OCELLUS
INFORMATION SYSTEMS AB

Rapport LIA

En tillgänglighetsanalys av medborgares avstånd till närmaste busshållplats.

Slutrapport LIA-period 2

Mikael Leonidsson

Avesta April 2024

OCELLUS
INFORMATION SYSTEMS AB

SAMMANFATTNING

Inom utbildningen Mobila system och GIT har jag gjort praktik på avdelningen Regional utveckling inom Region Dalarna. Uppgiften jag fick var ett slags tillgänglighetsanalys som gick ut på att ta reda på hur långt varje enskild medborgare i Dalarna har till sin närmaste busshållplats. Med hjälp av programmeringsspråket R samt databashanteraren PostgreSQL med tilläggen PostGIS och pgRouting har en funktion tagits fram som inom en rimlig tid kan beräkna detta. Denna rapport beskriver processen för att ta fram denna funktion.

OCELLUS
INFORMATION SYSTEMS AB

Innehåll

SAMMANFATTNING 2
INLEDNING 4
Bakgrund 4
Problemformulering/frågeställningar 4
Avgränsning 4
Syfte/Mål 4
ARBETSSÄTT/METOD 5
Om avdelningen 5
Verktyg 5
Programmeringsspråk 5
Databas 5
Process 5
1. Ladda hem och förbereda data 5
2. Ladda upp data till databasen 6
3. Skapa nya noder i nätverket 6
4. Skapa graf av nätverket 7
5. Koppla punkter till noder i graf 7
6. Beräkna kostnaden för att ta sig från varje adress till närmaste hållplats 8
RESULTAT 9
SLUTSATSER OCH DISKUSSION 11

INLEDNING

Bakgrund

Inom utbildningen Mobila System och GIT har vi två perioder med Lärande I Arbete (LIA), under LIA-period 2 har jag praktiserat på Region Dalarna.

Avdelningen Regional utveckling inom Region Dalarna arbetar för att främja utvecklingen inom olika områden såsom näringsliv, kompetensförsörjning och samhällsbyggnad. Under Regional Utveckling finns avdelningen Samällsanalys vars uppgift är att ta fram analyser inom dessa områden så att beslutsfattare kan fatta välgrundade beslut.

En av dessa analyser är en tillgänglighetsanalys där avstånd till närmaste bushållsplats för alla medborgare skall tas fram.

Problemformulering/frågeställningar

Går det att programmera en funktion som på ett lättanvändligt sätt kan utföra en tillgänglighetsanalys på avstånd adress till busshållplats inom en rimlig tid?

Hur kan denna funktion göras så allmän som möjligt att tillgänglighetsanalyser kan utföras på andra data?

Avgränsning

Rapporten beskriver den arbetsuppgift jag haft under min praktik på Region Dalarna snarare än mina upplevelser av själva praktikperioden.

Syfte/Mål

Syftet med arbetet är att beskriva processen samt vissa av de designbeslut som tagits i framtagandet av en funktion som på ett tids- och resurseffektivt sätt kan göra avståndsberäkningar på stora mängder data.

ARBETSSÄTT/METOD

Om avdelningen

På avdelningen Samhällsanalys har de under många år arbetat med att ta fram en stor mängd skript programmerade i R som dels laddar hem data och sedan manipulerar det. De har en GitHub där de delar sina skript och är också drivande i flertalet samverkansgrupper inom GIS och programmering. Så långt det är möjligt har de valt att arbeta med öppen källkod-programvaror och språk, därmed genomfördes även detta projekt med dessa programvaror.

Verktyg

Programmeringsspråk

Språket de arbetar med är R, ett språk som ofta används för statistiska beräkningar och visualisering av data. En stor fördel med R är att det finns många användarutvecklade tilläggspaket som utökat funktionaliteten, några av dessa ger möjligheten att snabbt visualisera geografisk data utan att behöva öppna ett GIS.

Databas

På en egen serveryta har de installerat en PostgreSQL-databas med tilläggen PostGIS och pgRouting vilket möjliggör avancerade geografiska beräkningar direkt i databasen.

Process

För att lösa uppgiften identifierades ett antal steg vilka behövdes gås igenom för att nå önskat resultat.

1. Ladda hem och förbereda data

Det fanns redan skript för att ladda hem data och manipulera detta data för att få fram den information som krävdes. Undantaget var data från kollektivtrafiken som laddas hem från Trafiklabs API enligt GTFS-specifikationen (General Transit Feed Specification). Det som var av intresse för uppgiften var att ta fram alla hållplatser som faktiskt användes i det aktuella GTFS-datat. Varje hållplats består av ett eller flera hållplatslägen, där hållplatsläge är själva punkten där bussen stannar och hållplats är centrumkoordinaten för alla hållplatslägen som hör till hållplatsen (se figur 1). Det skript som fanns hade ett fel som gjorde att det blev dubletter på en del ID:n.

	stop_id	stop_name	stop_lat	stop_lon	location_type	parent_station	platform_code
	All	knutpunkte	All	All	All	All	All
2330	9021020580000000	Falun Knutpunkten	60.604564	15.634954	1	NA	
9177	9022020580000000	Falun Knutpunkten	60.604807	15.635501	0	9021020580000000	A
9178	9022020580000002	Falun Knutpunkten	60.604681	15.635246	0	9021020580000000	B
9179	9022020580000004	Falun Knutpunkten	60.604501	15.634882	0	9021020580000000	C
9180	9022020580000004	Falun Knutpunkten	60.604231	15.634262	0	9021020580000000	D
9181	9022020580000004	Falun Knutpunkten	60.604771	15.635866	0	9021020580000000	E

Figur 1 - Utdrag av stops från GTFS-data. Rött streck visar på dubletter. Grönt på identifieringen av hållplats.

Felet uppstod då stop_id (i det här fallet) är sammansatt av olika bitar av information och därmed blev ”talet” större än vad som kan lagras som heltal i ett fält av datatypen flyttal. Detta ledde till att den avrundade det sista löpnumret felaktigt och därmed blev det dubletter. Lösningen var att läsa in id som tecken istället för tal.

2. Ladda upp data till databasen

När all data var förberedd och x- och y-koordinater omgjorda till geografiska objekt laddades de upp till databasen och geometrikolumnerna indexerades. Under utvecklingsfasen har en separat test-databas upprättats och samtliga tabeller har sedan kopierats till ett separat schema för snabb rensning och sedan omstart med originaldata.

3. Skapa nya noder i nätverket

Detta steg gjordes inte i den första versionen av funktionen utan då kopplades adresser och hållplatser till den närmaste existerande noden i nätverket. Även om funktionen lyckades exekvera utan fel så blev resultatet inte så tillförlitligt.

Figur 2 - Adresser (röd) och hållplatser (gul) kopplas till närmaste existerande nod (vit).

Som vi ser i figur 2 kopplades de fyra adresserna (med streck) till dess geografiskt närmaste nod samtidigt som hållplatsen kopplades till samma nod med resultatet att dessa fyra adresser hade 0 m i avstånd till närmaste hållplats.

För att öka precisionen i resultatet behövde varje adress kopplas på nätverket på dess närmaste punkt istället för nod och sedan skapa nya noder på nätverket (se Figur 3, nästa sida). Som alltid är det en avvägning mellan precision och effektivitet, ju fler noder i nätverket desto större precision men tyngre nätverksanalys. För att skapa en kompromiss mellan precision och effektivitet infördes en toleransnivå där den inte lägger till nya noder om det redan finns en existerande nod inom X antal meter från den nya noden. För att vidare minimera antalet nya noder användes samma toleransnivå till att skapa kluster av tänkta nya noder och välja ut den punkt i klustret som låg närmast centroiden som representant. På så vis garanterades också att noden verkligen hamnade på en linje i nätverket (se Figur 3, nästa sida). Toleransnivån tas emot

som en parameter så att användaren själv kan styra över hur precist resultatet ska vara, ett defaultvärde på 3 m gav ca 10% färre nya noder.

Figur 3 - Närmaste punkt på nätverket för adresser. **Referenspunkt (blå) för närliggande nya punkter (grön).**

Linjerna (gatorna, vägarna) i nätverket delades sedan upp utefter de nya punkterna för att skapa ett nytt nätverk. En referens till id:t för den ursprungliga linjen i nätverket lades till och sedan sparades resultatet i en ny tabell som då innehåller det nya nätverket uppdelat på den närmaste punkten för alla adresser. Denna tabell skickas sedan in i samma funktion som upprepar proceduren med hållplatser.

Här togs beslutet att skapa nya tabeller då det finns en mening med att ha kvar nätverket i dess originalform då det används i flertalet olika analyser. Genom att det nya, uppdelade nätverket har en referens till originalet kan information så som hastighetsbegränsningar etc enkelt kopplas till det.

4. Skapa graf av nätverket

När det nya nätverket är skapat körs funktionen pgr_createTopology() från pgRouting som gör hela nätverket till en graf med noder (start- eller slutpunkt på en väglinje) och kanter (själva väglinjen). Här läggs också kostnaden till för varje kant, i det här fallet längden på linjen i meter.

5. Koppla punkter till noder i graf

När nätverket har gjorts till en graf med noder och kanter som illustrerar hur man tar sig från en nod till en annan kan adresser och hållplatser nu kopplas till närmaste nod i grafen. Som vi kan se i figur 4 på nästa sida blir precisionen markant mycket bättre för samma område som i figur 2.

Figur 4 - Adresser med dess närmaste nod i det nya nätverket.

6. Beräkna kostnaden för att ta sig från varje adress till närmaste hållplats

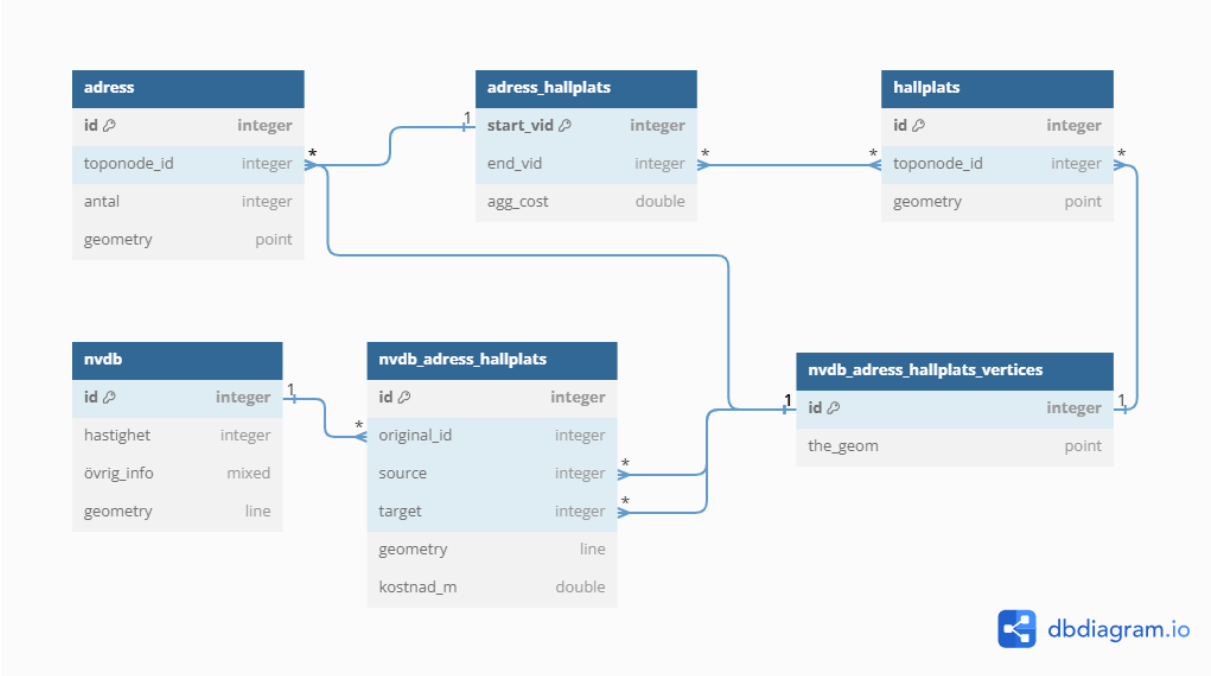
Sista steget är att köra själva nätverksanalysen för att beräkna hur långt varje medborgare har till närmaste hållplats. Det finns en mängd olika algoritmer som kan köras men valet föll på pgrDjikstraNearCost då det finns en möjlighet att köra den med ett många-till-många samband. Genom att sätta parametern ’global’ till ’false’ och låta ’cap’ ha dess defaultvärde ’1’ körs algoritmen och returnerar 1 (cap) resultat per startnod. Vad den returnerar för varje startnod (adress-noder) är den lägsta kostnaden, i det här fallet avstånd i meter, till någon av slutnoderna (hållplatsnod).

I en första version av denna funktion kördes alla nod-id:n från adresstabellen mot alla nod-id:n i hållplatstabellen och sedan uppdaterades adresstabellen med den lägsta kostnaden. Detta tillvägagångsätt fungerade bra på små datamängder men när den kördes på hela datamängden, 98000 adresser till 4000 hållplatser, klarade servern och databasen inte av belastningen rent minnesmässigt. En första del av lösningen var att köra processen i batchar, genom att dela upp alla adresser i körningar om 1000 åt gången lyckades den köra igenom alla utan att krascha men ca 20% fick inga resultat och tester gav att dessa 20% skiljde sig mellan varje körning. Den andra delen i lösningen var att lyfta ut alla unika nod-id:n från adress- och hållplatstabellerna i temptabeller och köra Djikstra-algoritmen på dessa istället. Istället för att uppdatera adresstabellen med avstånd skapades en ny tabell istället som bara innehåller start- och slutnod samt den lägsta kostnaden däremellan. Denna lösning är designmässigt också mycket bättre då adress- och hållplatstabellerna förblir oförändrade och man kan enkelt hitta det kortaste avståndet för varje adress genom att söka rätt på dess nod-id i den nya tabellen.

RESULTAT

Resultatet blev fyra stycken funktioner som var och en hanterar en del i processen. Hela processen med 6 st anrop till dessa funktioner tar totalt ca 25 minuter att köra igenom, vilket får anses vara godkänt då denna typ av analyser endast görs ett fåtal gånger/år. Detta med 98000 adresser, 4000 hållplatser och 250000 tusen vägsegment från den nationella vägdatabasen (nvdb).

Samtliga tabeller och scheman tas emot som parametrar i funktionerna och kan därmed köras på vilka start-, slut- och nätverkstabeller som är önskvärda i den aktuella analysen. De nya tabeller som skapas tar namnet från de tabeller analysen grundar sig på vilket gör resultatet tydligt.



Figur 5 - Förenklat ER-diagram över relationen mellan input- och outputtabeller

Idén med att strukturera databasen på detta sätt medför flera fördelar än bara effektivitet. Om användaren enbart är intresserad av avståndet mellan adress och närmaste hållplats är det enkelt att för varje adress hitta dess toponode_id i adress_hallplats och hämta agg_cost. Vill man ha ännu högre precision går det också att lägga till avståndet mellan adress- och hållplatspunkterna med deras kopplade nodpunkter (i tabellen nvdb_adress_hallplats_vertices). Andra användningsområden kan vara att räkna antalet invånare som har varje hållplats som sin närmaste hållplats eller att köra andra routing-funktioner på start_vid och slut_vid i adress_hallplats. Då funktionerna tar emot vilka tabeller som ska användas i analysen går det att skicka in vilka tabeller som helst som innehåller punkt-data samt ett nätverk vilket gör att det går att göra tillgänglighetsanalyser på andra data också.

Enligt god sed körs sammanhängande databasfrågor inom transaktioner med ’try-catch’-block som vid fel återställer databasen till transaktionens startpunkt. Detta säkerställer att integriteten i databasen förblir intakt. Genom att optimera databasfrågorna och designen av databasen har

processen effektivisierats så pass att den anses vara körbar både ur ett tidsperspektiv men också resursanvändning utan att förlora precisionen.

OCELLUS
INFORMATION SYSTEMS AB

SLUTSATSER OCH DISKUSSION

Även om det finns stora förbättringsmöjligheter så får jag ändå anse att jag inom ramen för vad som är rimligt att hinna med under en 10 veckors praktik har lyckats leverera en flexibel lösning som inte bara fungerar för den tänkta analysen utan även erbjuder möjligheter till andra analyser. Tids- och resursanvändningen är rimlig och användbarheten är ok varpå jag anser att frågeställningarna är besvarade.

Tyvärr räckte inte tiden till för att skapa en omslagsfunktion (wrapper function) som hanterar de olika funktionsanropen så att användaren anropar en funktion istället för som nu fyra olika funktioner vilket skulle göra det än mer användarvänligt. Det finns en önskan och även en vilja hos mig att kommentera upp framför allt de databasfrågor som körs då de i en del fall är väldigt komplexa. Jag är också säker på att det går att hitta fler optimeringar för att göra processen än mer effektiv.

Sammanfattningsvis har det varit en svår och utmanande uppgift att lösa men samtidigt väldigt givande och utvecklande. Det är också kul att detta arbete kommer att delas på Region Dalarnas GitHub så att andra kan fortsätta att utveckla det. Det ger också mig möjligheten att slutföra och förbättra arbetet även om praktikperioden har tagit slut.