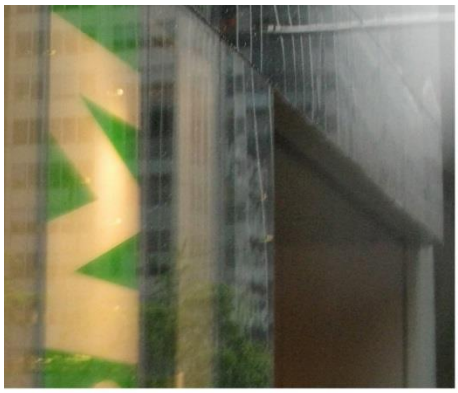
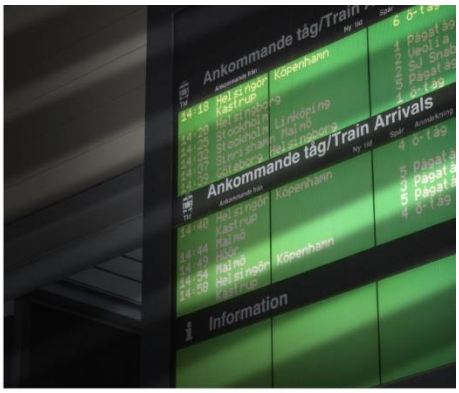
Trivector

Trivector Traffic

Rapport 2020:117, Version 1.0

RVU Nätutläggning







Dokumentinformation

Titel:	RVU Nätutläggning
Serie nr:	2020:117
Projektnr:	19247
Författare:	Joakim Fors
Medverkande:	
Kvalitetsgranskning:	Emeli Adell
Beställare:	Region Skåne Kontaktperson: Jenny Rassmus tel +46 (0)40-675 34 67

Dokumenthistorik:

Version	Datum	Förändring	Distribution
1.0	2020-09-30		Beställare

Trivector Traffic · Vävaregatan 21 · SE-222 36 Lund / Sweden
Telefon +46 (0)10-456 56 00 · info@trivector.se

Innehållsförteckning

Trivector Traffic · Vävaregatan 21 · SE-222 36 Lund / Sweden
Telefon +46 (0)10-456 56 00 · info@trivector.se

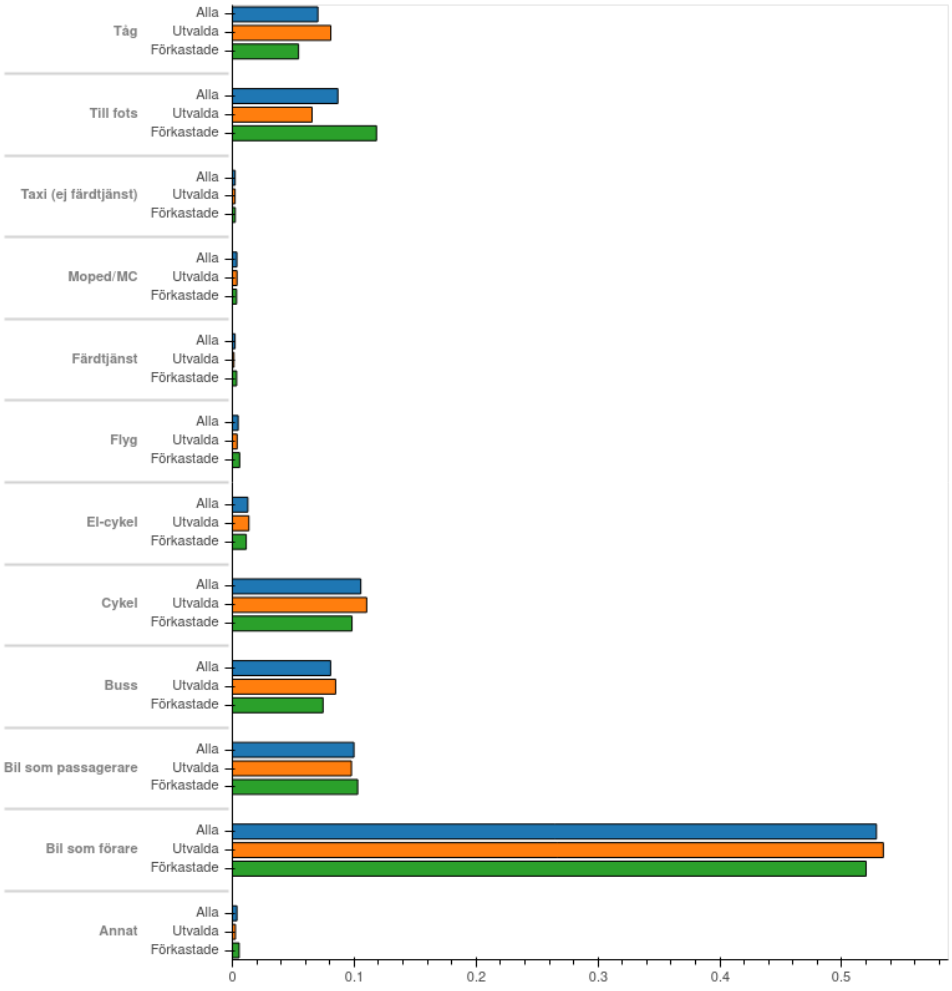
1. Introduktion

Traditionella resvaneundersökningar (RVU) baserar sig på att personer fyller i en enkät eller intervjuas. Region Skåne genomförde 2018 en stor RVU där information om över 75000 resor samlades in. Resdatan består av information om utgångspunkt, destination, tidpunkt, ärende, färdmedel samt självskattad restid. Syftet med nätutläggningen är att omvandla denna resdata till geografisk data som går att analysera och visualisera.

2. Begränsingar i RVU-data

Positionsdata som ingår i RVU:n baserar sig på adresser som de tillfrågade själva har skrivit in. Geokodningen har inte varit 100% säker och därför saknas koordinater för en del resor. Dessutom är vissa resor cirkulära, dvs har samma utgångspunkt som målpunkt, och kan inte användas för ruttning. Ett urval har gjorts och enbart de resor som har både start- och målkoordinater har använts för nätutläggning.

Efter sållning kan man se att färdmedelsfördelningen hålls relativt konstant men att det är främst resor till fots som sorterats bort. Av totalt 75373 resor i ursprungsdatan är det 44507 som använts för nätutläggning och 30866 som sorterats bort varav 7299 cirkulära.



Färdmedelsfördelning i RVUn

Färdmedel	Alla %	Utvalda %	Förkastade %
Annat	0.37	0.25	0.55
Bil_som_förare	52.85	53.41	52.00
Bil som passagerare	9.97	9.77	10.28
Buss	8.06	8.47	7.44
Cykel	10.54	11.02	9.81
El-cykel	1.27	1.35	1.14
Flyg	0.47	0.39	0.60
Färdtjänst	0.22	0.13	0.35
Moped/MC	0.37	0.38	0.35
Taxi (ej färdtjänst)	0.22	0.22	0.24
Till fots	8.65	6.54	11.82
Tåg	7.01	8.07	5.43

3. Genomförande

För att konvertera resdatan som i ursprungligt format enbart består av ett par koordinater per resa behöver den köras mot en multimodal ruttningsmotor. Exempel på ruttmotorer i dagligt bruk är t.ex. Google Maps och Apple Maps. För detta ändamål behövs däremot ett verktyg som tillåter konfiguration av både valet av kartdata och kollektivtrafiks-tidtabeller. Det finns ett antal ruttmotorer som är öppen källkod som tillåter multimodal ruttplanering däribland OpenTripPlanner (OTP), Valhalla, och Graphhopper. Den mest etablerade av föregående alternativ är OTP och används av bl.a. flera kollektivtrafikorganisationer t.ex. Helsingforsregionens trafik (Halsingfors, Finland), Ruter (Oslo, Norge), TriMet (Portland, Oregon, USA), m.fl.. Valet har därför blivit att använda den i det här projektet.

Som vägnätverk för ruttgenereringen har Nationell vägdatabas (NVDB) önskats användas. NVDB betraktas som en av samhällets grunddatabaser och omfattar ett referensvägnät och en stor mängd data kopplade till vägnätet. Det innebär att den innehåller alla Sveriges bilvägar, gator, cykelvägar samt en delmängd av gångvägarna. NVDB är ett samarbete mellan Trafikverket, Sveriges kommuner och regioner, skogsnäringen, Transportstyrelsen och Lantmäteriet. Trafikverket är huvudman för NVDB. Datan uppdateras kontinuerligt och är av jämngod kvalitet över hela landet.

OTP är utvecklad för att använda OpenStreetMap (OSM) data som vägnätverk. Således har en del av projektet gått ut på att konvertera NVDB data till OSM-format.

3.1 NVDB konvertering

I NVDB representeras varje vägattribut av ett eget kartlager där varje objekt refererar till unikt identifierbara vägsträckor i ett referensnät. Attributen innehåller även information om vilken utbredning attributet har på respektive länk i referensnätet. Utbredningen beskrivs som normaliserad distans från startpunkten av länken. OSM har en något annorlunda datastruktur där ett flertal attribut appliceras på samma geometriska objekt och objekten i sin tur kan grupperas i relationer som i sin tur kan ha attribut. För att konvertera NVDB-data till OSM behöver man således identifiera alla utbredningspunkter per länk i referensnätet för att sedan klippa sönder den i sina minsta beståndsdelar och sedan applicera samtliga relevanta attribut på dessa. En liknande funktion finns visserligen redan i Lastkajen[1] men den stödjer inte alla datalager och attribut.

Bearbetning

För att genomföra konverteringen har en samling verktyg utvecklats i Python och Bash. Verktygen kan hittas på https://github.com/cyklaiskane/nvdb-tools och https://github.com/cyklaiskane/pgr2osm.

[1]Lastkajen är Trafikverkets dataportal där geodataprodukter kan laddas ner.

Indata för bearbetningen är en länsfil i GeoDB-format som finns som färdig dataprodukt på Lastkajen. Skripten i `nvdb-tools/import` används för att både importera alla lager från länsfilen till en PostGIS-databas och skapa ett homogeniserat dataset med attribut som är förberedda för att kunna översättas till OSM-kompatibla nyckel/värde-par. Det homogeniserade datasetet kan sedan exporteras till OSM-format med verktyget i `pgr2osm`.

Importen och homogeniseringen är uppdelad i flera steg för att underlätta felsökning och modifiering. Det första steget, som sköts av `import_stage0.sh`, är att helt enkelt kopiera över alla lager från GeoDB-filen till PostGIS-databasen och skapa index för rutt-id. Efterföljande steg kan antingen utföras manuellt genom att köra de SQL-filer som finns i `import` i tur och ordningen eller automatiskt genom att exekvera `import.sh`.

Den första SQL-filen skapar en tabell i PostGIS med kolumner för utvalda attribut som används i senare steg. Tabellen fylls sedan med geometrier från referensnätet.

Steg 2 sköter om att dela upp referensnätet i minsta nödvändiga geografiska beståndsdelar på basen av vilka attributlager som kommer att ingå. Detta verkställs genom att analysera vilka lager som används i steg 3 för att sedan med hjälp av en mall generera SQL-kod som söker efter utbredningspunkter som används för att eventuellt dela och ersätta geometrierna som skapades i steg 1.

Skriptet i steg 3 utför översättning av utvalda NVDB-attribut till OSM-liknande värden. Flera attribut kan relativt enkelt läggas till genom att modifiera `nvdb_import_stage3.sql`. Översättningen är inte helt komplett utan fungerar som ett förberedande steg där `pgr2osm` sköter den slutgiltiga översättningen.

Det avslutande steget använder sig av pgRouting-tillägget till PostGIS för att bygga en tabell med noder och kanter. Denna information används av `pgr2osm` för att bygga sin nätverksgraf.

OTP kan som tidigare nämnts enbart använda OSM-data för att bygga sin interna ruttgraf. Därför behövs verktyget `pgr2osm` för att konvertera geometrier, attribut, och id:n från PostGIS databasen till OSM XML-format. Scriptet är utvecklat i Python och gör asynkrona förfrågningar mot PostGIS-databsen samt skriver ut konverterad data. Enkla attributöversättningar som redan gjorts i föregående import-steg kopieras över direkt. Mer komplicerade attribut som förbud kräver extra behandling då datamodellerna skiljer sig mer åt mellan NVDB och OSM. Scriptet använder uppslagstabeller för att översätta de fordonstyper och restriktioner/förbud som finns i definierade i NVDB till ekvivalenta taggar i OSM och bygger därefter de korrekta nyckel/värde kombinationerna för de berörda vägsegmenten.

Mjukvara

Följande mjukvara krävs för att kunna utföra alla arbetssteg:

Docker

> Mjukvara för att skapa och köra så kallade *containers*.
> Docker tillåter en att distribuera och köra färdigpaketerad mjukvara på ett smidigt sätt. Mjukvaran distribueras i s.k. *images* som innehåller nödvändig programvara och bibliotek. Dessa *images* kan sedan enkelt repeterbart instansieras när det behövs. I och med att mjukvaran är självstående påverkas den inte av vad som är installerat på

användarens dator. Det är därmed möjligt att återskapa samma mjukvarumiljö oavsett var den körs.
I detta projekt används Docker för att köra både PostGIS och OTP då dessa annars kräver att en stor mängd annan mjukvara och bibliotek installeras.

git

Distribuerat versionshanteringssystem.
Git används främst för att versionshantera källkod vid mjukvaruutveckling. Dess distribuerade natur möjliggör enkel spridning av källkoden och att flera utvecklare kan modifiera och dela kod med varandra. Eftersom allt är versionhanterat är det lätt att plocka ut en specifik version av källkoden för att t.ex. återskapa historiska resultat.
Källkoden som skapats i det här projektet versionshanteras i Git och distribueras främst via GitHub.

Python 3.7+

Programmeringsspråk.
Python är ett populärt programmeringsspråk inom databehandling. Det finns en stor samling bibliotek som tillåter kraftfull och effektiv analys av stora dataset. Python är ett så kallat tolkat språk som tillåter att samma kod körs på olika operativsystem och processorarkitekturer.

bash

Kommandotolk och skriptspråk.
Används i huvudsak till att automatisera kommandon. Kan även användas som ett programmaringsspråk. I projektet används bash-skript för att automatiskt utföra en del arbetsmoment.

sed

Textbehandlingsverktyg.
Används för att automatiskt modifiera och ersätta text i bash-skript.

ogr2ogr/ogrinfo (GDAL)

Verktyg för att behandla och konvertera geografisk data och format.
GDAL är ett mjukvarubibliotek som används för att både manipulera geografisk data på ett effektivt sätt samt att konvertera mellan olika koordinatsystem och/eller lagringsformat.
I projektet används GDAL-baserade verktyg för att importera och exportera data.

Windows

För att utföra alla arbetsmoment under Windows kan det krävas lite extra förberedelser. I Windows 10 inkluderas bash och sed men det kan krävas att man först slår igång Windows Subsystem for Linux (WSL). Om WSL inte redan är installerat kan man gå via `Settings > Update & Security > For Developers` och slå på `Developer mode`. Gå sedan till `Control Panel > Programs > Programs and Features` och kryssa i `Windows Subsystem for Linux`.

Utförande

Första steget är att s.k. klona verktygen till den egna datorn. Börja med att skapa en samlingskatalog där alla program och datafiler kan sparas. Starta sedan en kommandotolk (Terminal i macOS/Linux och PowerShell i Windows) och navigera till samlingskatalogen. Kör därefter:

```
git clone https://github.com/cyklaiskane/nvdb-tools.git
git clone https://github.com/cyklaiskane/pgr2osm.git
```

Databasbehovet täcks av PostgreSQL/PostGIS med pgRouting-tillägg som kan initialiseras genom att i en kommandotolk köra:

```
docker run --name postgis -e POSTGRES_PASSWORD=mysecretpassword -d
pgrouting/pgrouting:12-3.0-3.0.0
```

Ladda sedan ner vägdata för Skåne från Lastkajen via `Fillager -> Vägdata -> Län File GeoDb -> 12_Skåne_län.zip` spara filen, packa upp den, och kopiera `12_Skåne_län.gdb` till `nvdb-tools/import/` i samlingskatalogen. Gå därefter till samma katalog i kommandotolken och kör:

```
./import_stage0.sh
./import.sh
```

Databasen innehåller nu ett komplett vägnät för Skåne med alla nödvändiga attributer för att kunna generera en OSM-fil. För att exportera databasen navigerar man till `pgr2osm` katalogen i samlingskatalogen och kör därefter:

```
python3 pgr2osm.py > result.osm
```

3.2 Nätutläggning med OpenTripPlanner

Resvaneundersökningen pågick under oktober 2018 vilket gör att GTFS data från den perioden behövs. Trafiklab sammanställer nationella GTFS filer och tillhandahåller historisk data. Oktober månads datafil kan hämtas på följande adress: https://data.samtrafiken.se/trafiklab/gtfs-sverige-2/2018/10/

Förbereda och starta OTP

Innan en OTP instans kan startas behöver rutningsdatabasen förberedas. Börja med att skapa följande kataloghierarki i samlingskatalogen: `data/graphs/nvdb`. Kopiera in `.osm` filen från föregående steg samt GTFS `.zip` filen i den nyss skapade `nvdb` katalogen. Kör därefter följande kommando i samlingskatalogen för att bygga en ruttgraf:

```
docker run \
  --rm \
  -e JAVA_MX=4G \
  -v "$(pwd)/data":/data:rw \
  trivectortraffic/opentripplanner:rsrvu \
    --basePath /data \
    --build /data/graphs/nvdb
```

Ovanstående kommand startar en temporär *docker container* med *imagen* `trivectortraffic/opentripplanner` som kör OTP i grafbyggarläge (`--build`). I detta läge analyserar OTP vilka datafiler som finns i den angivna katalogen (`/data/graphs/nvdb`). Grafobjektet som skapas innheåller både ett optimerat vägnätverk samt GTFS-data från zip-filen. Grafobjektet används sedan för ruttuppslagningar när OTP startas i server-läge. Grafgenerering

behöver enbart utföras när antingen vägnätverket eller GTFS-datan ändras. Det går även att ha flera grafobjekt som innehåller olika data genom att skapa en ny katalog med data. Vilket grafobjekt eller `router` som ska användas kan väljas vid varje anrop och en standardrouter kan väljas vid uppstart av OTP serverinstansen.

När ruttgrafen har skapats kan en OTP instans startas med följande kommando:

```
docker run \
  --rm \
  -e JAVA_MX=4G \
  -v "$(pwd)/data":/data:rw \
  -p 8080:8080 \
  trivectortraffic/opentripplanner:rsrvu \
    --basePath /data \
    --graphs /data/graphs \
    --router nvdb \
    --server
```

Kommandot startar en temporär OTP serverinstans (`--server`) som lyssnar på port 8080 lokalt. Denna instans läser in grafobjektet som skapades i föregående steg. Standard, eller `default`, routern sätts till `nvdb`.

OTP har en mängd inställningar för att justera hur grafobjektet ska byggas samt standardvärden för ruttuppslagningar. Mer utförlig dokumentation kan hittas här: http://docs.opentripplanner.org/en/latest/Configuration/

Göra nätutläggning

För att göra nätutläggningen behövs Python-skriptet `process_rvu.py` samt filen `requirements.txt` som erhålls separat. Skriptet behöver tillgång till ett antal mjukvarubibliotek som installeras med kommandot:

```
pip install -r requirements.txt
```

Det är nu möjligt att med hjälp av skriptet `process_rvu.py` göra en nätutläggning av RVU datan genom att köra:

```
python3 process_rvu.py sökväg_till_spss_fil.sav
```

Resultatet sparas i en GeoJSON fil med namnet `result.json` i den aktuella katalogen.

Geodatan i resultatet sparas som *LineString* geometri kapslad i *Feature* objekt med följande attributer:

person_id
: Individ id direkt från RVUn

trip_id
: Per individ sekventiell id för var resa

i
: Sekventiellt id för samtliga resor i RVUn

n
: Räknare för varje lyckad ruttuppslagning

leg

Sekventiellt segmentnummer per resa

start

Anger om detta segment börjar i startpunkten för resan

end

Anger om detta segment slutar i målpunkten för resan

mode

Färdmedel för segmentet enligt OTP

huvudfm

Huvudfärdmedel enligt RVUn

arende

Ärende enligt RVU

time

Tidpunkt för resan

date

Datum för resan

type

Typ av geometri
null = vanlig geometri
'stops' = komplementgeometri med besökta hållplatser

Om färmedlet i OTP är något typ av kollektivtrafik returneras endast de hållplatser som fordonet enligt tidtabellen passerar. Då färdmedlet är *BUS* gör skriptet ett försök att omvandla denna punktdata till en geometri som följer vägnätet genom att göra ruttsökningar mellan varje punktpar med färdmedlet *CAR* istället. Lyckas denna uppslagning så ersätts geometrin. Den ursprungliga geometrin sparas dock med `type` satt till värdet `stops`.

Översättningen mellan de färdmedel som används i RVUn och de som OTP förväntar sig sker genom en tabelluppslagning i skriptet. Nedan ses den aktuella tabellen.

```
mode_map = {
    'Bil som förare': 'CAR',
    'Moped/MC': 'CAR',
    'Flyg': 'AIRPLANE',
    'Annat': 'TRANSIT,WALK',
    'Taxi (ej färdtjänst)': 'CAR',
    'Till fots': 'WALK',
    'El-cykel': 'BICYCLE',
    'Cykel': 'BICYCLE',
    'Tåg': 'RAIL,WALK',
    'Färdtjänst': 'CAR',
    'Bil som passagerare': 'CAR',
```

```
    'Buss': 'BUS,WALK'
}
```

För att laborera med översättningar kan värdena till höger redigeras i skriptet. Tillgängliga färdmedel är BICYCLE, WALK, CAR, BUS, TRAM, SUBWAY, RAIL, FERRY, CABLE_CAR, GONDOLA, FUNICULAR, AIRPLANE, TRANSIT (innefattar alla typer av kollektivtrafik) och ALL. Färdmedlen kan kombineras om så önskas; t.ex. `BUS,WALK,BICYCLE`.

Det är även möjligt att dynamiskt ändra ett antal andra parametrar vid varje ruttuppslag. Dokumentation över tillgängliga parametrar och värden finns på http://dev.opentripplanner.org/apidoc/1.0.0/resource_PlannerResource.html

4. Resultat

Totalt kunde rutter för 37569 huvudresor hittas. Dessa resor består i sin tur av 46684 segment. Orsaken till bortfall av resor beror på orsaker som koordinater utanför området för vägnätet, kollektivtrafikresor där tidsangivelser saknas, färdmedel som ej kunnat ruttas som t.ex. flyg, etc.

Ser man på färdmedelsfördelningen är det största tappet för tåg gentemot indata. Orsaken är troligtvis för snäva parametrar för översättning mellan uppgett huvudfärdmedel och färdmedelsvalen som skickas till OTP. I nätutläggningen har färdmedlen, som används i förfrågningen mot OTP, för resor med tåg begränsats till *RAIL,WALK* med maximal gångdistans på 1 km. Ett alternativ hade varit att öka max tillåtna gångdistans och/eller tillåta fler färdmedel. Ett möjligt alternativ skulle vara *RAIL,WALK,BICYCLE* eller *TRANSIT,WALK*. Det senare riskerar dock att tågresor helt faller bort om en bättre/snabbare färdväg med enbart buss hittas. En alternativ metod för att avhjälpa problemet är att göra en separat sökning efter närmaste tågstation per start- och målpunkt och dela upp ruttsökningen där resor mellan stationerna är begränsade till *RAIL* och resorna till och från järnvägsstationerna använder andra färdmedel.

Färdmedelsfördelning med avseende på färdmedel från RVUn

färdmedel	antal	resultat %	rvu %
Annat	64	0.17	0.25
Bil som förare	21595	57.48	53.41
Bil som passagerare	3783	10.07	9.77
Buss	3246	8.64	8.47
Cykel	4694	12.49	11.02
El-cykel	578	1.54	1.35
Färdtjänst	55	0.15	0.13
Moped/MC	148	0.39	0.38
Taxi (ej färdtjänst)	55	0.15	0.22
Till fots	2666	7.10	6.54
Tåg	685	1.82	8.07

För segmenten ser fördelningen annorlunda ut. Detta beror på att det tillkommit en mängd gångresor som knyter ihop kollektivtrafikresorna.
Färdmedelsfördelning för erhållna ruttsegment

färdmedel	antal	%
FERRY	1	0.00
RAIL	708	1.52
CAR	25636	54.91
SUBWAY	2	0.00

WALK	10325	22.12
TRAM	1	0.00
BICYCLE	5285	11.32
BUS	4726	10.12

Det erhållna resultatet kan ge en uppfattning om resvägar och mönster men ska inte ses som en definitiv sanning i och med att det enda som är känt är start- och slutpunkt för en resa. Eventuella genvägar, uppehåll, eller andra ruttavvikelser kan inte återges. Även kvaliteten på vägnätverket som används som underlag inverkar då inte alla vägar, speciellt gång- och cykelbanor, är med samt att alla nödvändiga attribut för korrekt ruttplanering inte behöver vara korrekta.

För att förbättra metoden behöver jämförelser med resedata insamlad med GPS i samma geografiska område göras. Referensdatan kan användas för att justera både statiska ruttningsparametrar och analyseras om vissa områden eller resmål kräver mer dynamisk ändring av parametrar vid ruttsökning för att resultatet ska bli mer representativt.

5. Appendix

5.1 NVDB lager till OSM

lager	osm nyckel
tne_reflinkpart	*referensnät*
tne_ft_vagtrafiknat_1	highway=
tne_ft_tattbebyggtomrade_1	highway=unclassified –> highway=residential
tne_ft_vaghallare_1	operator=
tne_ft_vagbredd_1	width=
tne_ft_trafik_4	*används ej*
tne_ft_slitlager_1	surface=
tne_ft_hastighetsgrans_1	maxspeed=
tne_ft_gcm_vagtyp_1	cycleway=,footway=
tne_ft_gatunamn_1	name=
tne_ft_funkvagklass_1	highway=
tne_ft_bro_och_tunnel_1	bridge=/tunnel=,layer=
tne_ft_driftbidrag_statlig_116	*används ej*
tne_ft_forbjudenfardriktning_1	oneway=
tne_ft_forbudtrafik_1	access=
tne_ft_motorvag_1	highway=motorway
tne_ft_motortrafikled_1	highway=trunk
tne_ft_c_rekbilvagcykeltrafi_1	bicycle=
tne_ft_c_cykelled_1	bicycle=
tne_ft_farjeled_1	highway=,ferry=

5.2 Anonymisering

Resultatet som sådant är ej anonymt och berörs av gälland GDPR-lagstiftning. För att kunna presentera resultat krävs troligtvis någon form av anonymisering. Nedanstående PostGIS SQL kod kan användas för att skapa ett anonymiserat resultat som, efter granskning, kan användas vid visualisering och/eller annan spridning.

```
with points as (
  -- samla ihop start- och målpunkter för varje resa
  select
    ogc_fid,      -- objekt id
    person_id,    -- person id
```

```
    "start",      -- anger om punkten är en startpunkt
    false "end",  -- anger om punkten är en målpunkt
    st_startpoint(wkb_geometry) geom
  from rsrvu
  where "start" and mode not in ('BUS', 'RAIL', 'FERRY', 'SUBWAY', 'TRAM') --
exkludera
  union all
  select
    ogc_fid,
    person_id,
    false "start",
    "end",
    st_endpoint(wkb_geometry) geom
  from rsrvu
  where "end" and mode not in ('BUS', 'RAIL', 'FERRY', 'SUBWAY', 'TRAM')
), clusters as (
  -- beräkna kluster för alla start- och målpunkter
  select
    ogc_fid,
    person_id,
    "start",
    "end",
    st_clusterdbscan(geom, {{cluster_radius}}, 3) over () cid,
    geom
  from points
), cluster_geoms as (
  -- beräkna antal punkter och unika person_id per kluster
  select
    cid,
    count(*) n_points,
    count(distinct person_id) n_person_id,
    st_collect(geom) geom
  from clusters
  where cid is not null
  group by cid
), point_stats as (
  -- sammanställ graden av anonymisering per start- och målpunkt
  select
    p.*,            -- alla attribut från points
    cid,            -- kluster id
    cg.n_points,    -- antal punkter i klustret där punkten ingår
    cg.n_person_id, -- antal unika person id i klustret där punkten ingår
    to2018.tatort,  -- namnet på eventuell tätort där punkten befinner sig
    case            -- beräkna lämplig anonymiseringsgrad
    when coalesce(n_person_id, 1) < 5 and tatort is null then
{{anon_high_ext}} * random() + {{anon_high_min}}
    when coalesce(n_person_id, 1) >= 5 and tatort is null then
{{anon_mid_ext}} * random() + {{anon_mid_min}}
    when coalesce(n_person_id, 1) < 5 and tatort is not null then
{{anon_low_ext}} * random() + {{anon_low_min}}
    else {{anon_none_ext}} * random() + {{anon_none_min}}
    end  / (st_length(rvu.wkb_geometry) + 1) cut_length
  from points p
  join clusters c using (ogc_fid, "start", "end")
```

```
  left join cluster_geoms cg using (cid)
  left join to2018 on st_intersects(to2018.geom, p.geom)
  join rsrvu rvu using (ogc_fid)
)
-- generera anonymiserad data
select
  ogc_fid,          -- GDAL genererat objekt id
  rsrvu.person_id,  -- person id
  trip_id,          -- sekventiellt reseid per person
  i,                -- globalt rese index från RVU
  n,                -- sekvensnummer för lyckad ruttning
  leg,              -- sekvens/segmentnummer per n
  rsrvu."start",    -- anger om denna geometri innehåller en startpunkt
  rsrvu."end",      -- anger om denna geometri innehåller en målpunkt
  "mode",           -- färdmedel för segmentet i OTP
  huvudfm,          -- huvudfärdmedel enligt RVU
  arende,           -- ärende enligt RVU
  "time",           -- tidpunkt för resan
  "date",           -- datum för resan
  "type",           -- typ av geometri, null = vanlig geometri, 'stops' =
komplementgeometri med besökta hållplatser
  -- följande fyra attribut bör lämnas bort vid distribution, används vid
felsökning/analys
  --trip_start.geom is not null anon_start,   -- anger om startpunkten är
anonymiserad
  --trip_end.geom is not null anon_end,       -- anger om målpunkten är
anonymiserad
  --cut_length_start, --
  --cut_length_end,
  case -- använd beräknade anonymiseringsvärden för att kapa linjen
  when coalesce(cut_length_start, 0) + coalesce(cut_length_end, 0) >= 1 then
null -- mer kapning än linjen är lång => null
  when cut_length_start is not null or cut_length_end is not null -- kapa
linjen med beräknade värden
    then st_linesubstring(wkb_geometry, coalesce(cut_length_start, 0), 1 -
coalesce(cut_length_end, 0))
  else wkb_geometry -- behåll ursprunglig linje (kollektivtrafiksegment och
resor som inte har något start/mål)
  end geom
from rsrvu
left join (
  select
    *,
    cut_length cut_length_start
  from point_stats where "start"
) trip_start using (ogc_fid)
left join (
  select
    *,
    cut_length cut_length_end
  from point_stats where "end"
) trip_end using (ogc_fid);
```

Ersätt `{{cluster_radius}}` med ett värde i meter (när t.ex. EPSG:3006 används) för önskat sökområde för klustringsalgoritmen. Värdena för `{{anon_*_min}}` anger minsta längd i meter

som ska kapas medan `{{anon_*_ext}}` är värdet i meter som maximalt ska kapas utöver minimi. Det senare värdet multipliceras med slumpmässigt värde mellan 0 och 1 för varje punkt som ska anonymiseras.

5.3 Modifierad OTP källkod

OTP hade vid projektets start inte stöd för färdmedlet *TAXI* i sin GTFS-tolk. Detta har krävt att viss modifikation av källkoden har behövts för att kunna hantera de GTFS-filer som erhålls från Trafiklab. Källkoden finns tillgänglig på följande adress: https://github.com/joakimfors/OpenTripPlanner i "branchen" `mode-taxi`. En *Docker-avbild* byggd från källkoden kan hämtas från https://hub.docker.com/r/trivectortraffic/opentripplanner med annoteringen *rsrvu*.