ning teekonna pikkus suurema sisendastmega tipu ja suurema väljundastmega tipu vahel on seostamise ülesande mõttes kui töö tegemise maksumus.

Seostamise ülesande lahendamiseks luuakse esmalt kulumaatriks. Kulumaatriksi koostamisel saavad maatriksi ridadeks tipud, mille sisendaste on suurem kui väljundaste, ning veergudeks tipud, mille väljundaste on suurem kui sisendaste. Kulumaatriksi loomisel duplitseeritakse iga suurema sisendastmega tippu nii palju, kui on vastava tipu sisendaste suurem väljundastmest, ning iga suurema väljundastmega tippu nii palju, kui on vastava tipu väljundaste suurem sisendastmest. Kulumaatriksi loomisel duplitseeritakse tippe seetõttu, et selle tulemusel on kulumaatriksil alati võrdne arv ridu ja veerge. Kulumaatriksi elementideks saavad eelnevalt leitud teekondade pikkused suurema sisendastmega tipust suurema väljundastmega tippudesse. Peale kulumaatriksi loomist leitakse kulumaatriksis optimaalsed seosed teegi scipy[8] abil. Scipy kasutab optimaalsete seoste leidmiseks muudetud Jonker-Volgenanti algoritmi [6].

Kui optimaalsed seosed ehk optimaalsed lisateekonnad on leitud, siis moodustatakse optimaalsed lisateekonnad kohustuslike kaartega graafis. Teekondade moodustamisel duplitseeritakse graafis olemasolevaid kaari või kui teekonnas sisalduvat kaare varem graafis

[8] scipy: https://scipy.org/

ei olnud, siis lisatakse uus kaar graafi. Kuna kaari võib olla vaja duplitseerida, siis muudetakse vastava graafi tüüp suunatud graafist suunatud multigraafiks. Teekondi moodustatakse, kuni graafi jääb alles üks tipp, mille väljundaste on sisendastmest täpselt ühe võrra suurem, ning üks tipp, mille sisendaste on väljundastmest täpselt ühe võrra suurem. Peale teekondade lisamist ainult kohustuslikult läbitavate kaartega graafi leitakse selles graafis Euleri tee. Euleri tee leitakse teegi NetworkX abil [16]. Euleri teed alustatakse sellest tipust, mille väljundaste on suurem. Euleri tee salvestatakse tipupaaride nimekirjana, kus iga tipupaari puhul on esimene paari element kaare algtipp ning teine paari element on kaare lõpptipp.

Peale Euleri tee leidmist arvutatakse kordaja, mis mõõdab graafi iga kaart vähemalt ühe korra läbiva tee efektiivsust. Kordaja arvutamiseks jagatakse Euleri tee pikkus graafi kõikide unikaalsete kaarte kogupikkusega. Kordaja näitab, kui palju on graafis iga kaart vähemalt ühe korra läbiva tee puhul vaja keskmiselt ühte kaart läbida, seega mida väiksem on kordaja, seda optimaalsem on graafi iga kaart vähemalt korra läbiv tee. Kui kordaja on üks, siis see tähendab, et iga graafi kaart läbitakse täpselt ühe korra - see on kõige optimaalsem graafi kõiki kaari vähemalt ühe korra läbiv tee.

2.2.5 Programmi väljund

Peale Euleri tee leidmist ja kordaja arvutamist salvestatakse tekstifaili (laiendiga .txt) järgnevad andmed:

1) programmi tööaeg kuni Euleri tee kordaja arvutamiseni (kaasa arvatud) minutites;
2) graafi kõikide kaarte pikkus meetrites;
3) Euleri tee pikkus meetrites;
4) Euleri tee efektiivsuse kordaja.

Seejärel salvestatakse graaf ja Euleri tee faili. Fail on GeoPackage vorminguga, faili laiend on .gpkg. GeoPackage [17] on avatud ja tasuta failivorming, millega on võimalik kompaktselt salvestada geograafilisi andmeid ning muid andmeid. Graaf salvestatakse faili kahe kihina: esmalt luuakse ja salvestatakse faili graafi kaarte kiht, seejärel luuakse ja salvestatakse faili graafi tippude kiht. Geograafiliste andmete salvestamiseks GeoPackage vormingus kasutatakse vaikimisi koordinaatsüsteemi standardit World Geodetic System (WGS84), töös kasutatakse geograafiliste andmete salvestamiseks seda standardit [18,19]. Graafi kaarte kihti lisatakse järgnevad andmed:

1) kaare identifikaator;
2) kaare geograafilised koordinaadid;
3) kaare järjekorranumbrite nimekiri Euleri tees. Järjekorranumbreid võib olla mitu, sest ühte kaare võidakse läbida mitu korda;
4) ajatempel vastavalt sellele, mitmes vaadeldav kaar Euleri tees on. Ajatempel on tekitatud kunstlikult selleks, et oleks mugav Euleri teed animeerida rakenduses QGIS [20]. Esimese Euleri tee kaare ajatempliks on kuupäev 01.01.1980 kellaajaga 00:00:00. Iga Euleri tee järgneva kaare ajatempel on ühe sekundi võrra hilisem sellele eelneva kaare ajatemplist.
5) kaare pikkus meetrites;
6) kaare tüüp. Selleks saab olla kas ristmik või tänav.

Graafi tippude kihti lisatakse järgnevad andmed:

1) tipu identifikaator;
2) tipu geograafilised koordinaadid.

Peale GeoPackage faili loomist lõpetab programm töö.

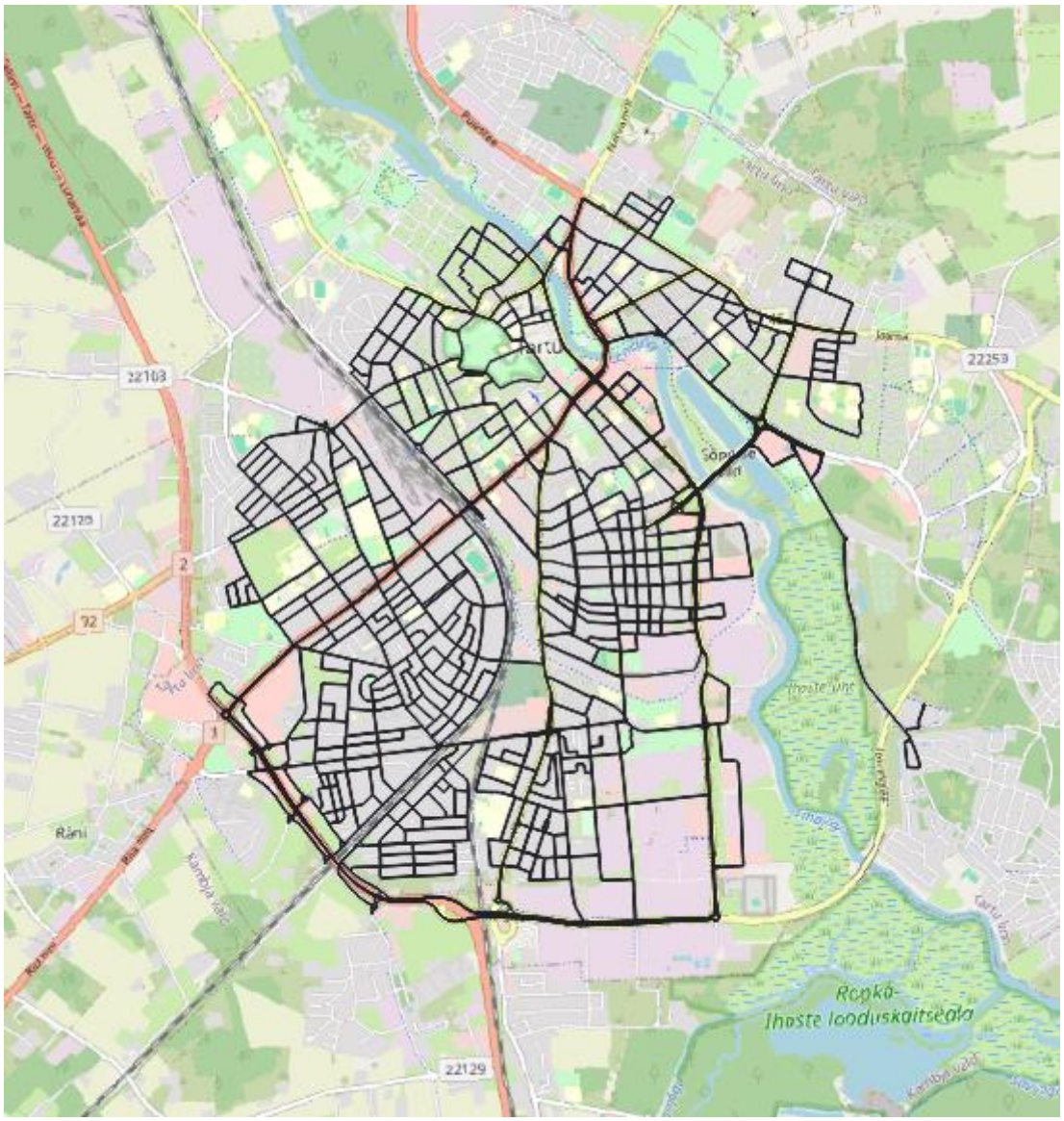
Programmi loodud GeoPackage faili on võimalik vaadata programmiga QGIS[9]. QGIS [20] on tasuta ja avatud lähtekoodiga programm, millega on võimalik visualiseerida erinevates formaatides geograafilisi andmeid, sealhulgas ka GeoPackage failina antud andmeid. Samuti on programmiga võimalik animeerida ajatempliga varustatud andmeid.

GeoPackage failis olevate andmete visualiseerimiseks ja animeerimiseks on esmalt vaja alla laadida programm QGIS. Lisas II antud animeerimise failid on testitud programmi QGIS versiooniga 3.34.12-Prizren, seega on soovitatav alla laadida programmi QGIS versioon 3.34.12-Prizren. Animeerimise faili loomise juhised lisas II olevatele failidele sarnase faili saamiseks on toodud lisas III.

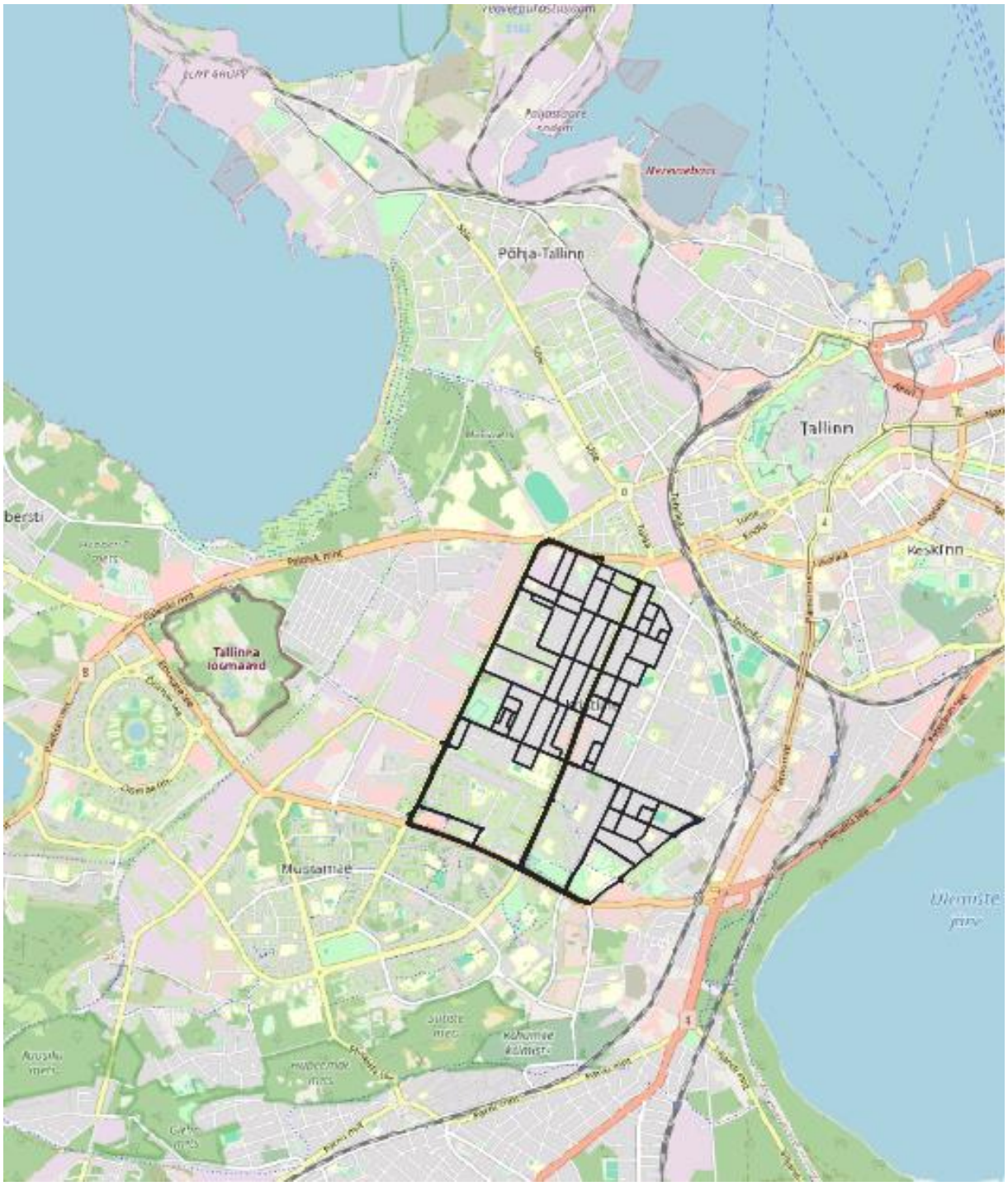
[9] Programmi QGIS allalaadimise lehekülg: https://www.qgis.org

3. Programmi rakendamise tulemused ja analüüs

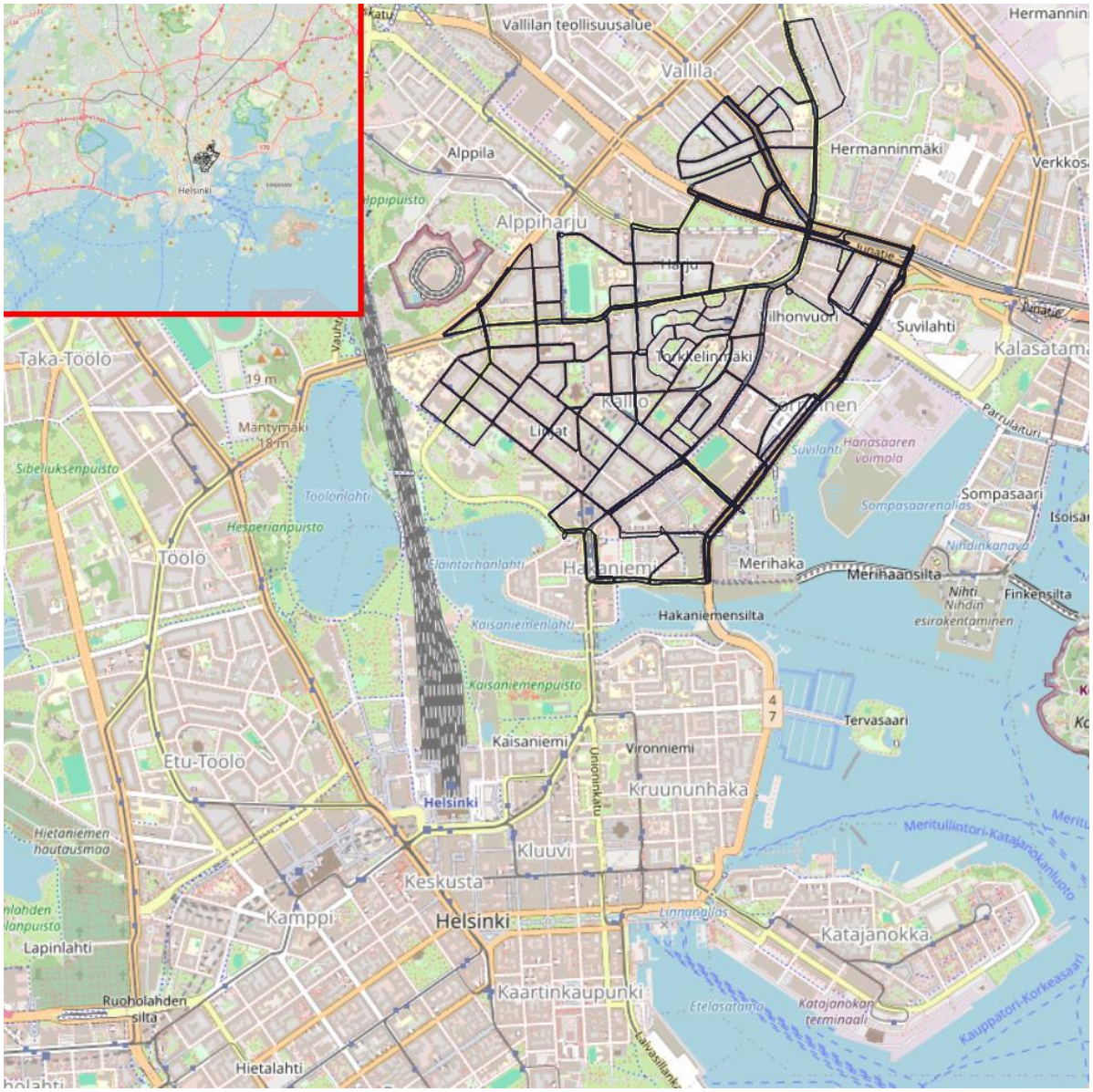
Programmi rakendati kolmel geograafilisel alal: suurem osa Tartu linnast, Tallinnas suurem osa Kristiine linnaosast ja väike osa Mustamäe linnaosast, Helsingis suurem osa Kallio linnaosast. Joonisel 2 on võimalik näha Tartu linna ala, mille peal programmi rakendati, joonisel 3 on võimalik näha Tallinna linna ala, mille peal programmi rakendati ning joonisel 4 on võimalik näha Helsingi linna ala (joonise ülemises vasakus nurgas punases kastis on väljasuumituna näha, kus programmi rakendati), mille peal programmi rakendati. Joonistel 2, 3 ja 4 tähistavad esile toodud musta värvi jooned graafi kaari, mis on vastavate linna tänavate sõiduradade põhjal loodud, taustaks on andmebaasi OSM teede kaart rakenduses QGIS. Täpsemalt on võimalik geograafilisi alasid ja nende graafe uurida lisas II toodud failidest.



Joonis 2. Osa Tartu linnast, mille peal programmi rakendati.



Joonis 3. Osa Tallinna linnast, mille peal programmi rakendati.



Joonis 4. Osa Helsingi linnast, mille peal programmi rakendati.

Tabelis 1 on toodud tulemused, mis saadi programmi vastavatel geograafilistel aladel rakendamisel. Tabeli veerud tähistavad andmeid, mida programm tagastas, ning tabeli read tähistavad geograafilisi alasid, mille peal programmi rakendati. Kaasatud sõiduradade kogupikkuse veerg tähistab nende sõiduradade pikkust, mis programmis graafi kaasati ning mille põhjal graafis Euleri tee arvutati, samuti on selle veeru andmete põhjal võimalik hinnata sisendi suurust. Euleri tee kordaja veerg tähistab seda, kui palju on vaja ühte sõidurada Euleri tee läbimiseks läbida. Kaasatud sõiduradade kogupikkuse ja Euleri tee pikkuse andmed on teisendatud meetritest kilomeetriteks ning ümardatud kümnendikeni. Täpsemaid tulemusi on võimalik vaadata lisas II olevatest failidest.

Tabel 1. Programmis rakendatud alade tulemused.

	Kaasatud sõiduradade kogupikkus (km)	Euleri tee pikkus (km)	Euleri tee kordaja	Programmi tööaeg (min)
Tartu	390,9	561,1	1,4353	2,865
Tallinn	101,0	180,7	1,7898	0,7518
Helsingi	68,8	116,2	1,6892	0,9335

Eelkirjeldatud geograafilistel aladel rakendati ka sellist programmi varianti, kus on graafi lubatud kaasata rohkem erinevaid teede tüüpe. Sellises programmi variandis kaasati graafi sellised teed, mille vastavad andmebaasis OSM sisalduvad `highway` atribuudid võib omada ka järgnevaid väärtusi [10]:

- `living_street`, mis tähistab õuealal olevaid teid;
- `service`, mis tähistab sisse- ja väljasõiduteid parklatesse, büroohoonetesse ja ostukeskustesse;
- `track`, mis tähistab selliseid teid, mis on sageli pealiskihita ning asuvad põllul, metsas või tühermaal.

Rohkemate lubatud teedega programmi rakendamiste tulemused on leitavad tabelis 2, täpsemaid tulemusi on võimalik vaadata lisas II olevatest failidest.

Tabel 2. Rohkemate lubatud teedega programmis rakendatud alade tulemused.

	Kaasatud sõiduradade kogupikkus (km)	Euleri tee pikkus (km)	Euleri tee kordaja	Programmi tööaeg (min)
Tartu	573,3	782,3	1,3646	6,9392
Tallinn	166,7	260,6	1,5633	1,6546
Helsingi	81,2	134,1	1,6504	1,2455

Tulemustest on näha, et programmi tööaeg ei kasvanud sisendi suuruse kasvades drastiliselt, millest võib järeldada, et programm töötab mõistliku ajaga ka suuremate sisendite korral.

Samuti töötab programm alla 3 minuti ka kõige suurema katsetatud sisendi (Tartu linna ala) korral.

Tulemustest selgub, et Euleri tee läbimiseks on Tartu linna alal vaja läbida ühte sõidurada keskmiselt 1,4353 korda, Tallinna linna alal on vaja ühte sõidurada läbida keskmiselt 1,7898 korda ning Helsingi linna alal on vaja ühte sõidurada läbida keskmiselt 1,6892 korda. Tulemustest on ka näha, et suurema (pikema sõiduradade kogupikkusega) vaadeldava ala korral on Euleri tee kordaja väiksem. See võib tuleneda sellest, et suurema ala korral on väiksem tõenäosus, et ala valimisel jäävad olulised regioone ühendavad tänavad valitud ala piiridest välja. Kui olulised regioone ühendavad tänavad jäävad valitud ala piiridest välja, siis võib juhtuda, et kõikide kaarte vähemalt ühe korra läbimiseks on vaja mõne kaare korral selle läbimiseks liikuda mitu korda mööda juba läbitud kaari, mis suurendab seetõttu Euleri tee kordajat.

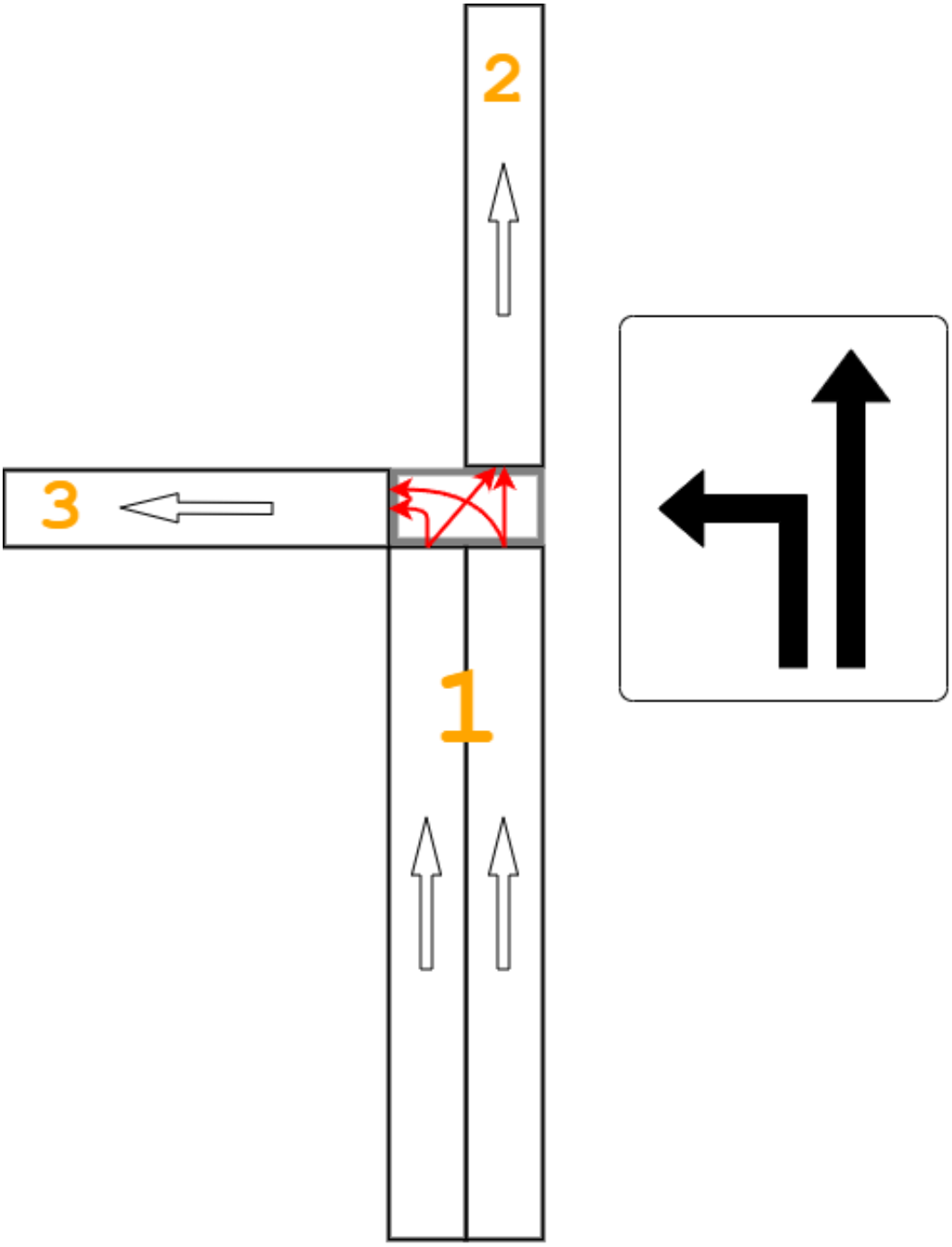
Tulemustest selgub ka, et rohkemate lubatud teedega programmis on Euleri tee kordajad väiksemad, kui vastavate geograafiliste alade tavalise programmi rakendamisel saadud Euleri tee kordajad. Tulemused kinnitavad trendi, et suurema vaadeldava ala korral on Euleri tee kordaja väiksem. Rohkemate lubatud teede korral võib väiksem Euleri tee kordaja tulla sellest, et tasakaalustamisel leidub väiksemate teede tõttu tasakaalust väljas tippude vahel lühemaid teid. Samas ei võta programm väiksemate teede kaasamisel arvesse seda, et vastavat tüüpi kaasatud teedel peab sõitma väiksema kiirusega, kui teistel kaasatud teedel. Samuti ei pruugi vastavat tüüpi teedel olla pealiskihti, mistõttu võib pikalt sellisel teel sõitmine autot kahjustada. Seetõttu ei pruugi väiksemate teede kaasamine olla tingimata kasulik otsus.

4. Edasiarendamise võimalused

Üks programmi puudus on, et vaadeldava geograafilise ala suurus on piiratud, sest veebirakenduses osm2streets on valitava geograafilise ala suurusel limiit, mida ei ole rakenduse dokumentatsioonis täpsustatud. Üks võimalus vaadeldava ala suurendamiseks on kaasa aidata veebirakenduse osm2streets Pythoni rakendusliidese valmimisele (töö tegemise hetkel ei olnud veebirakenduse osm2streets Pythoni liides veel valminud) [7], kus valitava geograafilise ala suurus ei ole piiratud. Teine võimalus on jupitada uuritav geograafiline ala väiksemateks osadeks ehk valida veebirakenduses osm2streets vaadeldava geograafilise ala osa ja alla laadida vastava osa kolm sisendfaili. Protsessi tuleks korrata seni, kuni kogu uuritav geograafiline ala on kaetud. Seejärel saab luua programmi, mis seoks mitme geograafilise ala sisendfailid üheks tervikuks ehk seob geograafilised alad tervikuks, mille kohta luuakse mitme geograafilise ala sisendfailide põhjal kolm sisendfaili. Kolmas võimalus on ise luua veebirakendusele osm2streets sarnane, kuid kitsama võimekusega, rakendus, mis leiab andmebaasi OSM andmetest vaadeldava geograafilise ala jaoks vajalikud andmed sõiduradade kohta ning loob leitud andmete põhjal programmi jaoks vajalikud sisendfailid.

Teine programmi puudus on, et sisendfailide põhjal koostatud graafis ei pruugi defineeritud ristmikel liikumised reaalsetel tänavatel olla lubatud. Näide võimalikust olukorrast on näha joonisel 5, kus hall kast tähistab ristmikku, liiklusmärk tähistab reaalselt lubatud liikumist, halli kasti sees olevad punased nooled tähistavad teekondi, kuhu saab programmi andmetel sisendandmete põhjal liikuda, ning oranžid numbrid tähistavad tänavate identifikaatoreid. Geojson failides sisalduks hüpoteetiliselt joonisel 5 olevate tänavate ja ristmiku kohta info, et tänavalt 1 võib liikuda tänavatele 2 ja 3, kuid tegelikult võib ainult vasakpoolselt tänava 1 sõidurajalt liikuda tänavale 2 ning ainult parempoolselt tänava 1 sõidurajalt liikuda tänavale 3. Olukord tekib seetõttu, et geojson failid, mille alusel ristmiku sees lubatud liikumised paika pannakse, sisaldab andmeid tänavate, kuid mitte sõiduraja tasandil.

Üks võimalus probleemi lahendada on luua programmis võimekus kasutada graafi koostamisel failis „osm.xml“ leiduvaid `relation` atribuute, mis on märgendiga `restriction`. Sellised atribuudid määravad sõidutee tasemel ristmikel liiklemise piirangud [21].



Joonis 5. Hüpoteetiline olukord, kus programmis lubatud liikumised ei vasta tegelikkusele.

Teine võimalus probleemi lahendada on kaasa aidata veebirakenduse osm2streets arendamisel geojson failides sisalduva info täiendamises. Failis „Lane polygons.geojson“ on atribuut `allowed_turns`, kus peaksid sisalduma reeglid, kuidas antud sõidurajalt on võimalik liikuda [7]. Töö kirjutamise hetkel ei olnud seda funktsionaalsust rakenduses osm2streets veel välja arendatud.

Programmi puudus on ka see, et kaarte pikkusi arvutatakse linnulennult algtipust lõpptippu, kuid tegelikult ei pruugi kaared olla sirged. Kuna kaarte pikkused arvutatakse eelnevalt kirjeldatud meetodil nii graafi kaarte kogupikkuse arvutamisel kui ka graafi Euleri tee pikkuse arvutamisel, siis mõjutab see Euleri tee kordajat vaid vähesel määral. Probleem on võimalik lahendada, kui töötada välja üldine algoritm, mis loob kaare hulknurgale hulknurga keskosa läbiva lõigu ning arvutab seejärel lõigu pikkuse.

Kokkuvõte

Bakalaureusetöö eesmärk oli luua programm, mis läbiks etteantud geograafilise ala kõik sõidurajad võimalikult lühikese vahemaaga. Samuti oli töö eesmärk rakendada programmi Tartu linnal, Tallinna linnal ja muudel linnadel. Loodud programmi rakendati suuremal osal Tartu linnast, osal Tallinna linnast ning osal Helsingi linnast. Programm leidis igal geograafilisel alal optimaalsed teekonnad kõikide sõiduradade läbimiseks, iga ala korral kulus programmil aega teekondade leidmiseks alla 3 minuti.

Samas on programm geograafiliste alade valimisel sõltuv veebirakendusest osm2streets [7], millel on mitmeid puudusi. Üks puudus on, et veebirakenduses on valitava geograafilise ala suurus on piiratud, mistõttu ei olnud võimalik programmi rakendada suurematel aladel. Teine puudus on, et veebirakendusest valitud geograafiliste alade failides ei pruugi defineeritud ristmikel liikumised reaalsetel tänavatel olla lubatud, mistõttu võib programmi poolt tagastatud teekondades esineda selliseid ristmikel liikumisi, mis ei ole tegelikult lubatud. Kuna veebirakendus on tasuta ja avatud lähtekoodiga, siis on võimalik nimetatud puudused kõrvaldada, kui aidata arendada veebirakendust osm2streets [7] ja muuta programmi lähtekoodi.

Viidatud kirjandus

[1] Poggenhans, F., Pauls, J.-H., Janosovits, J., Orf, S., Naumann, M., Kuhnt, F., Mayr, M. (2018). Lanelet2: A high-definition map framework for the future of automated driving. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI: IEEE. pp. 1672–1679. doi: 10.1109/ITSC.2018.8569929.

[2] OpenAI (2022). ChatGPT (GPT4-o): https://chatgpt.com.

[3] Buldas, A., Laud, P., Villemson, J. Graafid. Tartu: Tartu Ülikool Matemaatika-Informaatikateaduskond Arvutiteaduse Instituut. 2003.

[4] Dharwadker, A., Pirzada, S. Graph Theory. India, Gurgaon: Ashay Dharwadker Institute of Mathematics. 2011.

[5] Crouse, David F. (2016). On implementing 2D rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*. vol. 52, no. 4, pp. 1679–1696. doi: 10.1109/TAES.2016.140952.

[6] linear_sum_assignment. (2024). SciPy v1.14.1 Manual. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html (22.12.2024).

[7] Osm2streets. GitHub. https://github.com/a-b-street/osm2streets (31.12.2024).

[8] OpenStreetMap. https://www.openstreetmap.org/about (19.12.2024).

[9] Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., Hagen, S. (2016). The GeoJSON Format. *Internet Engineering Task Force*. doi: 10.17487/RFC7946.

[10] Key:highway. (2024). OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Key:highway (19.12.2024).

[11] NetworkX. (2024). NetworkX documentation. https://networkx.org/ (19.12.2024).

[12] strongly_connected_components. (2024). NetworkX documentation. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.strongly_connected_components.html (30.12.2024).

[13] Nuutila, S., Soisalon-Soininen, E. (1994). On finding the strongly connected components in a directed graph. *Information Processing Letters*. vol. 49, no. 1, pp. 9-14. doi: 10.1016/0020-0190(94)90047-7.

[14] single_source_dijkstra. (2024). NetworkX documentation. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.single_source_dijkstra.html (30.12.2024).

[15] Cherkassky, Boris V., Goldberg, Andrew V., Radzik, T. (1996). Shortest paths algorithms: *Theory and experimental evaluation. Mathematical Programming*. vol. 73, no. 2, pp. 129-174. doi: 10.1007/BF02592101.

[16] eulerian_path. (2024). NetworkX documentation. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.euler.eulerian_path.html (30.12.2024).

[17] GeoPackage. (2024). OGC GeoPackage. https://www.geopackage.org/ (22.12.2024).

[18] World Geodetic System (WGS84). (2015). GISGeography. https://gisgeography.com/wgs84-world-geodetic-system/ (07.01.2025).

[19] GeoPackage Implementation Guide. (2022). https://www.geopackage.org/guide/implementation_guide.html (07.01.2025).

[20] QGIS overview. (2024). QGIS Web Site. https://www.qgis.org/project/overview/ (30.12.2024).

[21] Relation:restriction. (2024). OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Relation:restriction (01.01.2025).

Lisad

I. Link ja juhend lähtekoodile

Programmi rakendamiseks on esmalt on vaja alla laadida Python, mis oleks vähemalt versioon 3.13, samuti on vaja alla laadida käsureapõhine teekide haldamise tarkvara pip[10]. Vajalik on ka tarkvara git olemasolu. Peale tarkvara git alla laadimist on vaja programmi rakendamiseks täita järgnevad sammud:

1) ava käsuviip;
2) lae alla programmi lähtekood. Kasuta selleks käsku `git clone <programmi lähtekoodi link>`, link programmi lähtekoodile on antud lisas I;
3) liigu käsuviibas alla laetud lähtekoodi kausta. Kasuta selleks käsku `cd find-streets-eulerian-path`;
4) programmi tavaversiooni haru valimiseks kasuta käsku `git checkout main`. Programmi sellise versiooni haru valimiseks, kus on teede graafi kaasatud rohkem sõiduradu (selgitatud lähemalt peatükis 3), kasuta käsku `git checkout extra-roads`;
5) lae alla vajalikud teegid, mis on programmi käivitamiseks vajalikud. Kasuta selleks käsku `pip install -r requirements.txt`;
6) lae veebirakenduse osm2streets kaudu alla uuritavale geograafilisele alale vastavat kolm sisendfaili, mis on toodud programmi sisendi peatükis;
7) liiguta sisendfailid asukohta, mis on määratud failis „config.py“. Faili „Lane polygons.geojson“ asukoha määrab muutuja `LANES_GEOJSON_FILE_PATH`, faili „Intersection polygons.geojson“ asukoha määrab muutuja `INTERSECTIONS_GEOJSON_FILE_PATH`, faili „osm.xml“ asukoha määrab muutuja `OSM_XML_FILE_PATH`;
8) käivita programm käsuga `python main.py`. Väljundfailid luuakse kohta, mis on defineeritud failis „config.py“. Väljundfail „output.gpkg“ luuake kohta, mille määrab muutuja `OUTPUT_GPKG_FILE_PATH`. Väljundfail „results.txt“ luuakse kohta, mille määrab muutuja `OUTPUT_TXT_FILE_PATH`.

[10] pip: https://pip.pypa.io/en/stable/

Link lähtekoodile: https://github.com/raunoraa/find-streets-eulerian-path

II. Link ja juhend rakendatud geograafiliste alade failidele

Selles lisas on toodud järgnevad failid iga geograafilise ala kohta, mille peal programmi rakendati:

1) vastava geograafilise ala sisendfailid. Need failid sisalduvad vastava geograafilise ala nimega kaustas asuvas kaustas nimega „Input Files“;
2) vastava geograafilise ala väljundfailid, mis saadi programmi vastavatel sisendfailidel rakendamisel. Need failid sisalduvad vastava geograafilise ala nimega kaustas asuvas kaustas nimega „Output files and qgz file“;
3) fail, mis on .qgz vorminguga. See fail sisaldub vastava geograafilise ala nimega kaustas asuvas kaustas nimega „Output files and qgz file“.

Lisaks on iga ala kohta toodud ka sellise programmi väljund, kus on kaasatud rohkem teid (täpsemalt kirjeldatakse sellist programmi peatükis 3). Vastavad failid on kaustas „Results with Extra Roads“, mis asub kaustas „Output files and qgz file“, mis asub vastava geograafilise ala kaustas.

Failid on soovitatav alla laadida ühe kokkupakitud failina. Kokkupakitud fail tuleks selles sisalduvate failide kasutamiseks enne lahti pakkida. Fail, mis on .qgz vorminguga, on programmis QGIS avamiseks ja kasutamiseks mõeldud fail, mis sisaldab lisas III kirjeldatud seadistusi, mis on loodud vastava GeoPackage faili jaoks.

Animeeritud teekonna vaatamiseks programmis QGIS avatud .qgz vorminguga failis on vaja valida „Layer“ > „Panel“ > „Temporal Controller“. Selle tulemusel tuleb projektis nähtavale animeerimise seadistamise paneel. Animeerimise väljalülitamiseks ja kõikide kaarte ja tippude korraga projektis kuvamiseks tuleb animeerimise seadistamise paneelis vajutada vasakpoolset nuppu. Animeerimise sisselülitamiseks tuleb esmalt vajutada parempoolset nuppu. Seejärel tuleb „Animation range“ nimelises reas vajutada „to“ nimelisest lahtrist paremal asuvat nuppu. Animeerimise sammu pikkuseks tuleb määrata üks sekund. Selle jaoks tuleb valida tekstist „Step“ paremal asuvas rippmenüüs valida väärtus „seconds“. Peale seda on võimalik Euleri teed animeerituna vaadata. Animatsiooni on võimalik peatada, edasi mängida, tagasi kerida ja edasi kerida animeerimise seadistamise paneelis olevate nuppudega. Animatsiooni saab kiirendada või aeglustada, muutes animatsiooni kaadrisagedust. Selle jaoks tuleb esmalt vajutada paneeli „Temporal Controller“ ülemises paremas nurgas asuvat hammasratast, seejärel saab muuta kaadrisagedust kirjutades lahtrisse „Frame rate (frames per second)“ sobiva

arvulise väärtuse. Suurem kaadrisagedus muudab animatsiooni kiiremaks, väiksem kaadrisagedus aeglustab animatsiooni.

Graafist parema ülevaate saamiseks võib välja või sisse lülitada tagataustal asuvat ortofotot või teede kaarti. Ortofotot või teede kaarti saab sisse lülitada, kui teha linnuke vastavalt kihi „Ortofoto“ või kihi „OpenStreetMap“ ette. Ortofotot või teede kaarti saab välja lülitada, kui vastavalt kihi „Ortofoto“ või kihi „OpenStreetMap“ eest eemaldada linnuke.

Link failidele:
https://drive.google.com/drive/folders/1iLgaCyLz092MU7t6YOZGydiwEh6wuK1s?usp=sharing

III. Animeerimise faili loomise juhised

GeoPackage failis sisalduva graafi ja Euleri tee animeerimiseks ja visualiseerimiseks tuleb täita järgmised sammud:

1) loo programmis QGIS uus projekt, valides „Project“ > „New“;
2) lisa projekti vaadeldava geograafilise ala ortofoto kiht või lisa projekti vaadeldava geograafilise ala tänavate kaardi kiht. See on kasulik, sest selle tulemusel on programmis QGIS võimalik seostada uuritavat ala päris maailmaga. Kui vaadeldav geograafiline ala on Eestis, siis Eesti ortofoto kihti saab tasuta Maa-ameti veebilehelt[11] lisada järgnevate sammudega:
    a) vali „Layer“ > „Add Layer“, mis avab uue kihi lisamise akna;
    b) vali lahti tulnud aknas „New“, mis avab kihi veebiallika lisamise akna;
    c) kirjuta lahtrisse „Name“ järgnev: „Maa-ameti aluskaart“;
    d) kirjuta lahtrisse „URL“ järgnev: „https://kaart.maaamet.ee/wms/alus“;
    e) vali veebiallika lisamise aknas „OK“;
    f) vali kihi lisamise aknas rippmenüüst „Maa-ameti aluskaart“;
    g) vali „Connect“, mis loob kihi lisamise aknasse Maa-ameti kaartide nimekirja;
    h) vali kaartide nimekirjast „Maa-ameti aluskaardid“ > „Ortofotod“ > „Ortofoto“;
    i) vali kihi lisamise aknas „Add“. Sellega lisatakse projekti ortofoto kiht nimega „Ortofoto“.

    Kui vaadeldav geograafiline ala asub Eestist väljas, siis on võimalik projekti lisada andmebaasi OpenStreetMap teede kaart, mis on juba vaikimisi programmis QGIS olemas. Selleks on vaja täita järgnevad sammud:
    a) vali „Layer“ > „Data Source Manager“;
    b) vali „XYZ Connections“ nimelise rippmenüü alt väärtus „OpenStreetMap“;
    c) vali „Add“. Sellega lisatakse projekti andmebaasi OpenStreetMap teede kaardi kiht nimega „OpenStreetMap“.

    Soovi korral võib lisada andmebaasi OpenStreetMap teede kaardi ka Eesti geograafilise ala projekti, kuid teede kaardi taustale lisamiseks tuleb eelnevalt kas eemaldada linnuke kihi „Ortofoto“ eest või liigutada kiht „OpenStreetMap“ kihist „Ortofoto“ ülespoole;
3) lohista GeoPackage fail projektis lahtrisse „Layers“. Lahti tulnud aknas vali „OK“. Selle sammuga ilmuvad projektis nähtavale graafi tipud (nodes) ja kaared (edges), mis kuuluvad kihti „output“. Juhul, kui tipud ja kaared nähtaval ei ole, tuleb kiht „output“

[11] Maa-ameti veebileht: https://geoportaal.maaamet.ee/est/

liigutada „Ortofoto“ ja/või „OpenStreetMap“ kihist ülespoole. Seda saab teha, kui valida kiht „output“ ning lohistada see kihist „Ortofoto“ ja/või „OpenStreetMap“ ülespoole. Juhul, kui peale seda ei ole tippe näha, tuleks kihi „output“ sees lohistada samal viisil tippude kiht ülespoole kaarte kihist;

4) vajuta paremat hiireklahvi kaarte kihi peal ning vali „Properties“. Lahti tulnud aknas vali „Symbology“;
5) vali lahti tulnud akna kõige ülemises rippmenüüs „Categorized“. Kirjuta „Value“ lahtrisse järgnev tekst: `CASE WHEN "order_nr" = @frame_number THEN 'white' ELSE 'blue' END`

   Peale teksti sisestamist vali „Classify“. Selle tulemusel tekib sümbolite sektsiooni kaks kirjet: ülemine kirje tähistab animeerimisel Euleri tee läbitud kaari, alumine kirje tähistab just läbitud kaare. Mõlema kirje puhul on võimalik muuta kaare sümbolit, kaare värvi ja kaare paksust. Soovitatav on mõlema kirje puhul muuta ainult kaarte värve järgmiselt: ülemise kirje värviks määrata sinine ja alumise kirje värviks määrata valge, et oleks võimalik vastavaid kaari kergesti eristada. Sinise värvi määramiseks ülemisele kirjele vajuta ülemise kirje peal paremat hiireklahvi, seejärel vali „Change Color…“, seejärel kirjuta avanenud aknas lahtrisse „R“ väärtus 0, lahtrisse „G“ väärtus 0, lahtrisse „B“ väärtus 255 ning vali „OK“. Valge värvi määramine alumisele kirjele käib sarnaselt sinise värvi määramisele, erinevusteks on, et lahtri „R“ väärtuseks tuleb kirjutada 255, lahtri „G“ väärtuseks tuleb kirjutada 255 ning lahtri „B“ väärtuseks tuleb kirjutada 255. See samm on vajalik selleks, et animeerimisel oleks selgelt näha, missugused kaared on varem läbitud ning missugust kaare antud animatsiooni sammu juures läbitakse;
6) pane linnuke teksti ‘„Control feature rendering order“ ette. Seejärel vajuta samast tekstist paremal olevale nupule. Avanenud aknas vali vasakul asuvast rippmenüüst väärtus „order_nr“. Seejärel vali keskel asuvast rippmenüüst väärtus „Ascending“. Seejärel vali „OK“. See samm on vajalik selleks, et animeerimisel jääksid hiljem läbitud kaared paremini nähtavale;
7) vali avatud aknas „Temporal“;
8) tee linnuke teksti „Dynamic Temporal Control“ ette;
9) vali „Configuration“ nimelises rippmenüüs väärtus „Single Field with Date/Time“;
10) vali „Limits“ nimelises rippmenüüs väärtus „Include Start, Include End“;
11) vali „Field“ nimelises rippmenüüs väärtus „order“;
12) tee linnuke teksti „Accumulate features over time“ ette;

13) vajuta „Apply“, seejärel vajuta „OK“. Selle sammuga salvestatakse muudatused ning suletakse aken;
14) lisa projekti taustale suurelt info, mitmenda animatsiooni sammu juures ollakse. Selleks on vaja täita järgnevad sammud:
    a) vali projektis „View“ > „Decorations“ > „Title Label…“;
    b) tee avanenud aknas linnuke teksti „Enable Title Label“ ette;
    c) kirjuta lahtrisse „Title label text“ järgnev tekst:
    ```
    Current animation step: [%@frame_number+1%]
    ```
    d) vali „Placement“ nimelises rippmenüüs väärtus „Bottom Center“;
    e) vajuta „Apply“ ja seejärel vajuta „OK“.

IV. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Rauno Raa**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Sõiduradu läbiva programmi loomine ja rakendamine**, mille juhendajad on **Tambet Matiisen**, **Edgar Sepp** ja **Karl-Johan Pilve**, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni;

2. annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni;

3. olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile;

4. kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Rauno Raa

09.01.2025