CHALMERS

Ruttdetektering av tågtrafik

Examensarbete

Anton Berzelius
Andreas Johansson

Department of Computer Science and Engineering
Division of Computing Science
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2010

Examiner: Dag Wedelin

Abstract

This master thesis investigates the possibilities of using GPS-information from trains to create a model of a railway system, register the routes of the the trains and predict how the trains will travel in the railway system. The thesis work was done at Icomera AB which is a company that specializes in internet connected vehicle solutions.

We describe how the GPS-information is sent from the trains in the form of samples and investigate limitations and possibilities with the available information. About three million samples from 83 SJ-trains were analysed and the samples were filtered with the intent of removing erroneous information. We here found that about 30 % of the samples from the trains were unreliable.

A solution for how the samples from the trains can be used to create a model of the railway system traveled by the trains is presented. We use a large set of historical samples and use cluster nodes and Hermite curves to generate a graph that represent the railway system with stations, nodes and the railway itself. A fully automatic solution is not presented since it did not fit into the time frame of the thesis; instead we created a simple tool which can be used to manually finish the model of the railway system.

Furthermore, we describe how the railway system and the samples from the train can be used to register the routes the trains travel. A state model was used to describe the behavior of the trains in the railway system. The state model creates the possibility of identifying the start and end stations and the stations the trains pass and stop at along the routes. The registered routes were saved in a database and analysed. We present a number of statistical patterns of the registered routes

At last we investigate the possibility of prediction how the trains will travel in the railway system based on the registered routes. Learning decision trees were used for prediction and we evaluate how well the algorithm succeeds with the predictions of the routes the trains will travel. Approximately 90 % of the predictions were correct and we therefore judge the service to be suitable for less critical implementations.

Sammanfattning

Detta examensarbete undersöker möjligheterna att använda GPS-information från tågtrafik för att skapa en modell av järnvägsnätet, kartlägga hur tågen trafikerar järnvägsnätet samt prediktera hur tågen kommer trafikera järnvägsnätet. Examensarbetet utfördes på uppdrag av Icomera AB under sommaren 2008. Icomera AB är ett företag som specialiserar sig på lösningar för internet-uppkopplade fordon.

Vi beskriver hur GPS-information skickas från tågen i form av samplingar och undersöker begränsningar och möjligheter med informationen som finns tillgänglig. Cirka tre miljoner samplingar från 83 SJ-tåg analyserades och samplingarna behandlades för att filtrera bort felaktig information, vi fann att cirka 30 % tågens samplingar var opålitliga.

En lösning presenteras för hur tågens samplingar kan användas för att skapa en modell av järnvägsnätet som trafikerats. Vi utgår ifrån en stor mängd historiska samplingar och använder oss av klusternoder och Hermite-kurvor för att maskinellt generera en graf som representerar järnvägsnätet med stationer, växlar och rälsens utsträckning. På grund av tidsbrist presenterar vi inte en helt och hållet automatisk lösning och använder slutligen ett enkelt verktyg för att manuellt färdigställa järnvägsmodellen.

Vidare beskrivs hur järnvägsmodellen och tågens samplingar kan användas för att kartlägga vilka rutter tågen trafikerar. En tillståndsmodell används för att beskriva tågens beteende i järnvägsmodellen. Tillståndsmodellen möjliggör identifiering av rutternas start- och slutstationer samt de stationer som tågen passerat och stannat vid längs rutterna. De kartlagda rutterna sparades i en databas och analyserades. Vi presenterar ett antal statistiska mönster hos de kartlagda rutterna.

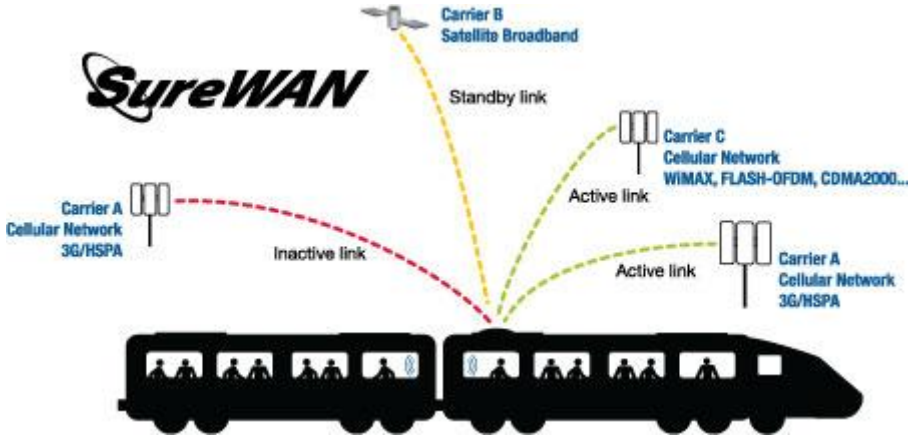
Till sist undersöker vi möjligheten att prediktera hur tågen kommer trafikera järnvägsnätet genom att använda de kartlagda rutterna som enda underlag. Lärande beslutsträd används för prediktering och vi utvärderar hur väl algoritmen lyckas prediktera tågens rutter. Cirka 90 % av prediktionerna blir korrekta och vi anser därför tjänsten är lämplig för mindre kritiska tillämpningar.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Icomera AB är ett företag vars affärsidé bygger på att tillhandahålla Internet-uppkoppling till fordon inom den kommersiella trafiksektorn. Den i dagsläget mest använda tjänsten är Internet för passagerare på bussar och tåg. I och med att fordonen har tillgång till Internet kan Icomera tillhandahålla andra tjänster så som GPS-positionering av fordonen i realtid.



Figur 1-1: En övergripande bild av Icomeras kommunikationssystem.

Positionering av tåg har historiskt sett skett genom att mäta när tåget passerar vissa avläsningspunkter i rälsen. Icomera har sedan de började sätta sina system på tåg använt GPS för att positionera tågen. Med hjälp av Icomeras system skickar tågen samplingar som bland annat innehåller tågens GPS-positioner. Icomera använder idag GPS-information för att visa tågens positioner på en karta. Enklare försök har gjorts för att koppla tågens positioner mot de officiella tidtabellerna men detta har komplicerats av att det oftast är svårt att få automatiserad tillgång till ruttinformationen från tågbolagen.

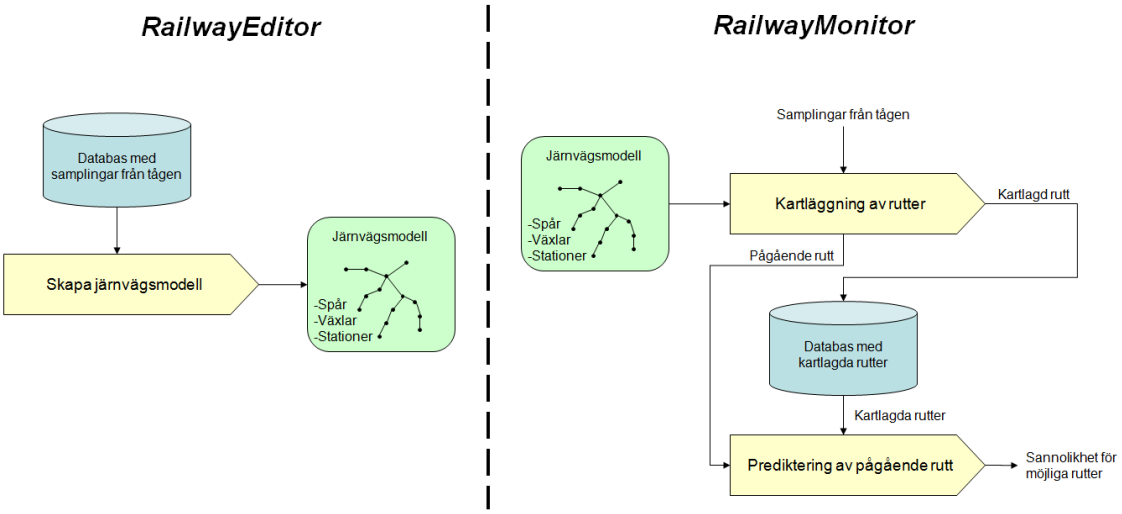
I och med precisionen i dagens satellitnavigering finns det en möjlighet att skapa en automatiserad bild av järnvägsnätet samt tidtabellerna. Detta kan sedan användas för många olika ändamål, t.ex. visa vilka rutter tågen har kört samt prediktera vilka rutter tågen kommer att köra.

1.2 Syfte och avgränsning

Examensarbetet syftar till att ta fram en tjänst som, baserat på GPS-positioner från tågens samplingar, bygger upp en modell över järnvägsnätet med räls, växlar och stationer. Utifrån modellen ska sedan tjänsten i realtid kunna bestämma vilken rutt ett visst tåg färdas. Ett önskemål är att tjänsten enbart använder sig av samplingarna, och inte någon extern information som till exempel tågoperatörernas tidtabeller.

Vi har avgränsat examensarbetet i tre problemområden:

- Skapa en modell som representerar järnvägsnätet med stationer, växlar och rälsens utsträckning.
- Utvärdera och anpassa algoritmer för att kartlägga tågens rutter.
- Analysera om tågens pågående rutter kan predikteras med en utvald algoritm samt utvärdera hur väl algoritmen lyckas prediktera rutterna.



Figur 1-2: Sammanfattning av slutlösning för de tre problemområdena.

Figur 1-2 illustrerar hur vi valde att lösa de tre problemområdena. Nedan följer en sammanfattning av varje problemområde och dess lösning.

1.2.1 Skapa järnvägsmodell

Vårt syfte med järnvägsmodellen är att använda den som en plattform vid kartläggning av tågens rutter. Järnvägsmodellen kan även på längre sikt vara till nytta för andra tillämpningar där tågens positioner i järnvägsnätet är intressanta, några exempel är:

- Dead reckoning – Vid förlust av ett tågs GPS-position kan tågets färdväg i modellen uppskattas med hjälp av senaste känd fart och riktning.
- Eco-driving – Undvika onödiga accelerationer och retardationer genom att förse tågen med information för om hur andra tåg trafikerar järnvägsmodellen.
- Val av mobiloperatörer och modemteknologi – Prioritera länkar baserat på historisk information om prestanda för olika rälssträckor i järnvägsmodellen.

Problemet är att generera en modell som beskriver järnvägsnätet och dess ingående komponenter. Kraven är att modellen baseras på samplingar från tågen och att modellen innehåller komponenterna för räls, växlar och stationer.

Vi valde att beskriva modellen för räls och växlar med en graf där komponenterna representeras av grafens noder och bågar. Tågens samplingar var lagrade i en databas, och vi implementerade algoritmer som genererar grafen med hjälp av informationen hos samplingarna. En applikation, *RailwayEditor*, används för att generera grafen med hjälp av olika verktyg i applikationen.

Vi valde att representera stationer som cirkulära områden där varje station har en position och en radie kring denna. Även stationerna genereras i *RailwayEditor*. De modeller som användaren har genererat med hjälp av *RailwayEditor* sparas i en databas och kan därför uppdateras och förfinas allt eftersom.

1.2.2 Kartläggning av rutter

Vårt syfte med att kartlägga tågens rutter är att använda dem som underlag vid prediktion av pågående rutter. De rutter som blivit kartlagda skulle även kunna användas till andra ändamål:

- Generera tidtabeller baserat på tågens verkliga avgångstider.
- Utvärdera hur väl tågen håller sig till tidigare uppsatta tidtabeller, med syfte att skapa statistik över förseningar med mera.

Problemet består i att, på ett så korrekt sätt som möjligt, kartlägga tågens pågående rutter för att i realtid förstå hur tågen trafikerar järnvägsnätet. Det innebär att identifiera var och när ett tåg påbörjar en rutt, vilka stationer som tåget passerar och stannar vid samt var och när tågets rutt avslutas. Informationen som finns tillgänglig för att lösa problemet är järnvägsmodellen samt de samplingar med GPS-positioner som skickas löpande från tågen.

För att identifiera hur tågen trafikerar järnvägsnätet används järnvägsmodellen som användaren skapat i *RailwayEditor*. Modellen kan laddas i en applikation, *RailwayMonitor,* där användaren kan se var tågen befinner sig i modellen samt hur de trafikerar järnvägsnätet.

För att kartlägga rutterna har vi tillämpat en tillståndsmodell som identifierar hur tågen trafikerar järnvägsmodellen. De kartlagda rutterna sparas i en databas och används för att prediktera vilka rutter tågen kommer trafikera.

1.2.3 Prediktering av pågående rutt

Syftet med analysen är att dra slutsatser om huruvida tågens pågående rutter kan predikteras med en utvald algoritm samt utvärdera hur väl algoritmen lyckas prediktera rutterna. Slutsatserna är värdefulla för Icomera då de kan hjälpa företaget att identifiera potentiella affärsmöjligheter inom området. Några användningsområden för ruttprediktering är:

- Prediktera kommande trafiksituationer, för att till exempel underlätta eco-driving.
- Information och reklam till passagerare beroende på vart tåget är på väg.
- Förseningsinformation i realtid.

Problemet är att utifrån ett tågs pågående rutt finna den redan kartlagda rutt som tåget med högst sannolikheten kommer åka. Vi har valt att begränsa problemet till att prediktera vilka stationer tågen kommer passera längs sina färdvägar samt förutsäga tågens slutstation.

Enligt önskemål från Icomera analyserades möjligheten att prediktera tågens rutter utan hjälp av tågoperatörernas tidtabeller. Vi angrep problemet genom att implementera en algoritm, som använder sig av beslutsträd, för att identifiera vilken av de redan kartlada rutterna som varje tåg med högst sannolikhet kommer åka.

2 Samplingsanalys

Då examensarbetet baseras på Icomeras befintliga system har vi analyserat hur dessa system fungerar, framförallt har vi analyserat de samplingar som skickas från tågen. Samplingar skickas från tågen till Icomeras system i form av UDP-paket som innehåller en mängd information om tågen, bland annat tågens GPS-positioner. Vår samplingsanalys inkluderar information hos samplingar, samplingsfrekvens samt samplingarnas pålitlighet. En insikt vi fick var att GPS-positionen för vissa av samplingarna är opålitlig även om antalet GPS-satelliter för dessa samplingar är stort. Vi såg även att en stor mängd samplingar kan anses vara överflödiga då de skickas klumpvis när ett tåg står still. Samplingsanalysen utfördes med hjälp av egenutvecklade analysverktyg samt genom litteraturstudier avseende positionering av fordon.

2.1 Introduktion

2.1.1 Information hos en sampling

Genom intervjuer med anställda på Icomera samt analys av befintlig källkod för hantering av positioneringsinformation tog vi reda på vad en sampling innehåller. En sampling som skickas från ett tåg innehåller följande information:

- *Tåg-ID*: Tågets unika ID-nummer.
- *Anledning*: Anledningen till att samplingen skickades från tåget.
- *Longitud*: Tågets longitud, beräknas av GPS.
- *Latitud*: Tågets latitud, beräknas av GPS.
- *Altitud*: Tågets altitud, beräknas av GPS.
- *Fart*: Tågets hastighet, beräknas av GPS.
- *Färdriktning*: Tågets färdriktning, beräknas av GPS.
- *Tid och datum*: Tid och datum då samplingen skickades från tåget.
- *Antal GPS-satelliter*: Antalet anslutna GPS-satelliter, beräknas av GPS.

2.1.2 Samplingsfrekvens

Samplingarna skickas inte från ett tåg med jämna mellanrum, utan skickas istället om någon av sju olika situationer uppstår. Systemet på tåget identifierar situationerna och skickar en sampling där *Anledning* beskriver situationen. Värdemängden för *Anledning* är:

- *Time*: Det har gått en viss tid sedan senaste samplingen skickades från tåget.
- *Movement*: Tåget har förflyttat sig ett visst avstånd sedan senaste sampling skickades.
- *Start*: Tågets fart har blivit större än noll.
- *Stop*: Tågets fart har blivit noll.
- *Acquired*: Antal GPS-satelliter har ökat.
- *Lost*: Antal GPS-satelliter har minskat.
- *Boot*: Systemet på tåget har startat om.

2.1.3 Samplingarnas tillgänglighet

Samplingar var tillgängliga dels via en UDP-ström i realtid men även från Icomeras databas som innehöll historiska samplingar från alla tåg. Då samplingarna skickas från tågen med IP-paket med hjälp av UDP-protokollet garanteras inte leveransen av de samplingar som skickas.

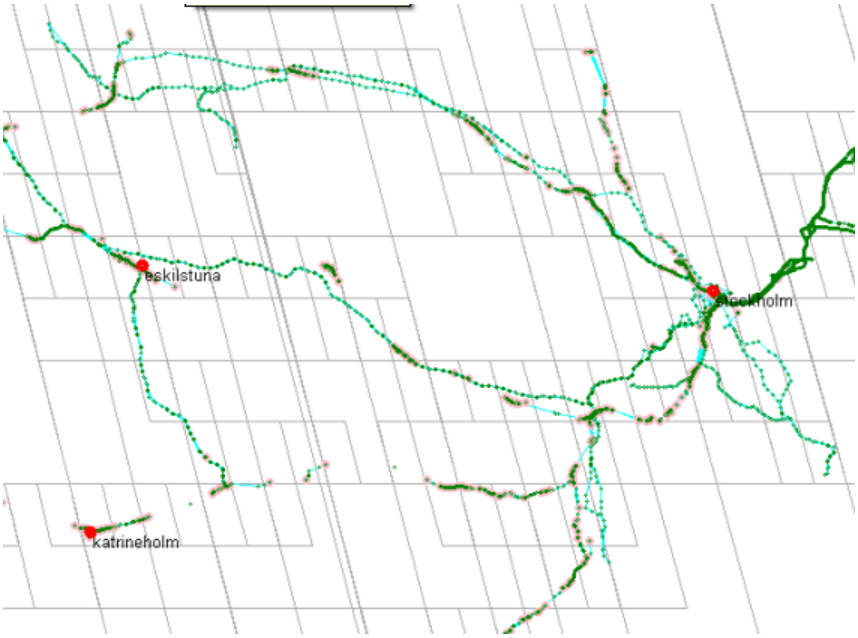
2.2 Analys

Syftet med samplingsanalysen var att förstå samplingarnas egenskaper vilket i högsta grad är intressant då samplingarna användas för att lösa examensarbetets tre problemområden. Nedan följer de analyser som utförts i syfte att vidare förstå våra förutsättningar. Analyserna baserades på 2 774 241 historiska samplingar som skickats från 83 SJ-tåg mellan 2009-09-01 och 2009-10-01, se Appendix 8.1 för mer information.

2.2.1 Visualisering av tillgängliga samplingar

För att bättre förstå samplingarnas egenskaper, och våra förutsättningar, utvecklade vi ett visualiseringsverktyg för att geografiskt visualisera samplingarnas GPS-positioner. Den första grundläggande visualiseringen som utfördes var att rita samplingarnas GPS-positioner som punkter på skärmen där vi hade möjlighet att navigera genom att panorera samt zooma in och ut. Vi såg att samplingar som skickats under loppet av en dag räckte för att en bild av Sveriges järnvägsnät skulle börja växa fram.

Vi hade problem att hantera stora mängder samplingar som skulle ritas ut på skärmen. Mer specifikt var problemet att grafiskt representera samplingarnas GPS-positioner när antalet samplingar blev fler än cirka en miljon.



Figur 2-1: Samplingarnas GPS-positioner ritade som punkter på skärmen med hjälp av ett rutnät.

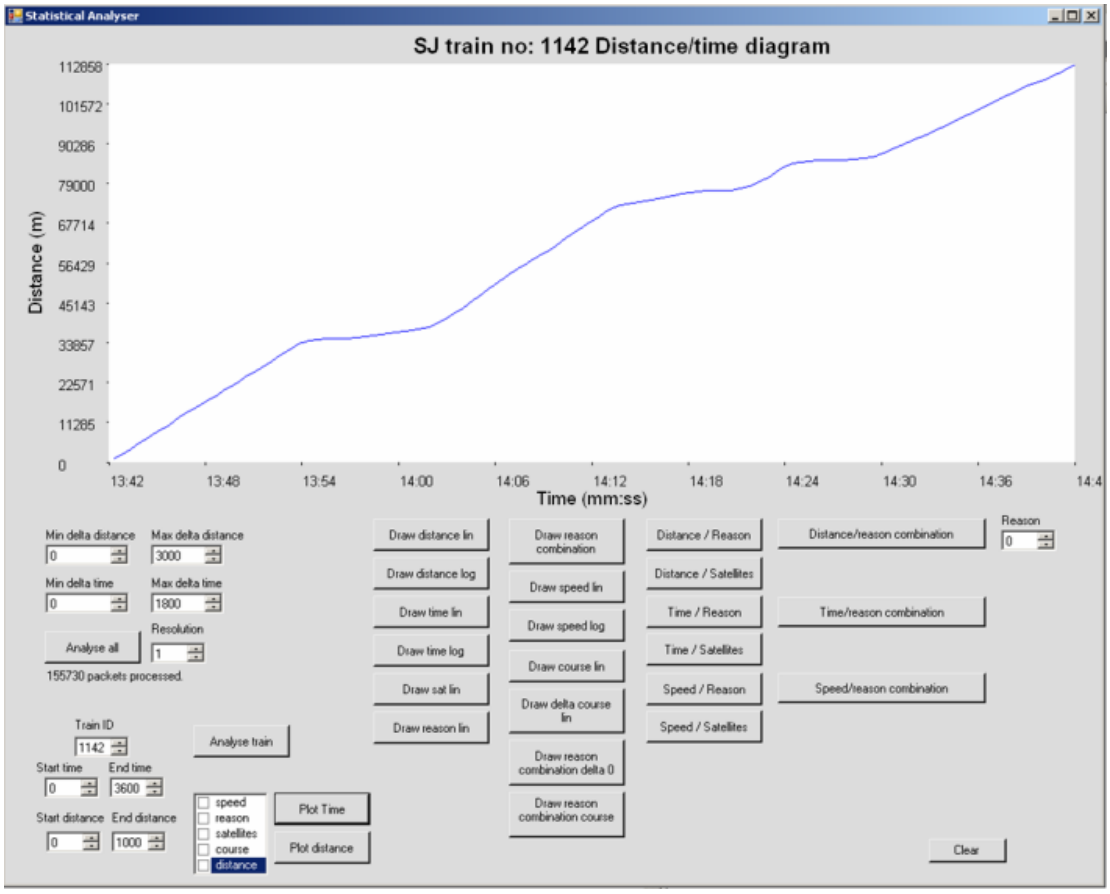
Vi valde att implementera en rutnätslösning som visas Figur 2-1 för att endast rita de samplingar som befann sig inom området som visades på skärmen. Lösningen bygger på att dela in i jordklotet i kvadranter och varje kvadrant delas i sin tur in i kvadranter rekursivt tills den minsta kvadranten uppnår en minimistorlek. Varje minimal kvadrant innehåller information om vilka samplingar som tillhör den, och varje större kvadrant innehåller information om vilka mindre kvadranter den innehåller. På så sätt avgör vi vilka kvadranter som är synliga på skärmen och ritar då enbart upp de synliga samplingarna. Rutnätslösningen användes under hela arbetets gång, bland annat för att identifiera samplingar med närliggande GPS-positioner.

2.2.2 Analysverktyg

Vi utvecklade ett analysverktyg för att åskådliggöra och förstå egenskaper hos samplingarna. Exempel på egenskaper vi ville förstå var:

- Av vilka anledningar skickar tågen samplingarna?
- Hur lång sträcka brukar tågen färdas mellan två samplingar?
- Hur lång tid går det mellan två samplingar?
- Hur påverkar antalet GPS-satelliter tågets position?

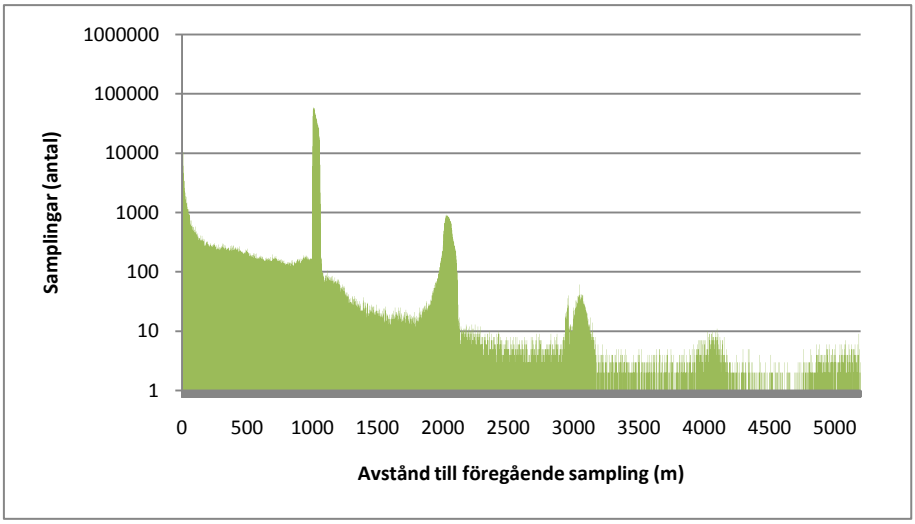
Verktyget innehåller en rad analysfunktioner och med hjälp av verktyget fick vi större förståelse för samplingarnas egenskaper. Ett antal analyser utfördes och nedan presenteras de resultat som vi anser mest relevanta för examensarbetet. Analysverktygets användargränssnitt visas i Figur 2-2, där en kurva för ett tågs sträcka över tid visas.



Figur 2-2: Analysverktygets användargränssnitt.

2.2.3 Analys av samplingsfrekvens

Figur 2-3 visar samplingsfrekvensen i distans, det vill säga hur långt ett tåg färdas mellan två samplingar. Denna analys utfördes för att bland annat förstå upplösningen hos samplingarna från tågen.

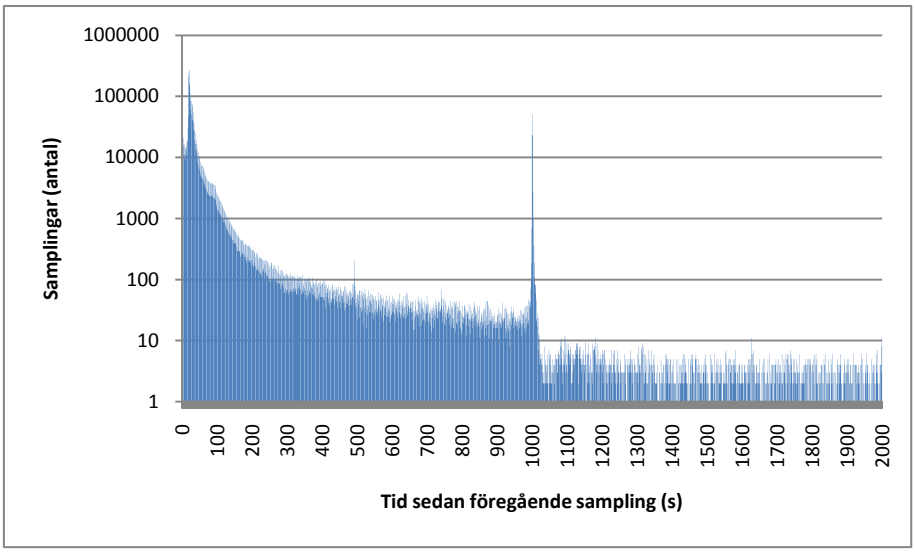


Figur 2-3: Antal samplingar beroende på avstånd mellan samplingarna.

Diagrammet i Figur 2-3 visar antalet skickade samplingar beroende hur långt tågen färdas mellan två samplingar, observera att diagrammet har en logaritmisk skala. Vi ser en topp vid 1000 meter och en mindre topp vid 2000 meter. Toppen vid 1000 meter innebär att en majoritet av samplingarna skickas då ett tåg har förflyttat sig 1000 meter på grund av *Movement*. En hypotes kring varför toppen vid 2000 meter uppstår är att en sampling som skickats vid 1000 meter drabbats av ett förlorat UDP-paket, och att nästa sampling för tåget inkommer först efter 2000 meter. Om hypotesen stämmer innebär det att cirka 1 % av UDP-paketen, och därmed samplingarna, går förlorade.

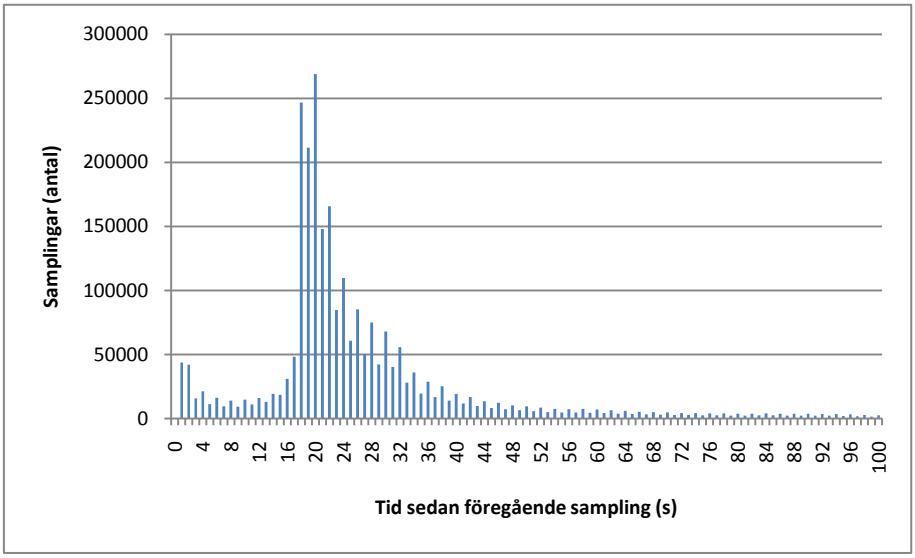
Den högsta toppen finns runt 0 meter. En hypotes är att många samplingar skickas då tågen står still vid en station eller en depå och antalet GPS-satelliter frekvent varierar och nya samplingar skickas på grund av *Acquired* eller *Lost*. En annan orsak kan vara att tåget har dålig satellittäckning och dess GPS-position varierar trots att tåget i verkligheten står still. GPS-mottagaren på tåget uppfattar detta som en förflyttning, om än liten, och samplingar skickas på grund av *Start* eller *Stop*.

Figur 2-4 visar samplingsfrekvensen i tid, det vill säga tiden som gått mellan att två samplingar skickats från samma tåg. Observera att diagrammet har en logaritmisk skala. Tiden analyserades för att förstå av hur snabbt vi kan förvänta oss se förändringar hos tågen.



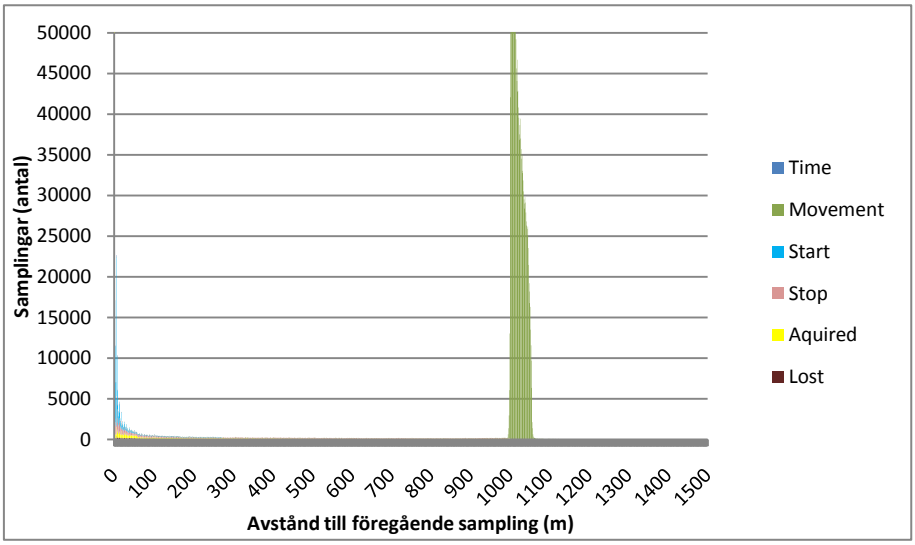
Figur 2-4: Antal samplingar beroende på tid mellan samplingarna.

Av diagrammet i Figur 2-5 ser vi att de flesta samplingar skickas med ungefär 20 sekunders mellanrum. Figur 2-3 visade att de flesta samplingar skickas med cirka 1000 meters mellanrum. En hypotes är att dessa motsvarar toppen vid 20 sekunders mellanrum. Om hypotesen stämmer kan vi grovt räkna ut att de observerade tågen förflyttar sig ca 1000 meter på ca 20 sekunder och därmed färdas med en fart på cirka 180 km/timme. De tåg vi studerar rör sig i hastigheter som motsvarar detta (SJ AB, 2009). Vid 1000 sekunder ser vi i Figur 2-4 de samplingar som skickas av anledningen *Time*.



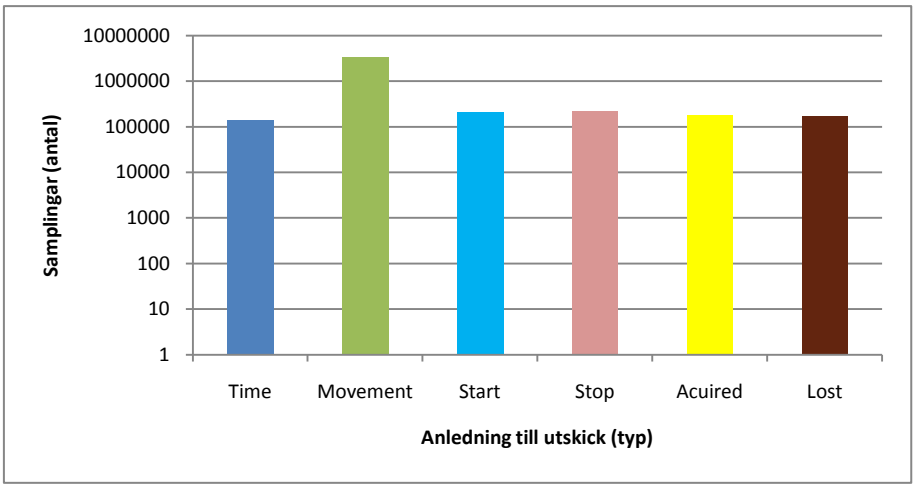
Figur 2-5: Antal samplingar beroende på tid mellan samplingarna, fokuserat mellan 0 och 100 sekunder

Vi analyserade även vilka situationer som oftast orsakar att tågen skickar samplingarna. Analysen utfördes genom att plotta samplingsfrekvensen i distans och färgkoda staplarna beroende på *Anledning* hos samplingarna.



Figur 2-6: Samplingarnas anledning, beroende på avstånd mellan samplingarna.

Diagrammet i Figur 2-6 beskriver antal samplingar beroende på avstånd som tåget färdats mellan två samplingar från ett tåg. Staplarna är färgkodade beroende på *Anledning* hos samplingen. Maximum för antalet samplingar är cirka 90 000, men framgår ej då diagrammet är begränsat till 50 000 samplingar för att bättre synliggöra andra anledningar än *Movement*, som är mest frekvent. Diagrammet i Figur 2-6 visar även att den vanligaste anledningen hos samplingarna är *Movement*, som skickas då tåget förflyttat sig cirka 1000 meter. Vi ser även att samplingar med annan anledning än *Movement* oftast skickas då tågen rör sig lite, eller står still.



Figur 2-7: Antal samplingar fördelade på anledning.

Hur anledningarna hos samplingarna fördelar sig visas i Figur 2-7. Vi ser åter igen att den klart dominerande anledningen till att samplingarna skickas är *Movement*, det vill säga att tåget förflyttat sig cirka 1000 meter.

2.2.4 Analys av samplingarnas pålitlighet

Samplingarnas pålitlighet är en faktor som påverkar hur de kan användas för att lösa examensarbetets tre problemområden. Med pålitlighet menar vi hur väl informationen hos samplingen stämmer överens med verkligheten.

Den GPS-mottagare, EKF CG1-RADIO, är en del av Icomeras system och har enligt specifikationen en maximal noggrannhet på under 10 meter (EKF CompactPCI Products, 2002). Detta gav oss förhoppningar om att samplingarna skulle kunna gett oss ett bra underlag.

Eftersom systemen vi arbetade med, det vill säga tågen, GPS-mottagare och mobila länkar, rör sig i verkligheten finns det en mängd felkällor som kan störa samplingarna. Några exempel är skymmande träd, dålig GPS-täckning, tillfällig störning i tågets uppkoppling och trasig hårdvara. Det är därför viktigt att förstå hur samplingarnas pålitlighet påverkas av omständigheter som vi inte kan påverka, utan istället måste anpassa oss till.

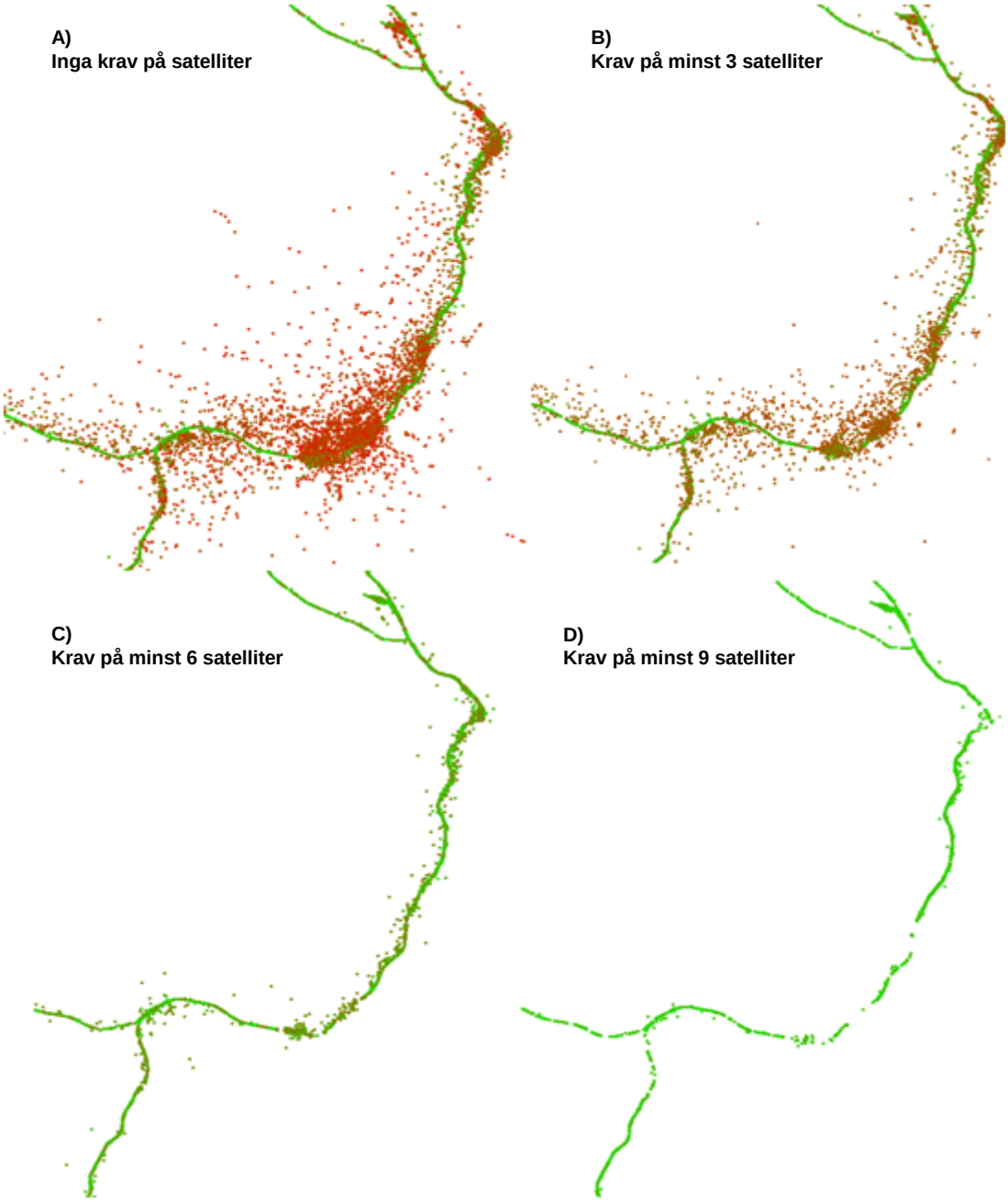
GDOP – Geometric Dillusion of Precision

Faktorer som påverkar GPS-positionens noggrannhet är bland annat kvalitén på GPS-mottagaren och satelliternas position på himlen. En GPS har möjlighet att avgöra hur säker en position är genom att ta hänsyn till satelliternas positioner (Langley, 1999). Således är antalet satelliter vi ser inte direkt avgörande för huruvida vi får en giltig position eller inte - det som är avgörande är hur separerade de satelliter vi ser är från varandra. Tyvärr har vi i vår undersökning inte haft tillgång till det värde som anger hur separerade satelliterna är ifrån varandra, GDOP värdet, utan har istället fått arbeta efter hypotesen att om vi ser tillräckligt många satelliter kommer de vara placerade tillräckligt separerat för att vi ska kunna få en pålitlig position. Vi har därför undersökt hur många satelliter som statistiskt sett krävs.

Visualisering

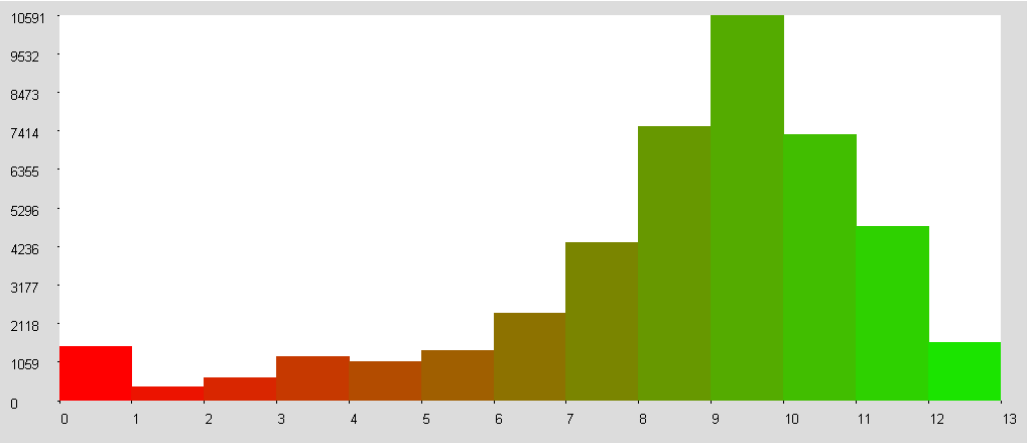
Vi analyserade samplingarnas GPS-positioner genom att visualisera dem på skärmen och jämföra ett antal av dem med en karta (Eniro Sverige AB, 2009). På så sätt kunde vi se hur väl samplingarnas positioner stämde med kartan som vi antog stämma med verkligheten. Vi koncentrerade oss på tunnlarna söder om Stockholm, ett område som kan anses svårt på grund av att GPS-mottagaren inte alltid hinner få kontakt med tillräckligt många satelliter när tåget befinner sig mellan två tunnlar.

Bilderna i Figur 2-8 visar samplingarnas GPS-positioner, färgade i en skala som går från rött till grön. Grönt innebär att GPS-mottagaren har kontakt med många satelliter och rött innebär att få satelliter fanns tillgängliga. Bilderna visar tydligt hur antalet satelliter vi har kontakt med kan påverka GPS-positionen.

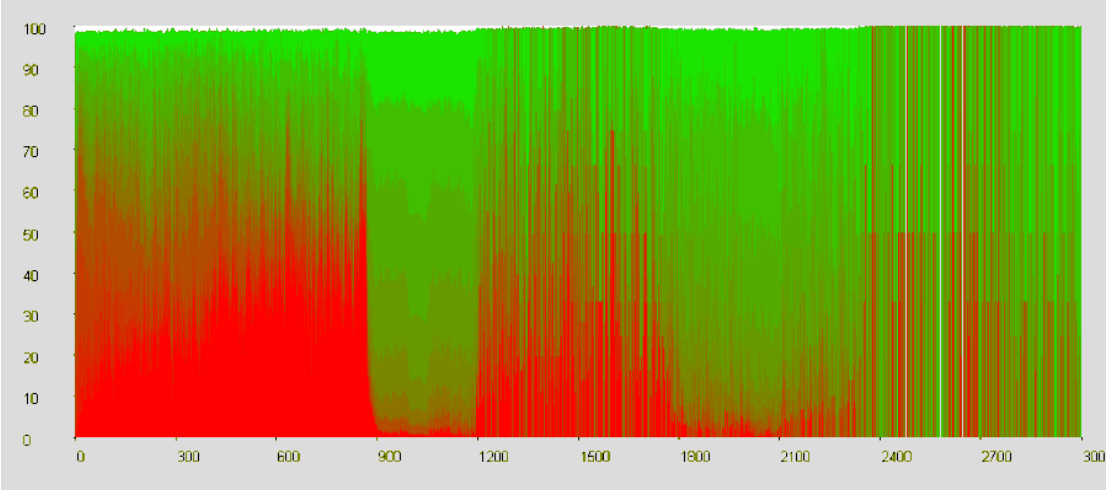


Figur 2-8: GPS-positionerna hos en mängd samplingar, färgkodade beroende på antal satelliter.

Vi ritade även ut en graf (Figur 2-9) som visar hur stor andel av samplingarna som skulle påverkas om vi skulle sätta ett krav på ett visst antal navigationssatelliter.



Figur 2-9: Y-axeln visar antal samplingar, X-axeln visar antal satelliter för respektive sampling.



Figur 2-10: Y-axeln visar andel samplingar i procent. X-axeln visar avstånd mellan samplingar.

Vi bedömde att vi utan större påverkan kunde ställa ett krav på minst 7 satelliter. Vi kan i även i **Error! Reference source not found.** se att de positioner vi fått in efter att tåg har rört sig 1000 m, d.v.s. de mest frekvent förekommande positionerna tenderar till att vara de positioner som understöds av bra GPS-täckning. Detta troligtvis på grund av att dessa positioner lästs av då tåg vart ute och rört sig i öppen terräng.

2.2.5 Sanity check

Vi införde en så kallad sanity check för att omedelbart filtrera bort orimliga samplingar. Vår sanity check baseras på egenskaper hos de tåg samplingarna kommer ifrån, samt idéer vi har fått ifrån en artikel (Froehlich J et.al., 2008) inom området.

Hastighet
Vi undersökte hastigheten som rapporteras i samplingen för att se om den är rimlig. I de fall då hastigheten är mer än 65m/s (234 km/h) och bortser vi ifrån samplingen. Även om X2000 har provkörts i 276km/h så har den i Sverige en maxhastighet på 200 km/h.

Avstånd
Vi undersökte om avståndet mellan positionerna för två samplingar från samma tåg var rimligt med avseende på tiden. Det vill säga, vi undersökte om tåget kunde färdas mellan de två positionerna under tiden mellan de två samplingarna och fortfarande hålla sig under hastighetsbegränsningen vi satt upp. Vi valde även att bortse ifrån samplingar vars positioner var för tätt ihop eftersom vi bedömde att dessa inte skulle hjälpa oss i våra tester.

2.3 Resultat - Samplingsanalys

I Tabell 2-1 till vänster undersöker vi samplingar från 2009-09-01 00:00:00 till 2009-09-08 00:00:00. I Tabell 2-1 till höger undersöker vi samplingar från 2009-09-01 till 2009-10-01.

Samplingar för en vecka		
	Antal samplingar	Andel (%)
Total	640684	100,0
Orimlig hastighet	12756	2,0
Få satelliter	188163	29,4
Opålitliga	200919	31,4
Pålitliga	439765	68,6

Samlingar för en månad		
	Antal samplingar	Andel (%)
Total	2774241	100,0
Orimlig hastighet	54580	2,0
Få satelliter	822670	29,7
Opålitliga	877250	31,6
Pålitliga	1896991	68,4

Tabell 2-1: Antal pålitliga samplingar under en månad.

Vi finner att vi enligt våra kriterier får in en stor mängd positioner som vi inte kan lita på. Vi kastar bort ca 30 % av samplingarna vi får in. Vi anser att detta är rimligt eftersom vi har tillgång till en väldigt stor mängd samplingar som passerar vår filtrering, detta medför att vi inte har något behov av att använda data som kan anses vara tveksam. Vidare ser vi att de samplingar som inte passerar vår filtrering uppträder grupperat i vissa problemområden, som det söder om Stockholm vi tidigare observerat. Vi ser inte heller några avbrott i järnvägsnätet när vi observerar positionen för de samplingar vi filtrerat bort.

2.4 Slutsatser - Samplingsanalys

Den vanligaste anledningen till att en sampling skickas är att tåget rört på sig. Den vanligaste sträckan mellan två samplingar är 1 km och det går oftast 20 sekunder mellan 2 samplingar från samma tåg. Värt att notera är att ett tåg som färdas i 180km/h förflyttar sig ungefär 1km på 20 sekunder.

Att den vanligaste anledningen till en sampling är rörelse är i enlighet med Icomeras design av rapporteringstjänsten för positionering, vars syfte är att skicka in så få samplingar som möjligt när tåget står still på samma position.

Med hjälp av analysverktyget kom vi även fram till slutsatsen att de samplingar som tågen skickar när de stannar och startar tenderar att ha dålig satellittäckning. Vi har dock valt att inte undersöka närmare vad detta har för innebörd i denna fas av examensarbetet.

Vår analys av hur många satelliter som krävdes för att ge oss en pålitlig position gav att vi skulle kräva minst 7 satelliter. Med detta krav blir vi relativt säkra på att varje position vi får är tillräckligt pålitlig. Vi grundar vår säkerhet i att vi ritar ut positionerna färgade efter antal satelliter och jämför det järnvägsnätet vi ritat ut med det järnvägsnät vi ser på Eniro (Eniro Sverige AB, 2009).

De få positioner som trots de krav vi ställt ändå är felaktiga antas vara försumbara vid arbete med stora mängder data bli försumbar. De falska negativa, alltså de vi kastar bort trots att vi kunde använt dem, anser vi även vara försumbara vid arbete med stora mängder data.

3 Modell av järnvägsnätet och dess komponenter

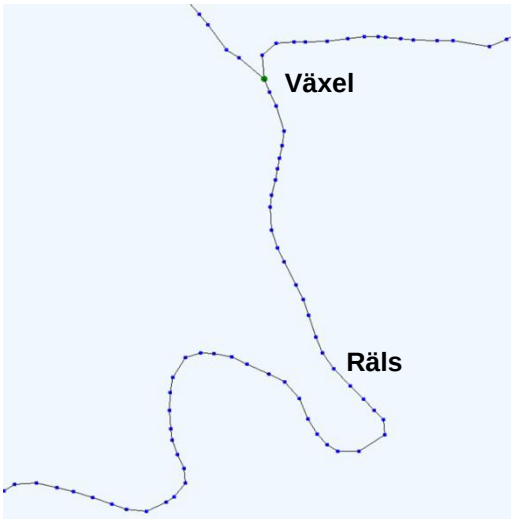
Ett av våra delproblem är att skapa en modell som beskriver järnvägsnätet. Vårt syfte med att skapa en järnvägsmodell är att bygga upp en grund som ska användas vid kartläggning av rutter. Järnvägsmodellen kan även på längre sikt vara av nytta för andra tillämpningar där tågets position är intressant. Ett krav är att modellen baseras på samplingar från tågen. Vi lade en stor del av examensarbetets tid på att ta fram verktyg som används för att generera en högkvalitativ modell som beskriver järnvägsnätet och dess ingående komponenter. Komponenterna som beskrivs är räls, växlar och stationer.

Genom att plotta positionerna för en mängd samplingar ges en tydlig grafisk bild för järnvägsrälsens utsträckning med en upplösning på cirka 20 meter. Detta tyder på möjligheten att konstruera en modell som innehåller högupplöst information om flera olika spår hos en depå eller större station.

Vi undersökte möjligheten att skapa järnvägsmodellen dynamiskt allt eftersom samplingar rapporterades. Det innebär att modellen från början är tom och komponenter skapas eller förändras när tågen skickar nya samplingar. Vi upptäckte dock att metoden var problematisk då modellen blev felaktigt på grund av samplingar med felaktiga positioner och felaktigheterna var svåra att dynamiskt korrigera. Därför valde vi istället att skapa en statisk järnvägsmodell baserat på en mängd historiska samplingar. Genom att betrakta en stor mängd samplingar, för ett geografiskt område, minimeras påverkan av felaktiga samplingar och det blir enklare att bygga en korrekt modell.

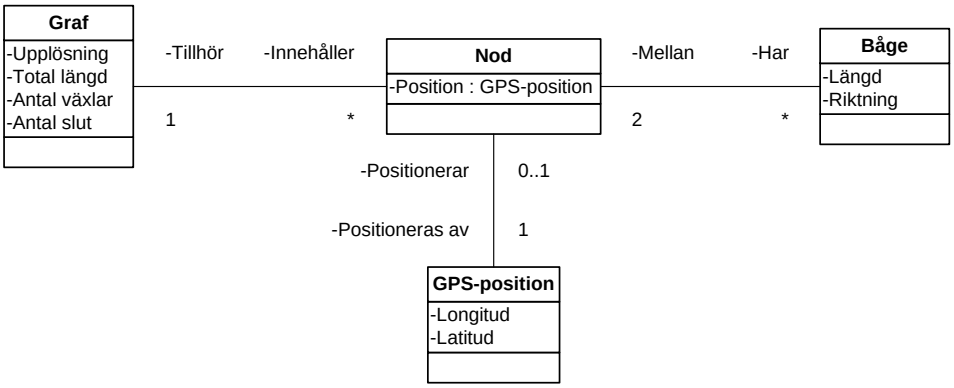
3.1 Modell för järnvägsnätet

Vi valde att basera modellen som beskriver järnvägsnätets komponenter med en graf där rälsen representeras av grafens noder och bågar. Växlarna representeras av noder med fler än två bågar. Ett exempel på en sådan graf visas i Figur 3-1.



Figur 3-1: Modellen som beskriver räls och växlar baseras på en graf.

Grafen innehåller information om nodernas position och hur bågarna kopplar samman noderna. Med hjälp av informationen i grafen kan man bland annat härleda hur växlarna är sammankopplade samt hur lång rälssträcka det är mellan växlarna. Klassdiagrammet i Figur 3-2 beskriver modellens klasser och deras relationer.



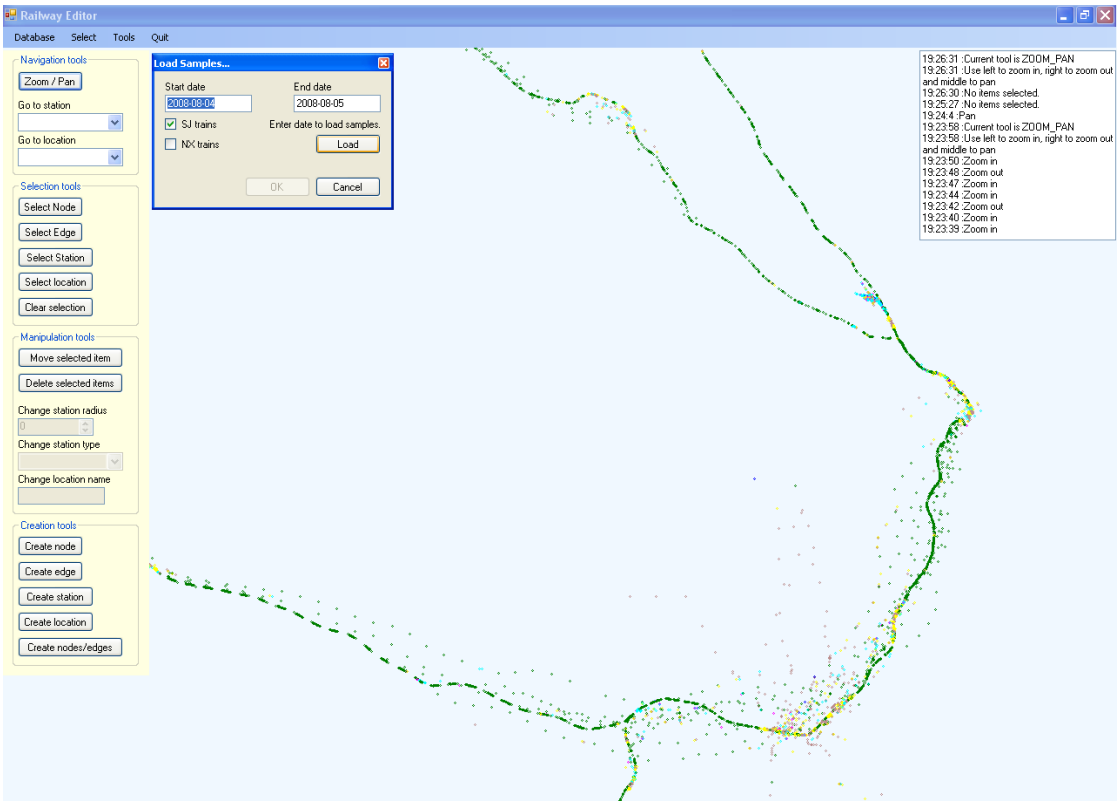
Figur 3-2: Klassdiagram för modellen som beskiver räls och växlar.

Annan information som t.ex. antal parallella spår, enkelriktade sträckor, möjlig färdväg i växel, tunnlar med mera beskrivs inte av modellen och vi har inte tagit hänsyn till dessa i examensarbetet.

3.2 RailwayEditor

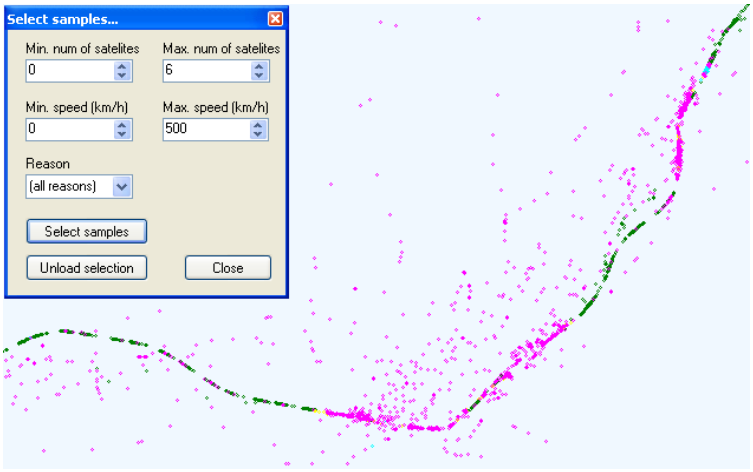
Vi utvecklade en applikation, *RailwayEditor*, som används för att skapa och förändra modellen för järnvägsnätet. Applikationen innehåller ett antal verktyg som används för att skapa och uppdatera komponenterna hos modellen.

En grundläggande funktion hos applikationen är att ladda historiska samplingar. Genom att användaren specificerar ett datumintervall laddas de samplingar som tågen skickat under de angivna datumen. Samplingarnas position plottas på kartan och användaren har möjlighet att navigera genom att panorera och zooma in och ut. Varje sampling färgkodas beroende på anledningen till att tåget skickade samplingen.



Figur 3-3: *RailwayEditor* användargränssnitt med laddade samplingar plottade på kartan.

När historiska samplingarna har laddats kan användaren välja att filtrera bort samplingar beroende på antal satelliter, anledning till att samplingen skickats, fart eller en kombination av dessa mätvärden. Denna funktion är speciellt användbar för att identifiera och filtrera bort samplingar som möjligtvis kan vara felaktiga.

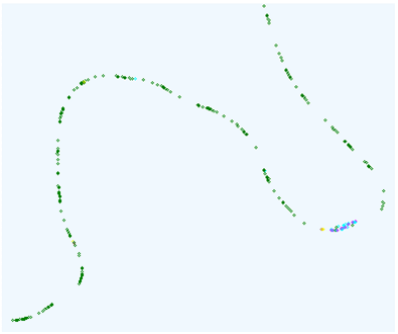


Figur 3-4: Filtrera bort laddade samplingar där antal satelliter är mindre än 6.

När samplingar har filtrerats bort kan användaren använda olika verktyg i applikationen för att generera modellens komponenter som sedan kan sparas i en databas. Resterande delar i det här kapitlet beskriver i detalj hur komponenterna för järnvägsnätets räls, växlar och stationer genereras.

3.3 Problembeskrivning

Problemet är att skapa en graf med hög kvalité, d.v.s. att grafen överensstämmer med det verkliga järnvägsnätet. I vårt fall begränsas informationen om verkligheten till samplingarna från tågen.

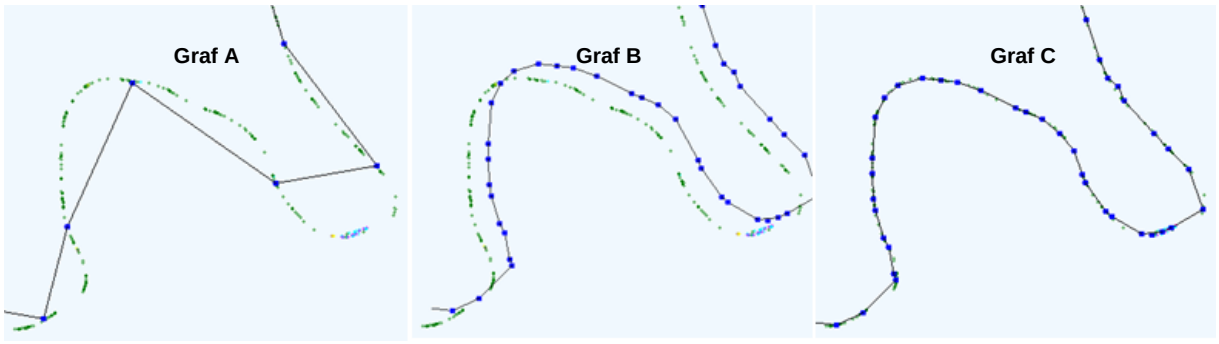


Figur 3-5: Samplingar från tågen är den kända verkligheten.

Vi definierar grafens upplösning som medelängden på grafens bågar, ju mindre medellängden är desto högre upplösning har grafen. Grafens kvalité påverkas av nodernas positioner, grafens upplösning samt hur noderna är sammankopplade. Ett mätvärde, *k*, för kvalitén kan beräknas genom att summera det minsta avståndet mellan samplingarnas positioner och närmsta båge i grafen.

$$k = \sum S_{min}$$

Där S_{min} är det minsta avståndet mellan samplingens position den närmsta bågen i grafen. Genom att minimera *k* kan grafens kvalité maximeras.



Figur 3-6: Illustration för hur avståndet mellan noderna och nodernas position påverkar grafens kvalité.

I Figur 3-6 illustreras tre grafer. I graf A överensstämmer nodernas position väl med verkligheten men grafen har relativt låg upplösning vilket medför ett relativt stort *k*. I graf B är upplösningen högre, men nodernas position stämmer sämre med verkligheten, vilket också medför ett relativt stort *k*. Graf C har relativt hög upplösning och nodernas position stämmer väl med verkligheten vilket medför ett relativt litet *k*.

3.4 Generera grafens noder

Vi har sett att grafens kvalité bland annat beror på nodernas positioner och grafens upplösning. Då grafens upplösning beror på bågarnas längder är upplösning direkt beroende av avståndet mellan noderna. Det är därför viktigt att avståndet mellan noderna blir tillfredsställande samt att nodernas position stämmer så väl som möjligt med verkligheten. Då avståndet mellan samplingarna från ett tåg oftast är cirka 1000 meter måste noderna baseras på information från flera tåg om avståndet mellan noderna skall vara mindre än cirka 1000 meter.

Vi valde att lösa problemet genom att gruppera samplingar från flera tåg med närliggande positioner och låta gruppens medelposition definiera positionen för en nod. Samplingar tillhör samma nod om avståndet mellan samplingarnas positioner är mindre än ett maximalt tillåtet avstånd R_{max}. Storleken hos R_{max} styr hur stort avståndet mellan noderna blir och påverkar därmed grafens upplösning. En parameter N_{min} bestämmer det minsta antalet samplingar som en grupp måste innehålla för att en nod skall skapas. Om det inte finns några grupper som innehåller N_{min} antal samplingar ökas avståndet R_{max} med R_{inc} och ett nytt försök görs, försöken avslutas då R_{max} är större än R_{stop}.

Avståndet mellan två samplingar beräknas utifrån värdena för latitud och longitud. Vi tog inte hänsyn till samplingarnas värde för altitud. Om θ är samplingens latitud och φ är samplingens longitud så beräknas avståndet d mellan två samplingar s_1 och s_2 som

$$d(s_1, s_2) = \arccos(\cos\theta_1 \cos\varphi_1 \cos\theta_2 \cos\varphi_2 + \cos\theta_1 \sin\varphi_1 \cos\theta_2 \sin\varphi_2 + \sin\theta_1 \sin\theta_2)R$$

, där *R* är jordens radie (Michels, 1997).

Algoritmen för att skapa noderna är en variant av *QT clustering* (Heyer et al., 1999), (Ashbrook, et al., 2003) och fungerar så här:

- Användaren anger värden för parametrarna R_{max}, R_{stop}., R_{inc} och N_{min}.
- Mängden *M* innehåller alla samplingar.

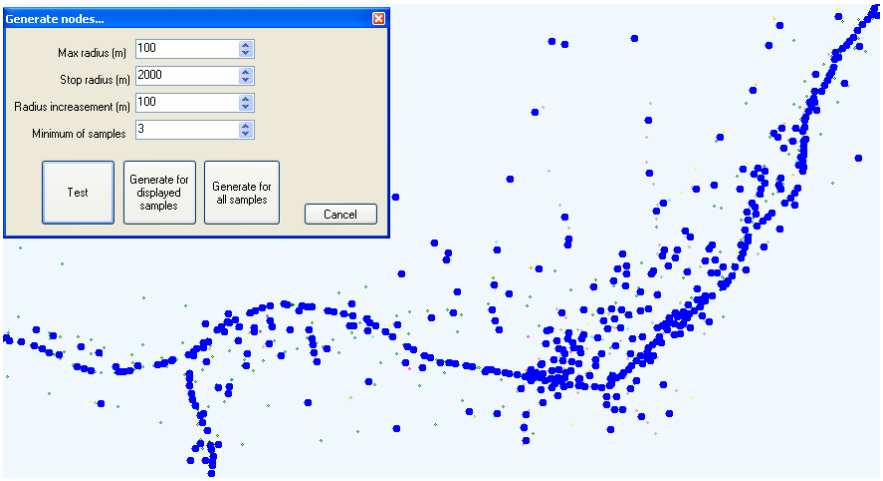
Start:

1. Skapa en grupp G för varje sampling $s \in M$. G innehåller s och de samplingar i M som befinner sig inom en radie R_{max} från s.
2. Välj den grupp G_{best} som innehåller flest samplingar.
3. Om G_{best} innehåller färre än N_{min} samplingar, öka R_{max} med R_{inc} och avsluta om R_{max} är större än R_{stop}, annars starta om med resterande samplingar i M.
4. Skapa en ny nod i medelpositionen för G_{best}.
5. Ta bort samplingarna som tillhör G_{best} från M.
6. Starta om med resterande samplingar i M.

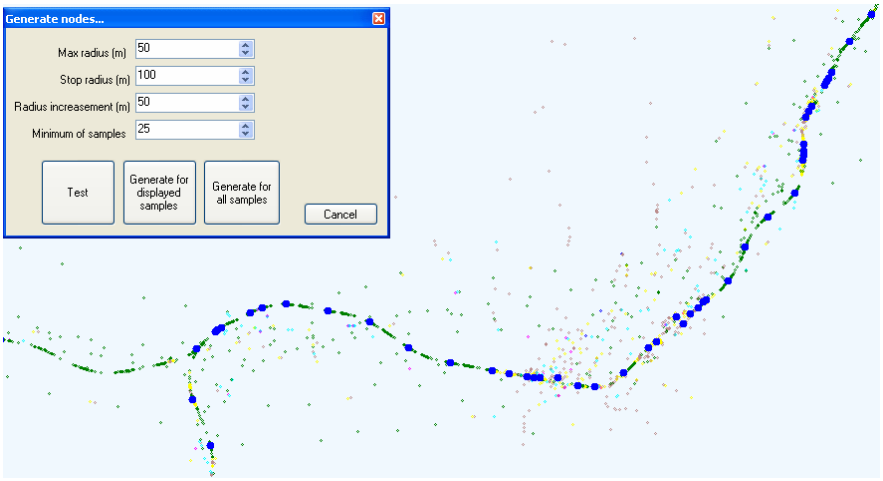
3.4.1 Verktyg för generering av noder

Vi implementerade algoritmen i *RailwayEditor* och skapade ett verktyg som kan användas för att generera noder baserat på de historiska samplingar som laddats. Användaren har möjlighet att ange värden för parametrarna R_{max}, R_{stop}, R_{inc} och N_{min}.

I Figur 3-7 och Figur 3-8 illustreras resultatet för nodgenerering för samma samplingsmängd men med olika parametervärden.



Figur 3-7: För många noder genereras p.g.a. för generösa parametervärden.



Figur 3-8: För få noder genereras p.g.a. för restriktiva parametervärden.

I Figur 3-7 och Figur 3-8 ser man att resultatet för nodgenereringen varierar beroende på parametervärdena; notera att samplingsmängden i de två figurerna är den samma. Figur 3-8 visar att för få noder har genererats då det saknas noder för samplingarna längs till vänster, parametervärdena är i det här fallet för restriktiva. Samtidigt ser man att för många noder har genererats i Figur 3-7 där parametervärdena är för generösa och noder har genererats för samplingar med felaktiga positioner.

Det är problematiskt att hitta de optimala parametervärdena för en stor samplingsmängd då antalet samplingar inom olika geografiska områden varierar kraftigt. En väldigt trafikerad rälssträcka med många samplingar och mycket brus behöver andra parametervärden än en rälssträcka med få samplingar om resultatet skall bli lyckat.

En möjlig lösning på problemet kan vara att låta en algoritm identifiera olika geografiska områden, t.ex. baserat på samplingskoncentrationen i området, och automatiskt bestämma parametervärdena för varje område. (Ashbrook, et al., 2003) föreslår en lösning där noder får genereras med olika parametervärden och ett tröskelvärde för R_{max} kan bestämmas där antalet genererade noder konvergerar beroende på radien R_{max}.

Vi försökte ej lösa problemet med parametervärdena och istället har vi lagt in stöd i *RailwayEditor* för att generera noder för de samplingar som visas på skärmen. På så vis begränsas samplingsmängden och användaren kan hitta lämpliga parametervärden för samplingsmängden som visas på skärmen.

När noderna har genererats ska de sammankopplas med bågar för att färdigställa grafen. Nästa kapitel beskriver hur detta går till.

3.5 Generera grafens bågar

Vi implementerade en egen algoritm för att generera grafens bågar. Noderna som ska sammankopplas identifieras genom att analysera de samplingar som är laddade i *RailwayEditor*. Varje tågs samplingar analyseras i kronologisk ordning och algoritmen söker efter noder som befinner sig på sträckan mellan två på varandra följande samplingar. En båge skapas mellan varje par av två på varandra följande identifierade noder. Om det redan finns en båge mellan noderna ökas bågens vikt med ett. När alla samplingar är analyserade, och bågarna är skapade och viktade, identifierar varje nod i grafen vilka bågar den ska behålla.

Algoritmens huvudsteg är:

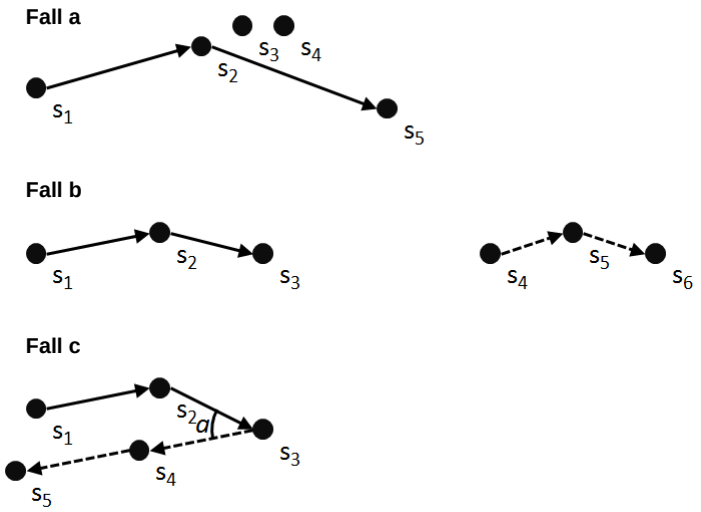
1. Skapa banor för varje tåg baserat på laddade samplingar.
2. Följ varje bana och skapa/vikta bågar mellan noderna som passeras längs banan.
3. Identifiera vilka bågar varje nod ska behålla baserat på bågarnas vikt.

3.5.1 Skapa banor för varje tåg baserat på laddade samplingar

Samplingarna för varje tåg sorteras i tidsordning $(s_1, s_2 \dots s_{n-1}, s_n)$ och sammankopplas sedan till banor som beskriver varje tågs färdväg. Om avståndet $d(s_{i-1}, s_i)$ mellan två samplingar är mindre än d_{min} tas kopplingen mellan s_{i-1} och s_i bort och istället kopplas samplingen s_{i-1} till nästa sampling med $d \geq d_{min}$. Detta fall illustreras i Figur 3-9 (Fall a) där s_2 sammankopplas med s_5 då $d(s_2, s_3) \leq d_{min}$ och $d(s_2, s_4) \leq d_{min}$.

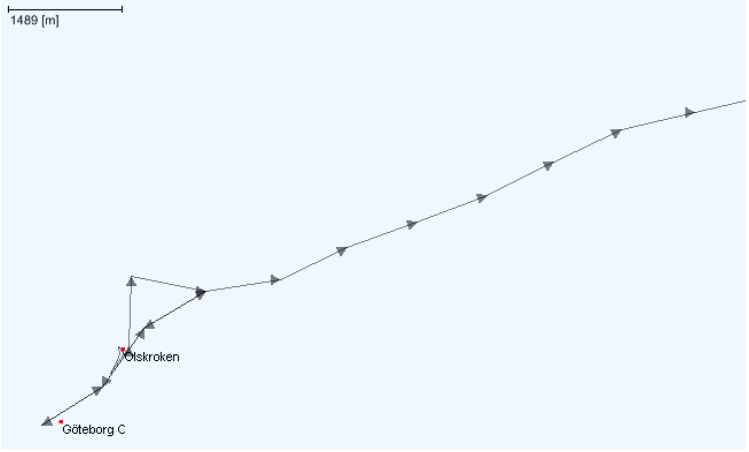
Om $d(s_{i-1}, s_i) > d_{max}$ tas kopplingen mellan s_{i-1} och $s_i$bort och banan avslutas med s_{i-1} som sista sampling. En ny bana skapas med samplings_i som start. Detta fall illustreras i Figur 3-9 (Fall b) där kopplingen mellan s_3 och $s_4$har tagits bort då $d(s_3, s_4) > d_{max}$. Den första banan avlutas med s_3 som sista sampling och nästa bana startar med $s_4$som första sampling.

Algoritmen tar även hänsyn till vinkeln α mellan två de kopplingarna för en sampling. Om $\alpha < \alpha_{min}$ avslutas banan vid samplingen och en ny bana skapas med samma sampling som första sampling. Fallet illustreras i Figur 3-9 (Fall c) där $\alpha(s_3) < \alpha_{min}$. Den första banan avslutas med s_3 och nästa bana startar med s_3 som första sampling.



Figur 3-9: Illustration av olika fall för skapande av banor.

Banorna beskriver tågens färdvägar och används i algoritmens nästa steg för att identifiera hur grafens noder ska sammankopplas. I Figur 3-10 plottas en del av banorna för ett tåg.



Figur 3-10: Del av bana för ett tåg.

3.5.2 Skapa bågar mellan noderna som passeras längs banorna

Algoritmen analyserar banorna var för sig och identifierar vilka noder som befinner sig längs en bana och i vilken ordning noderna ska sammankopplas. Varje bana innehåller minst två samplingar $(s_1, \dots, s_n)$ där banan startar med första samplingen och avslutas med sista samplingen. Noderna och vilken ordning de ska sammankopplas identifieras genom att algoritmen förflytta sig sträckan ds längs en kurva $c(s_i, s_{i+1})$ mellan ett samplingspar och söker efter närmaste nod, n_{best}, vid varje förflyttning. Om avståndet $\mathrm{d}(n_{best}, (s_i, s_{i+1}))$ mellan närmaste nod och förflyttningens position på kurvan är mindre än eller lika med parametern D_{min} är noden tillräckligt nära kurvan och en båge kan skapas mellan n_{best} och föregående identifierade nod. Om istället $\mathrm{d}(n_{best}, c(s_i, s_{i+1})) > D_{min}$ skapas ingen båge till n_{best} då noden befinner sig för långt från kurvan och algoritmen fortsätter med nästa förflyttning. Ingen ny båge skapas om en båge redan existerar mellan de två noderna som ska sammankopplas, istället ökas vikten för den existerande bågen med ett.

I Figur 3-11 illustreras hur noderna identifieras och hur bågarna skapas längs en kurva för ett samplingspar hos en bana. Kurvans längd och storleken på ds ger upphov till fem förflyttningar längs kurvan.

Vid första förflyttningen är $n_{best} = n$ och $\mathrm{d}(n_{best}, c(s_i, s_{i+1})) \leq D_{min}$, ingen båge skapas eftersom föregående nod ej ännu har identifierats.

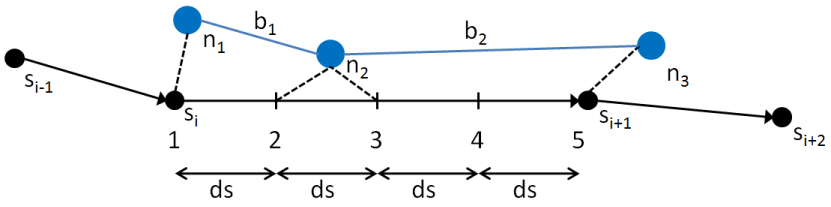
$n_{best} = n_2$ och $\mathrm{d}(n_{best}, c(s_i, s_{i+1})) \leq D_{min}$

Vid andra förflyttningen är $n_{best} = n_2$ och $\mathrm{d}(n_{best}, c(s_i, s_{i+1})) \leq D_{min}$, bågen b_1 skapas mellan n_{best} och föregående identifierade nod n_1.

Vid tredje förflyttningen är $n_{best} = n_2$ och $\mathrm{d}(n_{best}, c(s_i, s_{i+1})) \leq D_{min}$, ingen båge skapas eftersom n_{best} är samma som föregående identifierade nod.

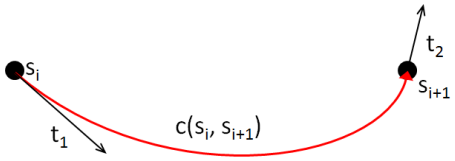
Vid fjärde förflyttningen är $n_{best} = n_3$ och $\mathrm{d}(n_{best}, c(s_i, s_{i+1})) > D_{min}$, ingen båge skapas eftersom n_{best} befinner sig för långt från förflyttningens position på kurvan.

Vid femte förflyttningen är $n_{best} = n_3$ och $\mathrm{d}(n_{best}, c(s_i, s_{i+1})) \leq D_{min}$, bågen b_2 skapas mellan n_{best} och föregående identifierade nod n_2.



Figur 3-11: Identifiering av noder att sammankoppla.

I vår första implementering valde vi att beskriva kurvan $c(s_i, s_{i+1})$ som en rät linje mellan samplingarna hos samplingsparet. Ett problem blev då att hitta ett bra värde för parametern D_{min}. Ett för litet parametervärde medför att algoritmen ej identifiera noder som borde sammankopplas, speciellt om avståndet mellan samplingarna hos samplingsparet är stort. Samtidigt medför ett för stort parametervärde att noder som befinner sig långt från kurvan identifieras och noderna blir felaktigt sammankopplade. En lösning på problemet är att beskriva $c(s_i, s_{i+1})$ på sådant sätt att kurvan bättre stämmer överens med tågets verkliga förflyttning mellan samplingsparet. Då samplingarna innehåller information om tågets fart och färdriktning valde vi att undersöka om informationen kan användas för att beskriva kurvan mellan två samplingar på ett bättre sätt.



Figur 3-12: Kurvan mellan samplingarna som funktion av samplingarnas fart och färdriktning

Vi fann att en Hermite-kurva kan användas för att beskriva kurvan och genom att testa olika sätt för beräkning av tangentvektorerna t_1 och t_2 fann vi att följande formel var lämplig:

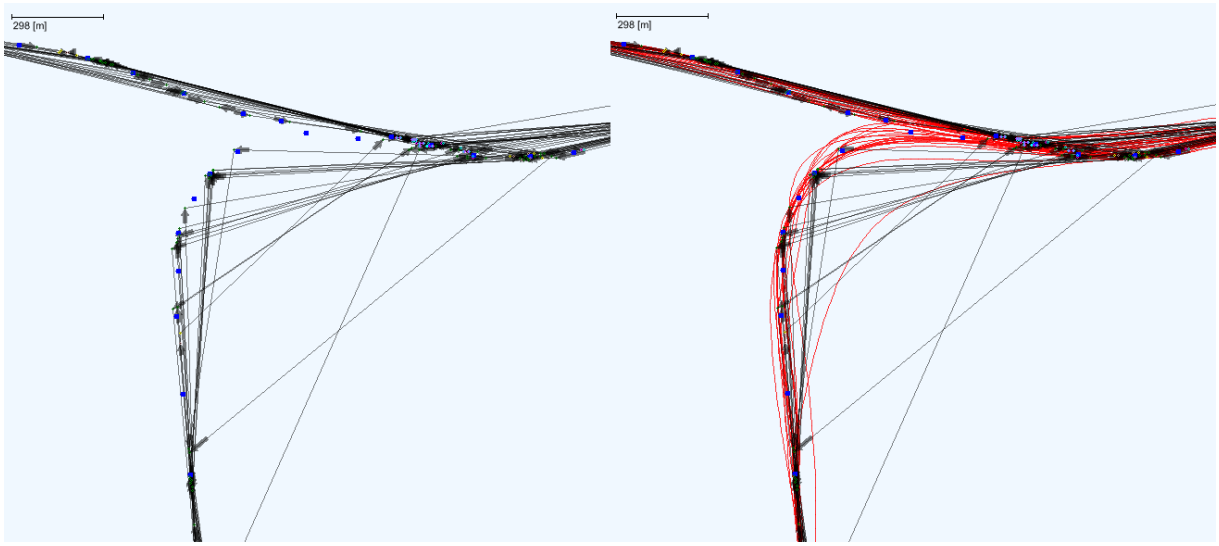
$$\angle t_1 = a_1, \angle t_2 = a_2, |t_1| = |t_2| = \left(\frac{\frac{d(s_i, s_{i+1})}{2}}{1.6}\right)\left(1 + \frac{\Delta\alpha}{60°}\right)^2$$

där a_1 är färdriktningen för sampling s_1, α_2 är färdriktningen för sampling s_{i+1} och $\Delta\alpha$ är minsta vinkelskillnaden mellan α_1 och α_2 i antal grader.

Ett problem är att en sampling från ett tåg utan fart får alltid har $\alpha = 0°$, vilket medför att kurvan får felaktiga tangentvektorer om någon av samplingarna saknar fart. Vi löste problemet genom att beräkna α_1 som vinkeln för tangenten hos s_i mellan de räta linjerna som från s_i till s_{i-1} och s_i till s_{i+1}. Metoden illustreras i Figur 3-13 där vinkeln för tågets färdriktning α_1 beräknas för samplingen s_i som saknar fart.

Figur 3-13: Beräkning av tågets färdriktning för en sampling som saknar fart.

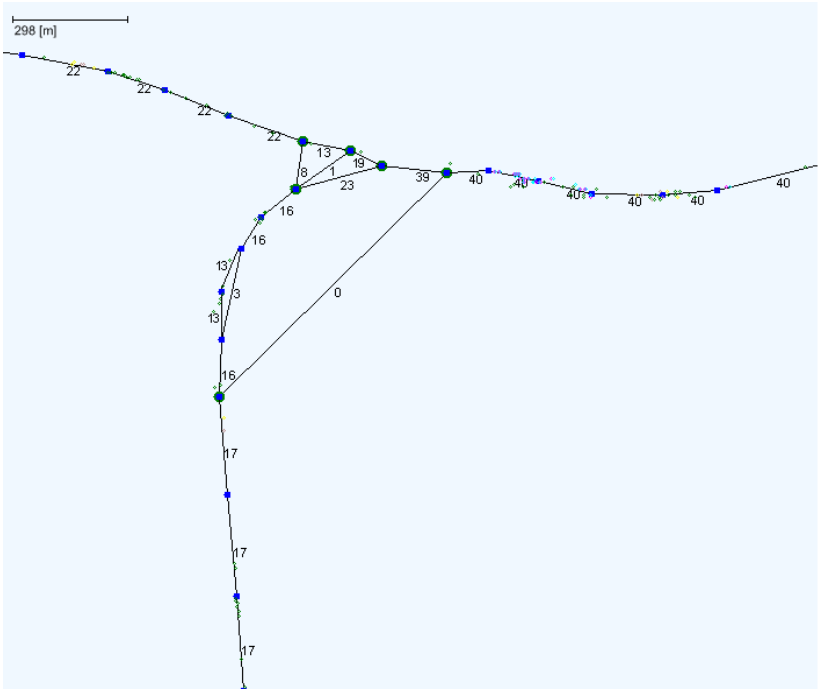
Beräkning av tågets färdriktning med hjälp av ovan beskriven metod misslyckas om s_i är den första eller sista samplingen hos banan. Om $s_i = s_1$ valde vi att beräkna α_i som vinkeln för vektorn mellan s_i och s_{i+1}, på motsvarande sätt om $s_i = s_n$ beräknas α_i som vinkeln för vektorn mellan s_{n-1} och s_n.



Figur 3-14: Ett antal banor samt Hermite-kurvorna mellan banornas samplingspar.

Figur 3-14 visar ett antal banor, samt Hermite-kurvorna, som beskriver tågets färdväg mellan banornas samplingspar. Av Figur 3-14 framgår att Hermite-kurvorna stämmer bättre överrens med tågets verkliga färdväg än vad linjära kurvor hade gjort och parametervärdet för D_{min} kan minskas.

Bågarna som skapas genom att identifiera och sammankoppla noder längs Hermite-kurvorna i Figur 3-14 visas i Figur 3-15 där även bågarnas vikt framgår.



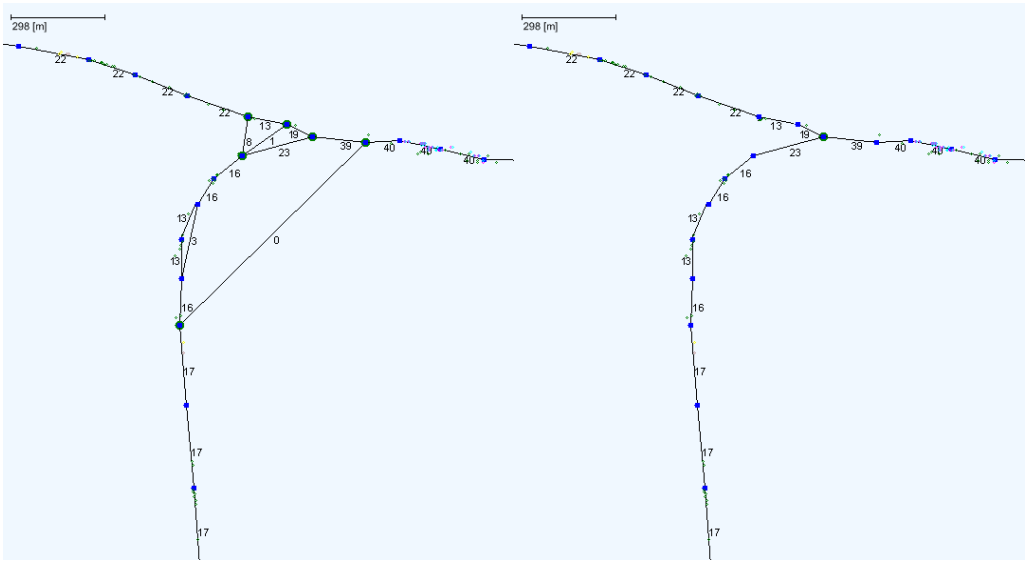
Figur 3-15: Bågar som skapats och dess vikt.

Av Figur 3-15 framgår att för många bågar har skapats, vilket framför allt beror på att Hermite-kurvorna beskriver felaktiga färdvägar mellan banornas samplingar. Av bågarnas vikt framgår hur många banor som identifierat bågen. Algoritmens nästa steg är att identifiera vilka bågar varje nod ska behålla.

3.5.3 Identifiera vilka bågar varje nod ska behålla

Genom att låta varje nod välja de två av dess bågar med högst vikt, och öka populariteten för dessa bågar, kan algoritmen identifiera vilka bågar som ska behållas och vilka som ska tas bort. Från början saknar alla bågar popularitet. Grafens noder analyseras och varje nod identifierar sina två bågar med högst vikt, de identifierade bågarnas popularitet ökar med ett. När alla noder har analyserats tas de bågar bort som saknar popularitet.

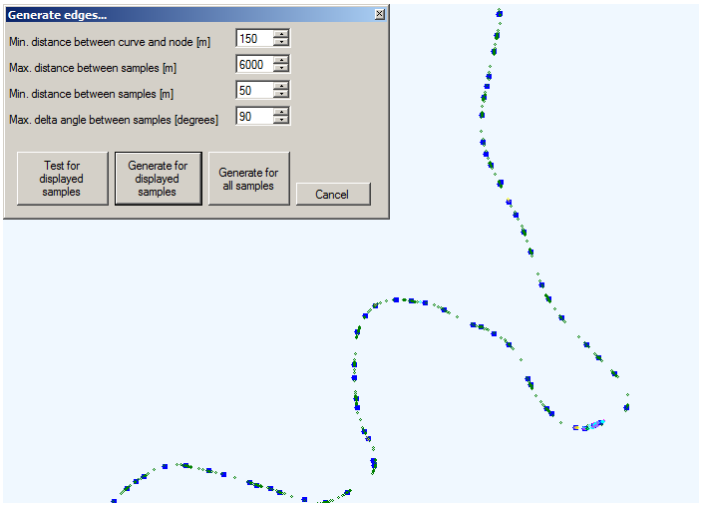
En nod som har exakt två bågar kommer alltid identifiera dess två bågar och öka deras popularitet, dessa bågar får därför alltid minst ett som popularitet. Sådana bågar tas bort om de inte även har identifierats av någon annan nod, bågen tas alltså bort om popularitet är ett. Ett exempel på resultatet efter identifiering av bågar att behålla illustreras i Figur 3-16.



Figur 3-16: Den högra grafen är resultat efter identifiering av bågar att behålla.

3.5.4 Verktyg för generering av bågar

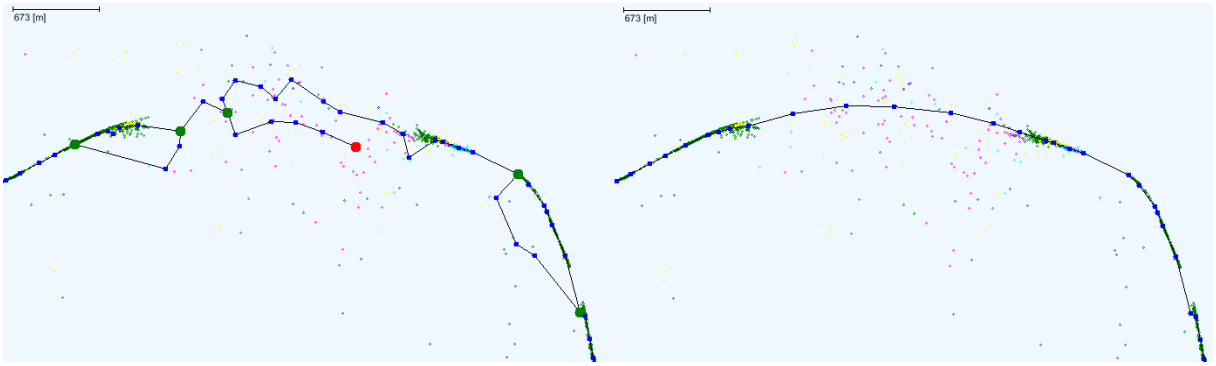
Algoritmen implementerades i *RailwayEditor* där användaren kan specificera parametervärden för D_{min}, d_{max}, d_{min}, a_{min} och illustreras i Figur 3-17. Parametern ds sattes till 25 meter.



Figur 3-17: Gränssnitt i *RailwayEditor* för generering av bågar.

Med hjälp av verktyget kan användaren välja att generera bågar för de samplingar som visas på skärmen eller att generera bågar för samtliga samplingar som är laddade i *RailwayEditor*.

För områden där samplingarna har felaktiga positioner, t.ex. vid tunnlar, skapas felaktigt noder där det inte finns någon räls i verkligheten. De felaktiga noderna sammankopplas ibland med bågar vilket medför att modellen blir felaktig. Vi korrigerade dessa fel manuellt genom att ta bort och lägga till noder och bågar i *RailwayEditor*. Ett exempel på en sådan korrigering visas i Figur 3-18.



Figur 3-18: Felaktig modell (vänster) och manuellt korrigerad modell (höger).

3.6 Resultat - Järnvägsmodell

Vi valde att basera järnvägsmodellen på samplingar mellan 2009-09-01 t.o.m. 2009-10-31 från 83 tåg som trafikerar det svenska järnvägsnätet, se Appendix 8.1 för mer information.

Noderna genererades med följande parametervärden:
$R_{max} = 100, R_{stop} = 1000, R_{inc} = 100, N_{min} = 10$

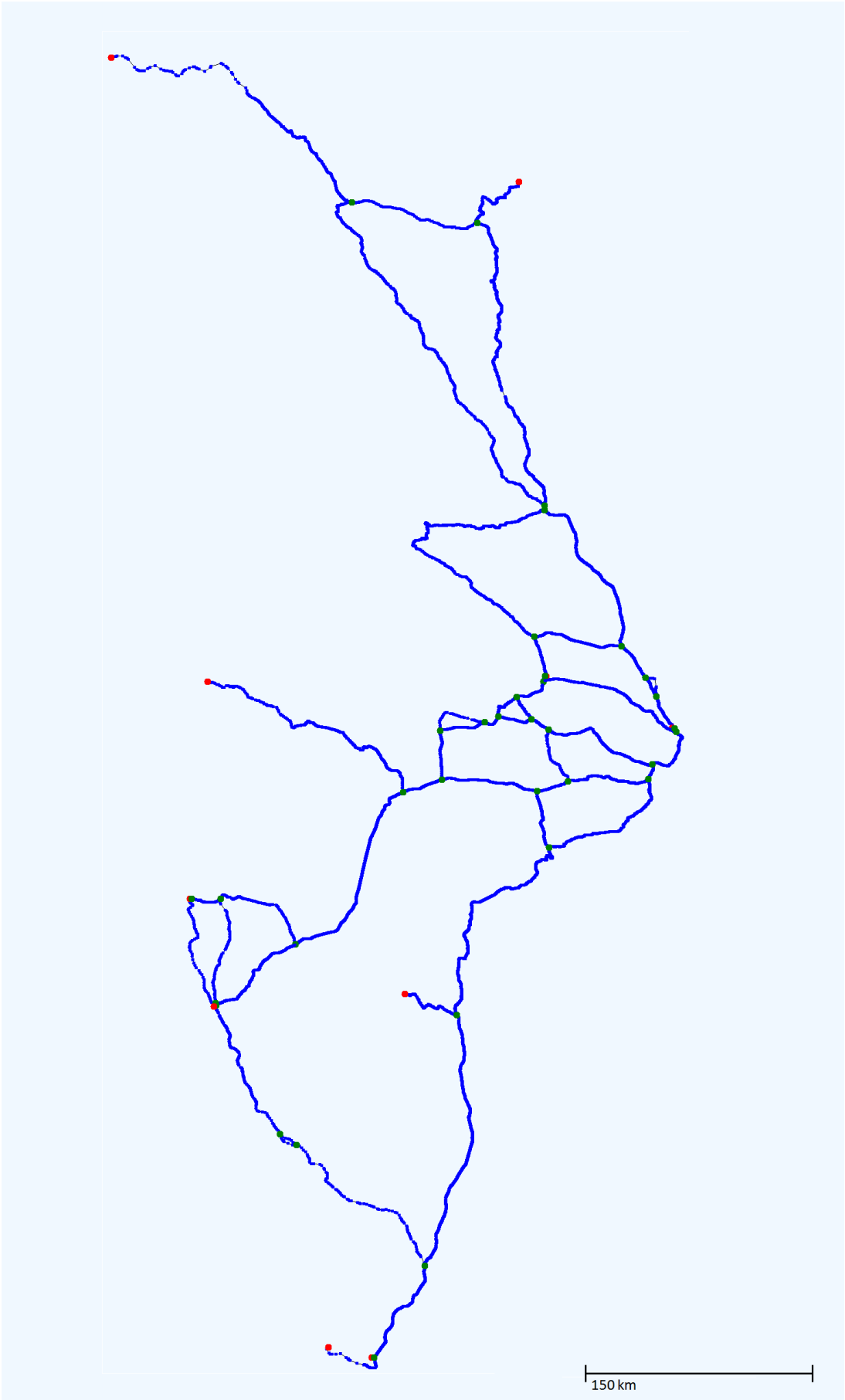
Värdet för R_{max} begränsar minsta avstånd mellan noderna, och därmed upplösningen, till 100 meter vilket innebär att parallella spår som ligger närmare än 100 meter från varandra representeras som ett enda spår.

Bågarna genererades med följande parametervärden:
$D_{min} = 150$, $d_{max} = 6000$, $d_{min} = 50$, $a_{min} = 90°$

Efter att manuellt korrigerat felaktigheter hos grafen fick den slutgiltiga modellen följande egenskaper.

- Antal noder: 7656
- Antal bågar: 7669
- Total längd: 3563 km
- Upplösning: 465 m
- Antal växlar: 36
- Antal slutpunkter: 10

En översiktsbild av modellen visas i Figur 3-19 där man kan se järnvägsnätets geografiska utsträckning för den räls som tågen trafikerat under de aktuella datumen.



Figur 3-19: Översiktsbild av järnvägsmodellen.

3.7 Slutsatser och diskussion - Järnvägsmodell

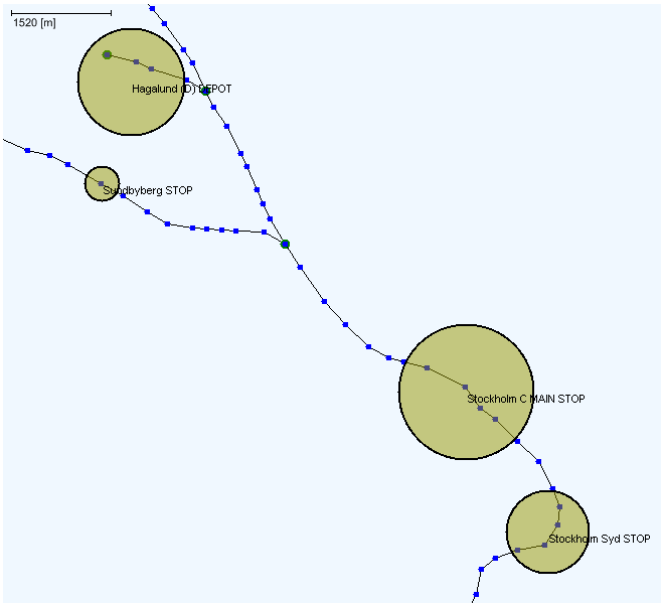
När vi påbörjade arbetet med att skapa järnvägsmodellen visste vi inte vilka krav som ställdes på modellen och vi visste inte vilken kvalité modellen behöver för att kunna användas till kartläggning och prediktion av tågrutter. Detta medförde att vi under för lång tid fastnade vid denna del av examensarbetet. Hade vi gjort om examensarbetet skulle vi istället valt att manuellt skapa noderna och bågarna och lagt mer tid på kartläggning och prediktion av tågrutter. Under arbetets gång gjorde vi förbättringar av järnvägsmodellen när vi fått en tydligare bild av vad som krävdes av vår järnvägsmodell.

Man kan ifrågasätta syftet med att skapa järnvägsmodellen med hjälp av samplingar från tågen eftersom järnvägsnätet redan finns dokumenterat i till exempel kartor. En anledning till att basera modellen på Icomeras mätdata är att Icomera då får ett verktyg för att enkelt skapa kartor där deras system används.

3.8 Generera stationer

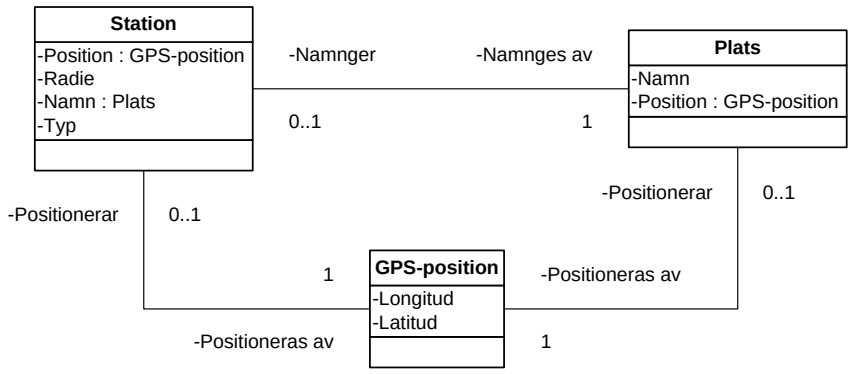
3.8.1 Modell för station

Vi valde att representera en station som ett cirkulärt område med en position för stationens mittpunkt och en radie kring denna. Stationer klassificeras med tre olika typer som är *station*, *huvudstation* eller *depå*. Stationerna skapas i *RailwayEditor* och visas på kartan. Ett exempel på några stationer visas i Figur 3-20 där Sundbyberg har typen *station*, Stockholm C har typen *huvudstation* och Hagalund har typen *depå*. Stationstypen har betydelse för kartläggningen av tågens rutter och beskrivs i *Kapitel 4*.



Figur 3-20: Stationer på kartan i *RailwayEditor* för ett område kring Stockholm.

Stationens namn härleds från en känd plats med ett namn som befinner sig inom stationens radie. Klassdiagrammet i Figur 3-21 beskriver stationsmodellens klasser och deras relationer.



Figur 3-21: Klassdiagram för stationsmodell.

3.8.2 Metod för generering av stationer

För att identifiera stationer, bestämma deras positioner och radier utvecklade vi en algoritm som använder sig av de samplingar som är laddade i *RailwayEditor*.

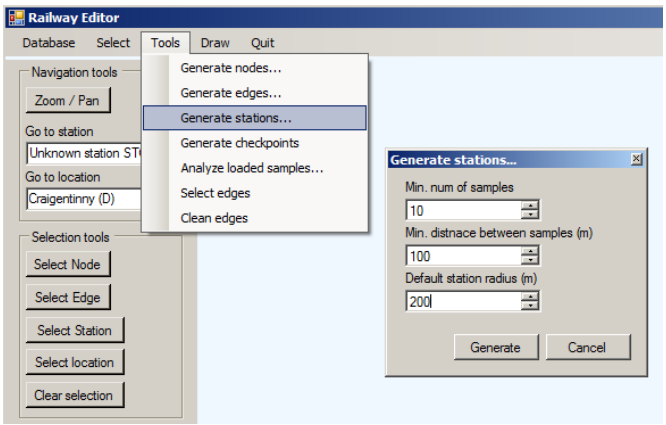
Algoritmens huvudsteg är:

- Användaren anger värden för parametrarna D_{max}, n_{min} och $r_{default}$..

1. Identifiera alla samplingar som har anledning *start*, *stop*, *time* eller där samlingens fart är noll.
2. Gruppera de identifierade samplingarna i kluster där varje sampling i klustret maximalt har avståndet D_{max} till någon annan sampling i samma kluster.
3. De kluster som innehåller färre samplingar än n_{min} tas bort.
4. Varje resterande kluster bildar en station där stationens position beräknas som medelpositionen av klustrets samplingar och stationens radie beräknas som avståndet mellan medelpositionen och den sampling i klustret som befinner sig längst från medelpositionen.
5. Om stationens radie är mindre än $r_{default}$ sätts radien till$r_{default}$.
6. Algoritmen undersöker sedan om det finns någon känd plats inom stationens radie och namnger i så fall stationen med platsens namn. Om det inte finns någon känd plats får stationen ett standardnamn som är "Unknown station".
7. Stationstypen sätts till det förvalda vädret *station*.

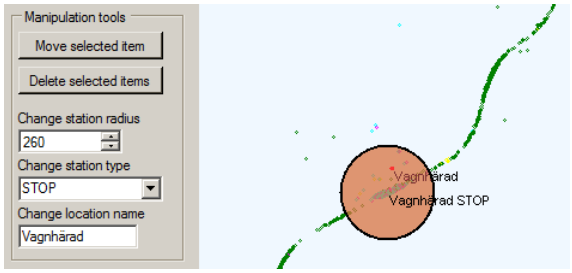
3.8.3 Verktyg för generering av stationer

Algoritmen implementerades i *RailwayEditor* där användaren kan starta generering av stationer och ange parametervärden för D_{max}, n_{min} och $r_{default}$. Användargränssnittet illustreras i Figur 3-22.



Figur 3-22: Användargränssnitt för generering av stationer.

För att namnge stationerna valde vi att använda ett antal namngivna platser med kända positioner som Icomera tillhandahöll, se Appendix 8.2 för mer information. Stationstypen får det förvalda värdet *station*, men kan manuellt ändras till *huvudstation* eller *depå* i *RailwayEditor*. Applikationen har även stöd för att manuellt ändra stationens position och radie samt att skapa nya platser eller byta namn på existerande platser. Användargränssnittet för dessa verktyg i visas i Figur 3-23.



Figur 3-23: Användargränssnitt i *RailwayEditor* för manipulering av stationer och platser.

3.9 Resultat - Stationer

Vi valde att generera stationer för samplingar med mer än 4 satteliter mellan datumen 2009-09-01 t.o.m. 2009-10-31 från SJ-tåg som trafikerar det svenska järnvägsnätet (se *Appendix 8.1 Identitetsnummer för analyserade tåg* för mer information) med följande parametervärden: $D_{max} = 100$, $n_{min} = 10$ och $r_{default} = 200$.

Totalt identifierades 208 stationer där 86 stationer kunde kopplas till en känd plats som inom stationsradien och 120 okända stationer som inte kunde kopplas till någon känd plats och därmed fick det förvalda namnet *Unknown station*.

Av de 120 okända stationerna kunde 34 avfärdas som felaktiga då dessa låg utanför järnvägsmodellen och sannolikt baserades på samplingar med felaktiga positioner. Resterande av de 86 okända stationerna utvärderades manuellt genom att slå upp stationens position i ett kartverk (Eniro Sverige AB, 2009) och verifiera om det fanns en tågstation vid platsen eller inte. Utvärderingen resulterade i att 17 av de 86 okända stationerna hade en tågstation i kartverket och vi valde behålla dessa stationer och namnge dem med platsens namn enligt kartverket.

Positionen för de resterande 69 okända stationerna saknade station enligt kartverket och togs bort. En stor del av dessa stationer kan vara platser där tågen stannar i väntan på grön signal eller mötesplatser där tåg kan passar varandra. Ingen djupare analys av dessa platser utfördes.

Vi valde alltså att behålla totalt 103 stationer och dessa sparades i databasen, se *Appendix 8.3 Identifierade stationer* för mer information. Nästa steg var att klassificera stationerna genom att bestämma deras typ vilket gjordes baserat på vår och Icomeras kunskap om stationerna. De största stationerna, Stockholm, Göteborg, Malmö och Köpenhamn fick typen *huvudstation*. Vi valde också att klassificera stationer som befann sig vid slutet på en järnvägsstäcka som *huvudstation* då tåg som åker till en sådan station ej rimligtvis kan ha någon annan station destination efter att ha anlänt till stationen. Ett antal stationer klassificerades som *depå*, framförallt baserat på Icomeras kunskap om stationerna men även genom att slå upp stationens position i kartverket och utvärdera om stationen är en depå eller ej. De övriga stationerna fick behålla den förvalda typen *station*. I Tabell 3-1 visas en sammanställning av klassificeringen av de identifierade stationerna.

Stationstyp	Antal
Station	86
Huvudstation	10
Depå	7

Tabell 3-1: Klassificering av identifierade stationer.

3.10 Slutsatser och diskussion - Stationer

Resultatet visar att algoritmen kan användas för att identifiera platser där tågen brukar stanna men att platserna inte nödvändigtvis är stationer i verkligheten. Mer information om verkligheten, i form av en t ex en karta, behövs för att verifiera om de identifierade platserna verkligen är stationer eller ej. En alternativ lösning kan vara att manuellt skapa alla tågstationer baserat på information från Banverket eller en karta. Algoritmen skulle sedan kunna användas för att identifiera vilka av dessa stationer som tågen stannar vid och endast använda dessa stationer i modellen.

Vi valde att behålla cirka 50 % av de stationer som identifierades. Det innebär att ungefär hälften av de identifierade stationerna inte är stationer i verkligheten utan är andra platser där tåg brukar stanna, t.ex. stoppsignaler vid växlar och tunnlar och mötesplatser där tåg kan passera varandra. Även om vi valde att exkludera dessa platser i modellen så kan de ändå vara intressanta. En utökad modell skulle t.ex. kunna använda dessa platser för att identifiera tåg som har stannat vid en signal och estimera hur länge tåget måste vänta innan resan kan fortsätta.

Stationerna klassificerades manuellt baserat på vår och Icomeras kunskap om olika stationer. Vi har inte analyserat möjligheten att automatisk klassificera stationerna, men en möjlig lösning kan vara att identifiera karaktäristiska parametrar för varje stationstyp (*station*, *huvudstation* och *depå*) och låta algoritmen klassificera stationerna.

4 Kartläggning av rutter

Syftet med att kartlägga tågens rutter är att använda dem som underlag vid prediktion av pågående rutter. De rutter som kartlagts skulle även kunna användas till andra ändamål, till exempel generering av tidtabeller för att utvärdera av hur väl tågen hållit sig till de officiella tidtabellerna.

4.1 Problembeskrivning

Vi definierar en rutt som en sorterad mängd av stationer som ett tåg har passerat eller stannat vid. Den första stationen i mängden betecknar ruttens startstation och sista stationen betecknar ruttens slutstation. Rutten innehåller även information om ankomst- och avgångstid för stationerna som ingår.

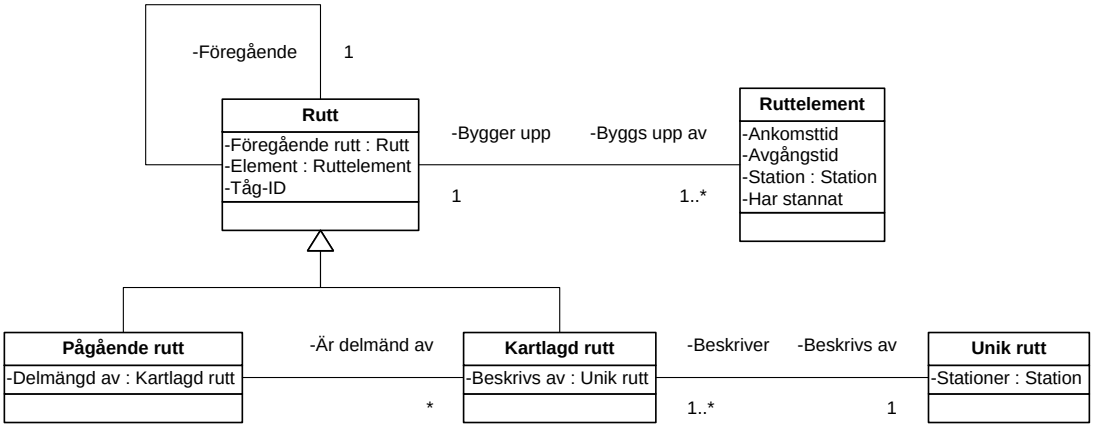
Problemet består i att, på ett så korrekt sätt som möjligt, kartlägga tågens rutter i realtid, det vill säga medan tågen trafikerar dem. Till vår hjälp har vi järnvägsmodellen och samplingarna som skickas från tågen. Svårigheten ligger i att identifiera rutternas startstationer, stationerna som tågen passerar och stannar vid, samt identifiera när rutterna avslutas.

Vi undersökte olika metoder för att kartlägga rutterna. Den första metoden vi undersökte tog enbart hänsyn till stationerna och använde inte järnvägsmodellen som sammanbinder stationerna. Problemet vi hade med denna metod var att vi inte kunde identifiera alla stationer som tågen passerade. Detta eftersom tågen oftast skickar samplingar med mellanrum på 1 km och ibland skickades ingen sampling då tågen befann sig vid stationerna som passerades.

Vi valde istället att monitorera hur tågen trafikerar järnvägsmodellen med hjälp av en tillståndsmodell som beskriver tågens beteenden. Metoden gör det möjligt att identifiera de stationer som ett tåg har passerats även om avståndet mellan två samplingar är stort.

4.2 Datamodellen för en rutt

I vår datamodell byggs rutten upp av en sorterad lista med ruttelement. Ett ruttelement beskriver en station som ingår i rutten, när tåget ankom och avgick från stationen samt om tåget stannade på stationen eller ej. För en rutt sparar vi även information om anledningen till att vi började bygga upp rutten och anledningen till att vi ansåg den avslutad. Vi sparar även en flagga som benämner om tåget gjort ett hopp längre än 10km under ruttens gång.



Figur 4-1 Klassdiagram över datamodellen för en rutt

Vi särskiljer även kartlagda rutter och unika rutter. Unika rutter beskrivs med en ordnad lista på stationer och är oberoende av tid, till skillnad från beteckningen kartlagd rutt som även medför att rutten färdats vid en viss tid på dagen och stannat vid vissa stationer. Kartlagda rutter som passerar samma stationer kan beskrivas av samma unika rutt. Ett klassdiagram över vår modell ses i Figur 4-1.

4.3 Järnvägsmodell, användning

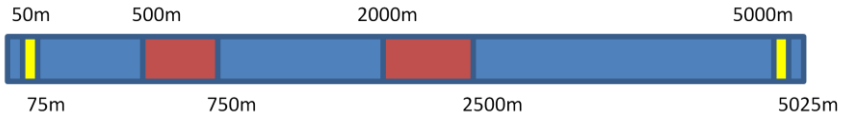
Vi använder här vår järnvägsmodell för att avgöra tågets riktning och huruvida tåget har vänt på en rälssträcka. För att underlätta beräkningar har vi delat in rälsen i segment vilka avgränsas av korsningar, ett tågs position kan då beskrivas med ett rälssegment samt avstånd ifrån segmentets första ändpunkt. Indelningen i segment hjälper oss med att representera både tågens och stationernas positioner.

Segmentering av järnvägen

För att underlätta komplexiteten i de beräkningar vi utför har vi delat in järnvägsmodellen i olika segment. Segmenten representerar delar av järnvägsmodellen och avgränsas av korsningar. Ett segment innehåller information om vilka bågar och noder som ingår, och varje nod och båge innehåller information om vilket segment den tillhör. Den nod som utgör en korsning benämner vi segmentnod, och denna nod ingår i alla segment som leder fram till, eller bort från, korsningen. Vi får även en möjlighet att beskriva järnvägsnätet som en förenklad graf där bågarna har vikter som representerar avstånd mellan segmentnoderna.

Stationerna representerade på segmentet

Ett segment representeras som tidigare nämnt av alla de noder och bågar som ingår i segmentet. I ett segment ingår även en uppdelning av segmentet i två typer av segmentelement. Ett segmentelement är ett objekt som innehåller information om hur långt in på segmentet elementet tar vid, hur långt in på segmentet elementet slutar, samt information om elementet representerar en vanlig rälssträcka eller om det är en station. I de fall då elementet är en station innehåller även objektet information om stationen i fråga. Ett segment kan till exempel grafiskt representeras som i Figur 4-2, där blått representerar vanlig rälssträcka, gult representerar en liten station, och rött representerar en större station. Den övre siffran visar starten för det närmast angränsande stationselementet, och den undre visar slutet för det närmast angränsande stationselementet.



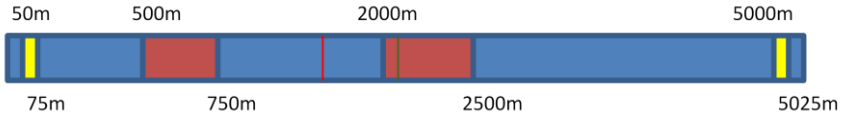
Figur 4-2: Ett segment i järnvägsmodellen, de röda och gula fälten representerar stationer.

Projicering av GPS-position på järnvägen

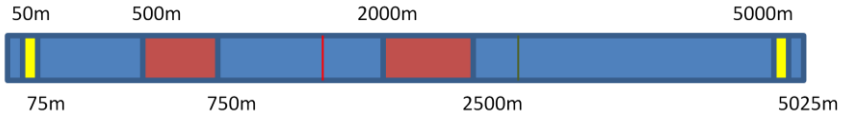
För att sammankoppla vår uppdelning av järnvägen i segment med GPS-positioner försöker vi projicera varje inrapporterad GPS-position på en båge i vår järnvägsmodell. Detta gör vi genom att undersöka samplingens vinkelräta avstånd till den närmsta bågen. I de fall då avståndet är mindre än 1000 meter projicerar vi samplingens position på bågen och använder den nya positionen för att representera samplingens position, den nedre blå samplingen i figuren. Skulle det vinkelräta avståndet till närmsta båge vara större än 1000 meter, som den övre blå samplingen i figuren, projicerar vi positionen på närmsta nod, ifall avståndet där är mindre än 1000 meter.

Samplingens position projicerad på en båge ger oss möjlighet att istället för att representera samplingens position med longitud, latitud och altitud kan vi

representera tågets position med en båge samt avstånd ifrån bågens ena kant. Eftersom vi även har information om vilket segment varje båge tillhör kan vi även representera samplingens position med ett segment samt avstånd ifrån segmentets ena kant. Med hjälp av de element som segmenten är indelade i vet vi om tåget befinner sig på en station eller på en rälssträcka. Vi kan även med lätthet avgöra om ett tåg har passerat en station utan att skicka någon sampling inom stationens radie genom att jämföra tågets föregående position projicerat på ett segment jämfört med den nuvarande projicerade positionen. Exempelvis studerar vi i Figur 4-3 tågets nuvarande position, representerat med ett rött streck, samt tågets föregående position, representerat med ett grönt streck. Vi ser med lätthet att tåget just lämnat en station, även om den föregående positionen skulle skickats vid 2600m på segmentet som vi observerar i Figur 4-4 skulle vi se att tågets passerat en station.



Figur 4-3: Bild av segment av järnvägen som illustrerar ett tåg som just lämnar en station.

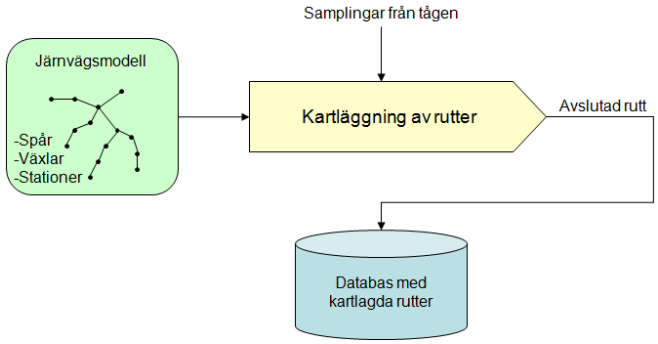


Figur 4-4: Bild av segment av järnvägen som illustrerat ett tåg som passerat en station utan att ha skickat någon sampling.

I de fall då tågens föregående och nuvarande position projiceras på olika järnvägssegment gör vi ett antagande att tågen alltid tar den kortaste vägen mellan två segment och använder oss av Dijkstra's algoritm för att avgöra vilka segment, och därmed eventuella stationer, som ett tåg passerat.

4.4 Process för kartläggning

Processen för kartläggning av rutter består av tre tillstånd. I starttillståndet identifierar processen ruttens startstation. När startstationen är identifierad går processen in i nästa tillstånd. Processen är kvar i detta tillstånd tills tåget avgår ifrån startstationen. Det tredje tillståndet består av kartläggning av rutten, med hjälp av järnvägsmodellen, samt identifiering av slutstation. När rutten avslutats sparas den i en databas och processen går tillbaks till starttillståndet. Figur 4-5 ger en övergripande bild av processen.

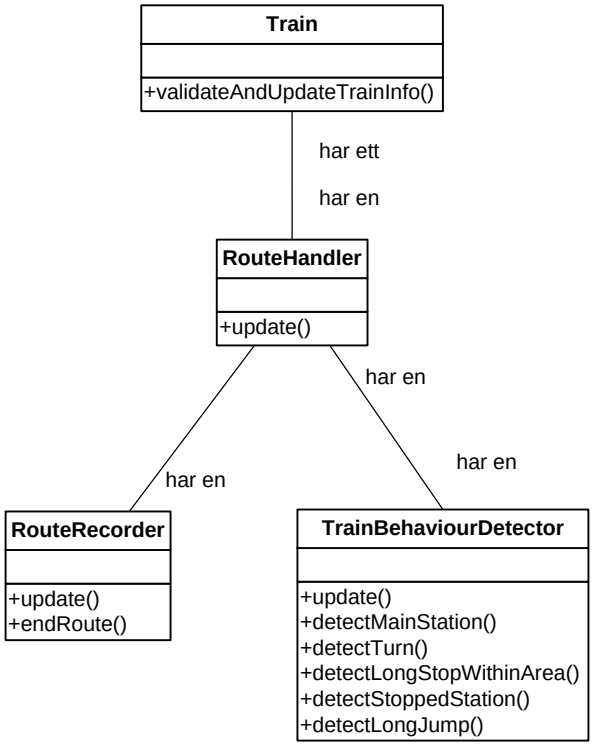


Figur 4-5: Övergripande bild av kartläggningsprocessen

Tågets rutt bestäms genom att kartlägga hur tåget färdas i järnvägsmodellen. Rutten innehåller information om vilken ordning tåget har passerat stationer. Genom att använda järnvägsmodellen kan vi identifiera vilka stationer ett tåg har passerat, även om avståndet mellan två på varandra följande samplingar är stort.

Nedan följer en ingående beskrivning av varje tillstånd och villkor för övergångar, deras namn och möjliga övergångar är även beskrivna i tillståndsdiagrammet Figur 4-6.

4.5 Beskrivning av implementationen



Figur 4-6: Klassdiagram för kartläggningsprocessen där de viktigaste delarna blir belysta

RouteRecorder.update()
Denna metod undersöker tågets föregående och nuvarande position och lägger till eventuella stationer som passerats i rutten.
RouteRecorder.endRoute()
Avslutar den pågående rutten och sparar den. Exakt hur rutten avslutas beror på anledning till att rutten ansetts avslutad. I de fall då rutten avslutats om tåget stått still länge på en station avslutas rutten och slutstationen sätts till den station tåget står still på. I de fall då rutten avslutas med anledning av en vändning på järnväg eller stopp i depå sätts slutstationen till den station som tåget senast stått still på.

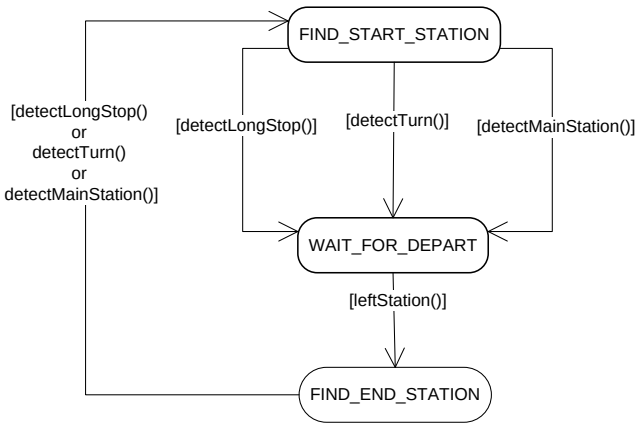
TrainBehaviourDetector.update()
Anropar flera olika metoder för att undersöka tågets beteende. Förutom att de metoder som listas nedan anropas så uppdateras även informationen om vilken station tåget för närvarande är på, eller precis har passerat i de fall då tåget inte skickar någon sampling när det befinner sig på stationen i fråga.
TrainBehaviourDetector.detectLongJump()
Avgör om tåget inte har skickat några samplingar under en längre sträcka genom att mäta avstånd mellan två på varandra följande samplingar från ett tåg. Vi

bedömer att vi har haft ett längre avbrott om vi inte fått in någon rapport på 10 km. När detta sker markeras rutten senare som en rutt där det skett ett avbrott i inrapporteringen.
TrainBehaviourDetector.detectMainStation()
Denna metod undersöker om tåget stått still eller haft en hastighet under 10 knop på en station som vi klassar som huvudstation. De stationer vi klassat som huvudstationer är Göteborg C, Köpenhamn C, Malmö C och Stockholm C.
TrainBehaviourDetector.detectTurn()
Denna metod undersöker om ett tåg har bytt riktning och nu är på väg åt motsatt riktning. För att avgöra detta använder vi oss av vår järnvägsmodell och undersöker vilken riktning tåget har haft på järnvägssegmentet.
TrainBehaviourDetector.detectLongStopWithinArea()
Denna metod undersöker om ett tåg har befunnit sig inom en area på 500m inom en längre tid. Vi anser att ett tåg har åkt in i en depå om detta har skett.
TrainBehaviourDetector.detectStoppedStation()
Avgör om tåget har stått still på en station.

Train.validateAndUpdateTrainInfo()
Denna metod läser in en sampling och validerar den gentemot föregående sampling. Därefter lägger den till samplingen i tågets lista på samplingar och kallar på *RouteHandler.update()*

RouteHandler.update()
Börjar med att anropa *TrainBehaviourDetector.update()* och analyserar därefter resultatet och uppdaterar tågets tillstånd.

4.6 Tillståndsmodellen



Figur 4-7: Tillståndsmodellen för kartläggningsprocessen

FIND_START_STATION
Starttillståndet för vår modell (som beskrivs i Figur 4-7). Ifrån detta tillstånd kan vi gå över till tillståndet WAIT_FOR_DEPART. I FIND_START_STATION används följande metoder:

TrainBehaviourDetector.detectMainStation()
Resulterar i positiva fall i att *RouteRecorder* registrerar den aktuella stationen som start station för rutten, och vi går över till nästa tillstånd, WAIT_FOR_DEPART.

TrainBehaviourDetector.detectTurn()
I de fall där tåget bytt riktning på en rälssträcka registreras nästa station där tåget senast stannat som start för rutten. I de fall där tågen bytt riktning på en station registreras den aktuella stationen som start för rutten. När startstationen har registrerats av *RouteRecorder* går vi över till tillståndet WAIT_FOR_DEPART.

TrainBehaviourDetector.detectLongStopWithinArea()
Följden av ett längre stop är detsamma som en vändning på en rälssträcka.

WAIT_FOR_DEPART
WAIT_FOR_DEPART är ett något enklare tillstånd än FIND_START_STATION och FIND_END_STATION.

TrainBehaviourDetector.update() avgör vilken station tåget för närvarande befinner sig på, och vi går över till FIND_END_STATION om tåget inte längre är kvar på den första stationen.

FIND_END_STATION
I det slutliga tillståndet avgörs om rutten är en giltig rutt. Det är även här som ruttens skapas. Tillståndet fick namnet FIND_END_STATION eftersom dess huvudsakliga logik innefattar att detektera slutstationen för en rutt. När tågens pågående rutt är i detta tillstånd registreras de stationer tåget passerar, tillsammans med stationens namn och identifikation registreras även tiden för ankomst, tiden för avgång samt om tåget stått still på den passerade stationen.

TrainBehaviourDetector.detectTurn()
I de fall då tåget har vänt på en rälssträcka registrerar *RouteRecorder* stationen tåget senast stannat på som slutstation för rutten, annars om tåget vänt på en station registrerar *RouteRecorder* stationen tåget senast stått still på som slutstation.

TrainBehaviourDetector.detectMainStation()
Resulterar i positiva fall i att *RouteRecorder* registrerar den aktuella stationen som slutstation för rutten.

TrainBehaviourDetector.detectLongStopWithinArea()
Följden av ett längre stop är densamma som en vändning på en rälssträcka.

I samband med att *RouteRecorder* avslutar rutten kontrollerar *RouteHandler* om rutten har samma slut- och startstation och kastar bort rutten om så skulle vara fallet. När rutten avslutas och blir kartlagd går processen tillbaks till tillståndet FIND_START_STATION.

Ett specialfall av station som också registreras är våra kontrollpunkter. Vid varje korsning finns det en kontrollpunkt, genom att kontrollera vilka kontrollpunkter tåget har passerat kan vi med lätthet och säkerhet avgöra vilken rälssträcka ett tåg väljer i en korsning. Kontrollpunkterna har vi automatiskt genererat och placerat i ändarna på de rälssegment där det inte finns någon station i närheten.

4.7 Exempel, Kartläggning av rutt

Exempel:
Ett tåg registreras först klockan 00:00, tåget står då inne på depån i Hagalund. Tåget fortsätter skicka rapporter ifrån Hagalund och när *TrainBehaviourDetector. detectLongStop Within Area()* har registrerat att tåget stått still i 30 minuter på depån byter vi tillstånd på tåget till FIND_START_STATION och börjar leta efter en startstation.

När tåget till slut klockan 06:00 rör sig ut ifrån Hagalund och åker in till Stockholm C registrerar *RouteHandler* med hjälp av *TrainBehaviourDetector.detectMainStation()* Stockholm C som startstation för rutten. Systemet byter nu tillstånd till WAIT_FOR_DEPART och börjar nu vänta på att tåget skall avgå. När den första rapporten där tåget befinner sig utanför stationen kommer in registrerar *RouteRecorder* avgångstiden för första stationen. Därefter börjar *RouteHandler* kartlägga tågets rutt tills det att tåget har nått sin slutstation.

4.8 Resultat - Kartläggning av rutter

Genom att använda metoden som finns beskriven i detta kapitel registrerar vi 16897 kartlagda rutter. Detta medför totalt 423198 ruttelement, det vill säga varje rutt är i snitt uppbyggd av 25 ruttelement. Antal unika rutter som registreras är 487 och antal start och slutstationer vi registrerar är 87 stycken. Vi undersöker vidare i resultatavsnittet hur stor del av de rutter vi hittat som är korrekta. Vi visar i

Tabell 4-1 upp ett exempel på informationen vi sparar i en rutt.

Ankomst	Station	Avgång	Stannat
00:00	Göteborg C	06:44	Ja
07:06	Alingsås	07:07	Nej
07:21	Herrljunga	07:23	Ja
07:34	Falköping	07:34	Nej
07:47	Skövde C	07:48	Ja
08:02	Töreboda	08:03	Nej
08:27	Hallsberg	08:27	Nej
08:42	Vingåker	08:43	Nej
08:53	Katrineholm C	08:54	Ja
09:03	Flen	09:04	Nej
09:20	Gnesta	09:20	Nej
09:31	Södertälje syd	09:33	Ja
09:36	Malmsjö	09:37	Nej
09:42	Flemingsberg	09:43	Nej
09:51	Stockholm Syd	09:52	Nej
09:53	Stockholm C	00:00	Ja

Tabell 4-1: Exempel på informationen som vi sparar om en rutt presenterad på tabellform

4.8.1 Unika rutter

Vi benämner en unik rutt som en sträcka mellan två stationer vilken passerar ett antal stationer och kontrollpunkter. En unik rutt är en återkommande rutt som färdas upprepade gånger. Vi hittar 487 olika unika rutter. För att avgöra hur pålitlig en unik rutt är undersöker vi hur ofta ett tåg färdas på den. Vi gör dock ingen filtrering av unika rutter som vi anser vara opålitliga eftersom vår rutt-predikteringsalgoritm tar hänsyn till alla kartlagda rutter, och de avgångar som ger upphov till en opålitlig unik rutt är små till antalet i sammanhanget. Vi gör ett antagande att rutter som förekommer åtminstone 1 gång per vecka under den 8 veckors period vi har undersökt är pålitliga. Av de totalt 16897 avgångarna som har lett till registrerade rutter har 692 gett upphov till unika rutter som har förekommit färre än 8 gånger på en 8 veckors period. Därmed bedömer vi 4 % av de avgångar vi registrerar som opålitliga.

Vi kan även kolla på de mycket pålitliga rutterna vilka vi antar vara de som förekommer 3 eller fler gånger per vecka under en 8 veckors period. Detta är 15410 stycken, d.v.s. 91 %.

I Tabell 4-2 presenteras topp 10 av de mest pålitliga rutterna, siffran betecknar antal avgångar som färdats den unika rutten.

Startstation	Slutstation	Avgångar
Göteborg C	Stockholm C	885
Stockholm C	Göteborg C	879
Södertälje Syd	Stockholm C	689
Stockholm C	Södertälje Syd	679
Malmö C	Stockholm C	624
Västerås	Stockholm C	608
Stockholm C	Malmö C	583
Stockholm C	Västerås	562
Stockholm C	Linköping C	494
Malmö C	Köpenhamn H	491

Tabell 4-2: Topp 10 av de mest pålitliga rutterna.

4.8.2 Möjliga felorsaker, förbättringar

Vi har upptäckt möjliga fel med vårt tillvägagångssätt för kartläggning av rutter, saker som vi på grund av tidsbrist inte hinner gå in i detalj på inom ramen för detta examensarbete.

Vi studerar en av rutterna mellan *Stockholm C* och *Malmö C* och de stationer som passeras. Denna rutt förekommer 583 gånger under perioden som observerats.

Stockholm C, Stockholm Syd, Flemingsberg, Malmsjö, Södertälje Syd, Gnesta, Flen, Katrineholm C, Norrköping, Linköping C, Linköping Depot, Mjölby, Tranås, Nässjö C, Alvesta, Hässleholm C, Lund C, Malmö vagnhall, Malmö C

Bland rutterna finner vi andra unika rutt som är en delmängd av rutten ovan. Ett exempel är rutten *Stockholm C* till *Alvesta*. Rutterna innehåller samma stationer från *Stockholm C* fram till *Alvesta*. Den kortare rutten förekommer 24 gånger under perioden som observerats.

Stockholm C, Stockholm Syd, Flemingsberg, Malmsjö, Södertälje Syd, Gnesta, Flen, Katrineholm C, Norrköping, Linköping C, Linköping Depot, Mjölby, Tranås, Nässjö C, Alvesta

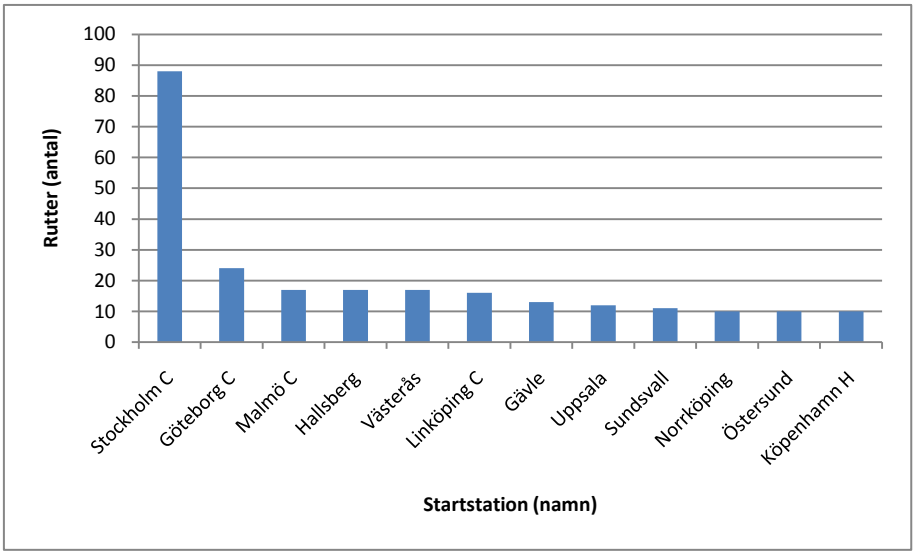
Bland rutterna finner vi även en unik rutt som börjar i *Alvesta* och slutar i *Malmö C*. Dock förekommer denna rutt endast 13 gånger under den aktuella perioden.

Alvesta, Hässleholm C, Lund C, Malmö vagnhall, Malmö C

Detta kan bero på att vissa av avgångarna från *Stockholm C* till *Malmö C* får stå still och vänta en stund längre än vad vårt antagande tillåter vid skapandet av rutterna. Skulle vi ändra på vårt antagande angående hur lång tid ett tåg står still på en station när den kartlagt en rutt är det en risk att detta skapar fler opålitliga rutter för oss på en annan sträcka. En möjlig lösning på detta skulle kunna vara att observera föregående rutt för det aktuella tåget för att se om den aktiva rutten tillsammans med den föregående är en frekvent förekommande rutt.

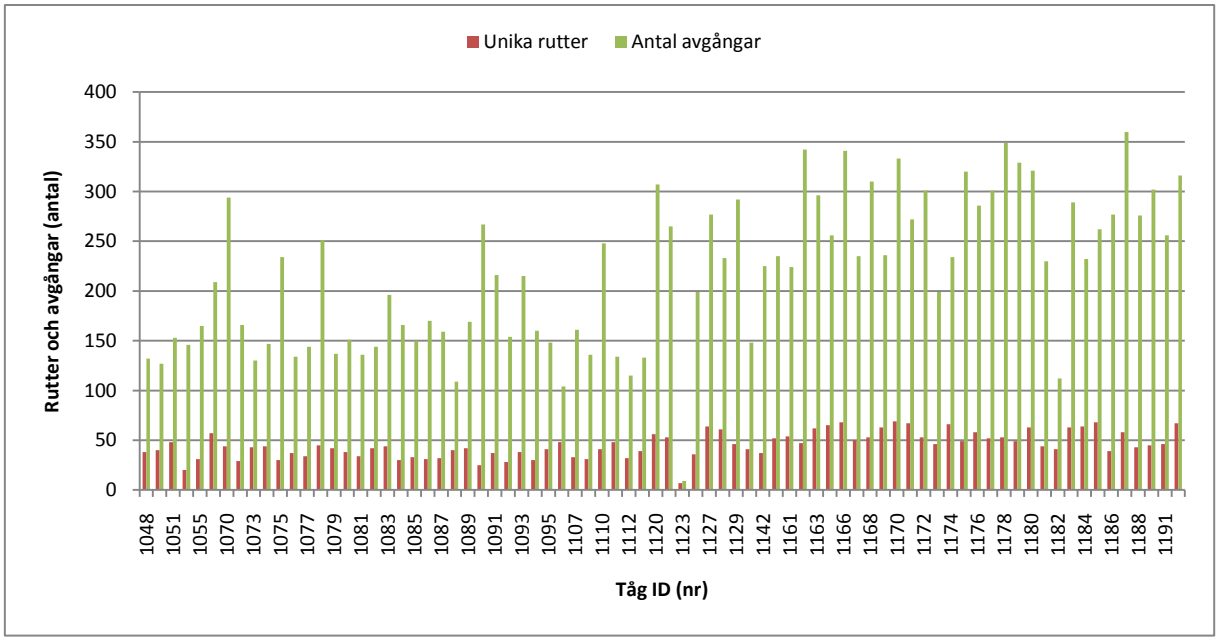
4.8.3 Möjligheter för prediktion

För att undersöka möjligheten för prediktion av rutter har vi försökt undersöka vilka parametrar som tidigt låter oss avgöra vilken rutt ett specifikt tåg kör. Vi börjar med att undersöka den informationen vi har tillgång till i början av rutten, nämligen startstation och starttid. Vi har även tillgång till information om vilken individ vi observerar samt historisk information om individen, som till exempel föregående rutt.
Vi observerar antal möjliga rutter och hur de kan bero på start station. Vi väljer att i Figur 4-8 enbart visa start stationer som kan ge upphov till minst 10 olika rutter för att demonstrera att vi kan utesluta många rutter enbart på vilken start station vi utgår ifrån.



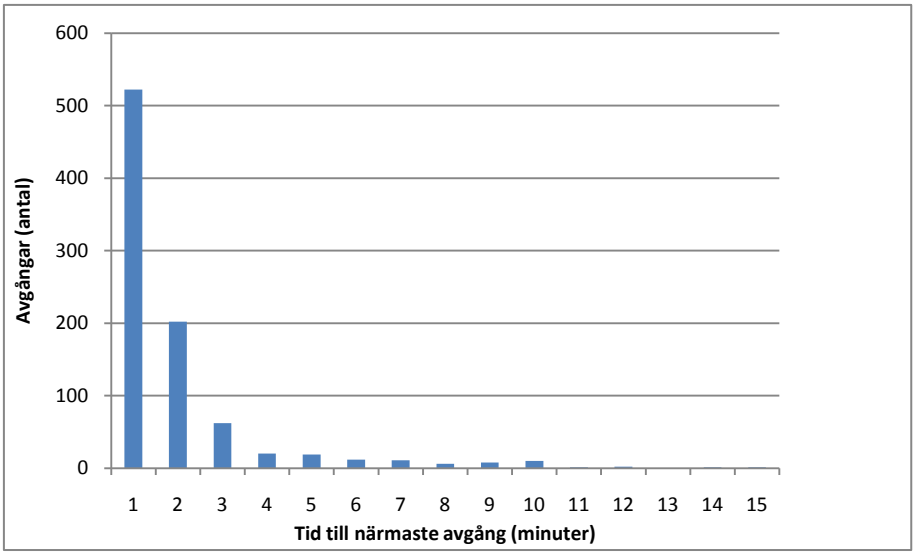
Figur 4-8: Antal rutter som avgår ifrån respektive startstation.

För att undersöka hur en sträcka kan variera beroende på vilken individ vi observerar undersöker vi antal avgångar för en individ, samt hur många av de avgångar ett tåg gör är på återkommande sträckor. Vi ser i Figur 4-9 att individen ger oss viss information om vilken sträcka vi kan befinna oss på.



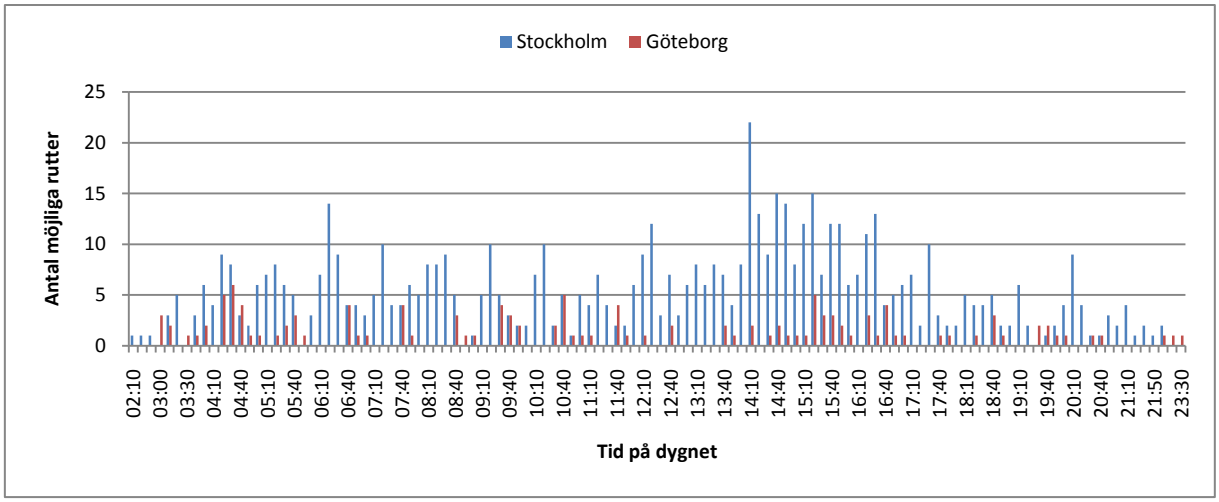
Figur 4-9: Antal avgångar och antal unika rutter för varje tåg.

För att se hur mycket vi kan lita på start tiden på en rutt har vi undersökt hur stor avgångsdifferensen är för rutter som går mellan Stockholm C och Göteborg C. Med avgångsdifferensen menar vi skillnaden i tid beroende på dag för avgångar som skall avgå på samma tid varje dag. X-axeln i diagrammet representerar hur stor skillnad i avgångstid det är gentemot den med närmast avgångstid, oberoende på dag, Y-axeln representerar antal avgångar med motsvarande skillnad i avgångstid. Utifrån Figur 4-10 ser vi att tågen statistiskt sett avgår inom 3 minuter från den utsatta avgångstiden.

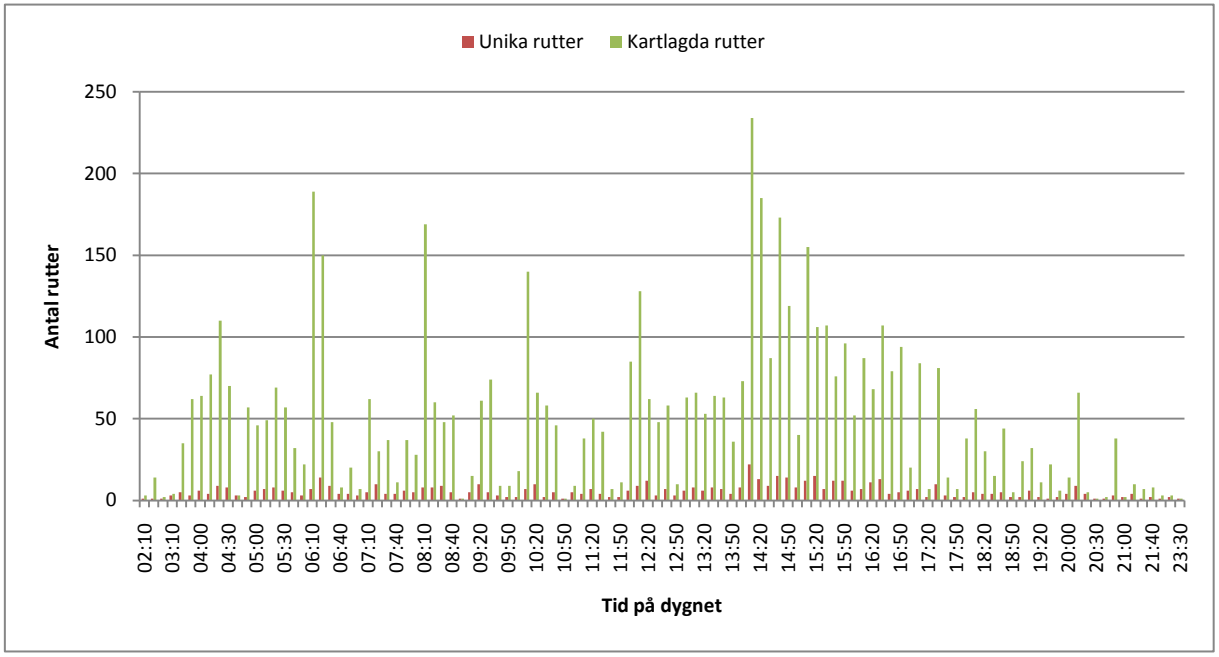


Figur 4-10: Tid i minuter till närmsta avgång för varje avgång.

För att se hur rutterna beror på avgångsstation och tid har vi här undersökt möjliga unika rutter beroende på start station och tid på dygnet med en tidsupplösning på 10 minuter. Vi visar i Figur 4-11 antal möjliga unika rutter om vi startar ifrån Stockholm C respektive Göteborg C vid ett specifikt klockslag. Vi undersöker även hur de unika rutterna förhåller sig till de totala avgångarna ifrån Stockholm C vid olika tider på dygnet i Figur 4-12.

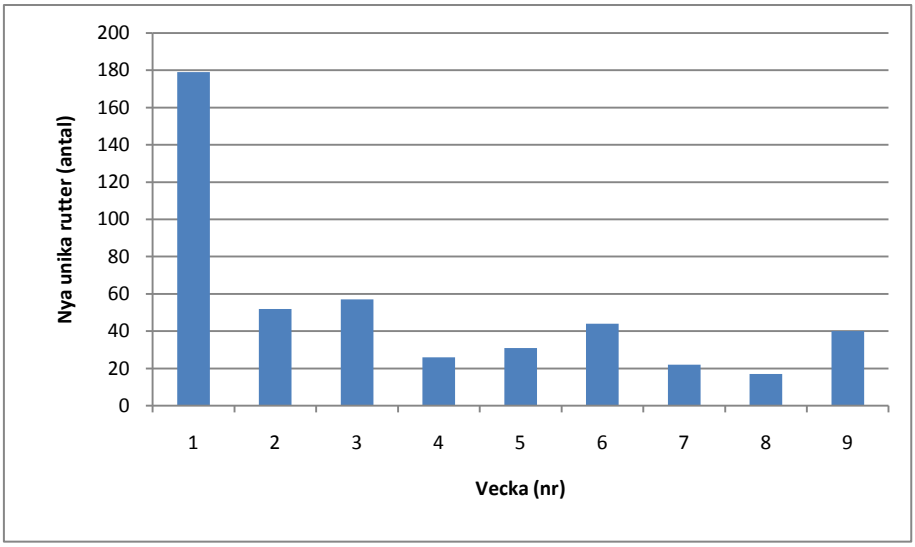


Figur 4-11: Antal möjliga rutter från Stockhom och Göteborg vid olika tider på dygnet.

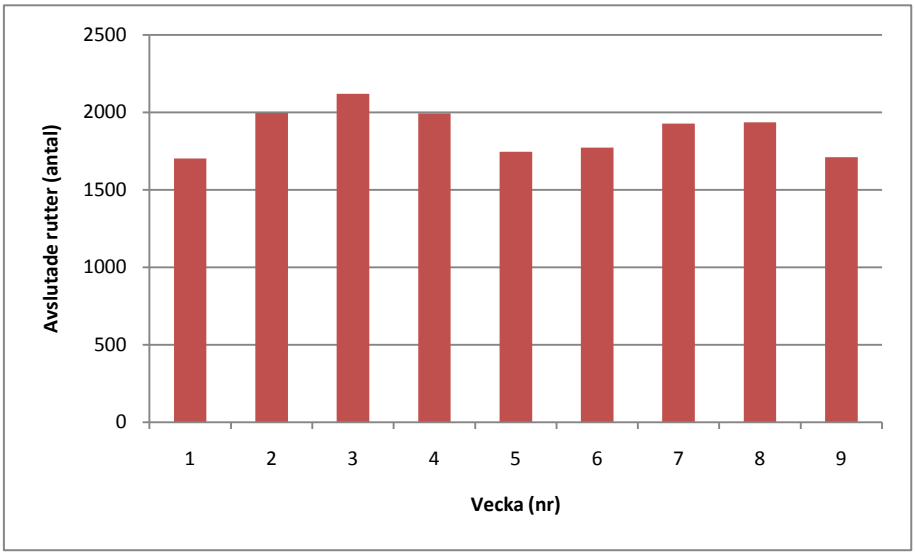


Figur 4-12 Unika och totalt antal avgångar ifrån Stockholm C beroende på tid på dygnet

Vi ser i Figur 4-13 antal nya unika rutter som hittas varje vecka. Första veckan hittar vi flest unika rutter eftersom vi tidigare inte har haft några rutter alls att utgå ifrån. Följande veckor hittar vi färre unika rutter eftersom fler unika rutter har hittats tidigare. I Figur 4-14 ser vi antal nya kartlagda rutter per vecka.



Figur 4-13: Antal nya unika rutter som hittas varje vecka.



Figur 4-14: Antal kartlagda rutter som registreras varje vecka.

4.9 Slutsatser - Kartläggning av rutter

Vi upptäcker en felaktighet med vår algoritm vilken påvisas i Figur 4-13. Vi anser det rimligt att anta att varje verklig rutt körs minst 1 gång per vecka. Detta borde innebära att vi efter ett par veckor har registrerat alla unika rutter som finns. Dock ser vi att vi hittar fler unika rutter även efter att vi undersökt 8 veckors samplingar, detta tyder på en felaktighet i vår modell för kartläggning av rutter.

Vi ser även att tydliga mönster i de rutter vi har registrerat och anser att även om det finns utrymme för förbättringar så har vi ett bra underlag för prediktion av rutter. En initial slutsats angående prediktion av rutter är att vi enbart med hjälp av avgångstid samt avgångsstation kan göra en ganska bra prediktion av slutstation.

4.10 Diskussion - Kartläggning av rutter

I det vidare arbetet inom detta område borde tid läggas ner på att analysera varför nya unika rutter inte slutar uppkomma. Detta kan till exempel göras genom att undersöka anledningen till att nya unika rutter skapas. Spåromläggning eller spårutbyggnad kan ha påverkat vår kartläggning av rutter.

Vi har i efterhand upptäckt att Banverket byggt ut en mötesplats på Svealandsbanan i området kring Stockholm och Mälardalen. Sträckan öppnades igen den 26:e oktober, därför misstänker vi att en del av de unika rutterna som hittas under den sista veckan i oktober är på grund av att tågen börjat trafikera den sträckan normalt igen (Trafikverket, 2010). Vi har dock inte gjort någon djupare undersökning av hur mycket detta har påverkat oss.

Vi har under arbetet med kartläggningen av rutter kommit fram till ett antal möjliga annorlunda tillvägagångssätt. Dessa tillvägagångssätt är dock inte lika flexibla som vår metod, men beroende på vad syftet med kartläggningen är kan de möjligen lämpa sig bättre.

Ett interface där användaren kan mata in vilka unika rutter man är intresserad skulle kunna tas fram. Tidtabeller för rutterna skulle sedan kunna byggas upp utifrån de inmatade rutterna och vi skulle undvika eventuella svagheter i algoritmen vad avser att hitta start och slutstation för rutten.

Ett helt annorlunda sätt att angripa problemet vore att bygga upp rutterna utifrån tidtabeller direkt ifrån tågoperatören och mata in dem direkt i en databas och basera rutterna på den informationen, istället för att själv generera dem.

En metod för kartläggning av rutter som baseras på en analys av en stor mängd historisk data som delas in i rutter skulle troligtvis ge upphov till en större pålitlighet hos rutterna. Detta skulle dock innebära att man behöver samla in all data som behövs för analysen innan den påbörjas, medan vår metod kan användas för att analysera en ström av data. Detta skulle kunna göra det lättare att observera mönster där vi nu begränsas av de antaganden som den metod vi valt tvingat oss till att göra.

Oavsett metoden för att bygga upp rutter behövs alltid en metod för att bygga upp rutter medan de pågår för att prediktion av rutter skall fungera. Metoden för att bygga upp rutter medan de pågår skulle dock hjälpas mycket av en lista över giltiga unika rutter som grund.

5 Analys av ruttprediktion

Syftet med analysen är att dra slutsatser om huruvida tågens pågående rutter kan predikteras med en utvald algoritm samt utvärdera hur väl algoritmen lyckas prediktera dem. Slutsatserna är värdefulla för Icomera då de kan hjälpa företaget att identifiera potentiella affärsmöjligheter inom området. Förhoppningen är att tågens rutter skall kunna predikteras på ett sådant sätt att en verklig tillämpning kan hittas. Exempel på sådana tillämpningar är:

- Prediktera kommande trafiksituationer, för att till exempel underlätta eco-driving.
- Information och reklam till passagerare beroende på vart tåget är på väg.
- Förseningsinformation i realtid.
- Information om tåg till supporten på Icomera

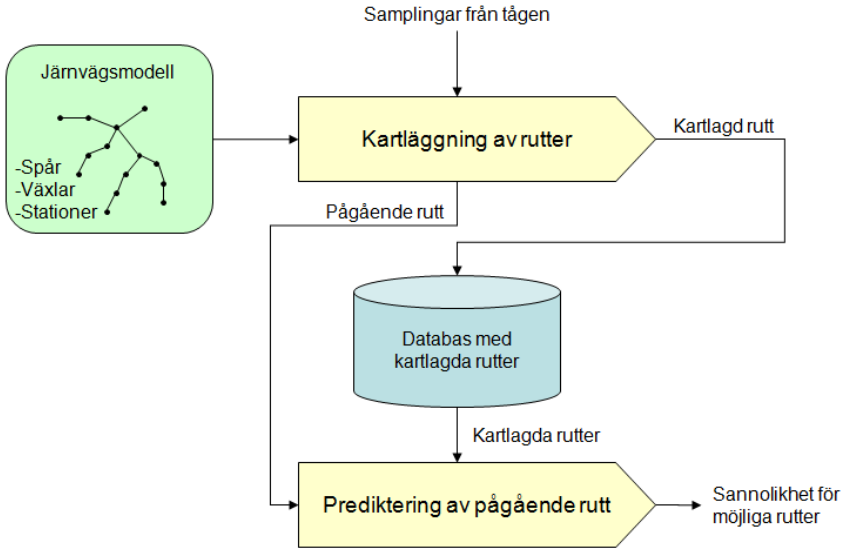
Tågens rutter kan predikteras på flera olika sätt. Man kan tänka sig att använda tågoperatörernas officiella tidtabeller och matcha tågens pågående rutter mot dessa. Man kan även tänka sig att prediktera rutterna utan hjälp av officiella tidtabeller. Icomera uttryckte önskemål om att analysera möjligheten för prediktering av tågens rutter enbart baserat på tågens samplingar. Motiveringen till detta är att Icomera ville undersöka om information från enbart företagets system är tillräcklig för att prediktera tågens rutter. Detta skulle i så fall medföra ett ökat värde på Icomeras system och tjänster. Vi valde därför att utföra analysen av ruttprediktering utan hjälp av officiella tidtabeller.

Vi begränsade uppgiften till att välja ut och implementera en algoritm för prediktion av rutter och därefter utvärdera hur väl algoritm fungerar. Vi valde att prediktera tågens rutter med hjälp av beslutsträd och resterande delar av här kapitlet beskiver problemet, lösningen, resultat och slutsatser.

5.1 Problembeskrivning

Problemet kan beskrivas som att utifrån ett tågs pågående rutt finna den redan kartlagda rutt som tåget med högst sannolikheten kommer åka. Vi har valt att begränsa problemet till att prediktera vilka stationer tåget kommer att passera längs sin färdväg samt att prediktera tågets slutstation. Vi har inte analyserat möjligheten att prediktera vid vilka stationer som tågen kan antas stanna samt vilken tid tågen förväntas ankomma vid stationerna. Ett önskemål är att predikteringen endast ska använda sig av samplingar från tågen. Önskemålet ökar problemets svårighet eftersom predikteringen på grund av det måste baseras på de rutter som kartlagts med kartläggningsalgoritmen, även om den algoritmen har en del brister. En annan faktor som påverkar svårigheten är att flera tåg kan avgå från samma station vid samma tidpunkt, vilket visas i Figur 4-11. Det innebär att vi inte kan avgöra vilka rutter de olika tågen kan tänkas åka förrän tågen har trafikerat en viss sträcka av sina rutter.

Som beskrivs i *Kapitel 4* används järnvägsmodellen och samplingar från tågen för att kartlägga tågens pågående rutter. Innan kartläggningen av en rutt avslutas, och den kartlagda rutten sparas i databasen, kan den pågående rutten jämföras mot de redan kartlagda rutterna och en mängd av möjliga rutter kan identifieras. Problemet är att identifiera den rutt i mängden som med störst sannolikhet beskriver tågets resterande rutt. En översiktlig beskrivning av hur vi valt att utföra predikteringen visas i Figur 5-1.



Figur 5-1: Översiktlig beskrivning för prediktion av pågående rutter.

En rutt beskrivs av ett antal sorterade ruttelement där varje ruttelement representerar en station som tåget har passerat eller stannat vid. Rutten kan beskrivas som $r = \{s_1, s_2, \dots, s_{n-1}, s_n\}$, där s representerar ruttelementen hos rutten *r*. Stationen för s_1 representerar ruttens startstation och om rutten är kartlagd representerar stationen för s_n ruttens slutstation. För en pågående rutt, som är under kartläggning, representerar stationen för s_n den senaste stationen som tåget har passerat eller stannat vid. En pågående rutt har alltid minst ett ruttelement, och därmed en startstation, vilket gör det möjligt att identifiera en mängd kartlagda rutter som den pågående rutten är en delmängd av.

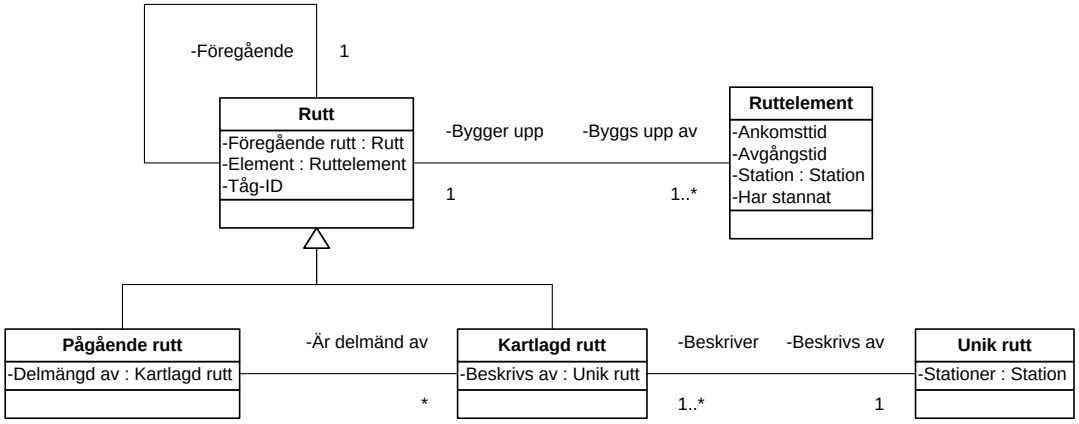
Definitioner
R är mängden av alla kartlagda rutter.
$\boldsymbol{K}(r_p)$ är mängden av kartlagda rutter som den pågående rutten r_p är en delmängd av.
$\boldsymbol{U}(r_p)$ är mängden av de unika rutterna som kan beskriva resterande del för den pågående ruttenr_p.

En rutt, r_1, är delmängd av rutt r_2 , om stationen för ruttelementet $s_{i,1}$ är samma som stationen för $s_{i,2}$ för samtliga ruttelement $s_{i,1}$ hos r_1.

$$r_1 \subseteq r_2 \text{om } s_{i,1} = s_{i,2} \text{ för alla ruttelement hos } r_1$$

Två rutter, $r_1$och r_2 är ekvivalenta om $r_1 \subseteq r_2$ och $r_2 \subseteq r_1$, vilket innebär att de beskriver samma rutt men vid olika tidpunkter, och beskrivs därför av samma unika rutt.

$$r_1 \Leftrightarrow r_2 \text{ om } r_1 \subseteq r_2 \wedge r_2 \subseteq r_1$$



Figur 5-2: Klassdiagram för relationen mellan pågående, kartlagd och unik rutt.

Av Figur 5-2 framgår bland annat att en pågående rutt kan vara delmängd av ingen eller flera kartlagda rutter. Det framgår även att kartlagda rutter som är ekvivalenta beskrivs av samma unika rutt. För att finna de unika rutter som kan beskriva resterande del för den pågående rutten r_p måste först mängden $\boldsymbol{K}(r_p)$ identifieras från ***K*** och sedan kan mängden unika rutter $\boldsymbol{U}(r_p)$ identifieras.

Exempel
Om vi antar att mängden av alla kartlagda rutter $\boldsymbol{K}$ = $\{r_1, r_2, r_3, r_4, r_5, r_6\}$ där de kartlagda rutterna definieras som:

r_1 =*{Flen, Kumla, Sala, Timrå, Nykvarn}*
r_2 =*{Malmö, Heby, Halmstad, Gävle}*
r_3 =*{Flen, Kumla}*
r_4 =*{Göteborg, Alingsås, Hallsberg}*
r_5 =*{Flen, Kumla, Sala, Timrå, Nykvarn}*
r_6 =*{Flen, Kumla, Sala, Timrå, Nykvarn, Heby}*

Om den pågående rutten r_p = *{Flen, Kumla, Sala}* finner vi att r_p är en delmängd av de kartlagda rutterna $\{r_1, r_5, r_6\} = \boldsymbol{K}(\boldsymbol{r_p})$, vilket är mängden av kartlagda rutter som den pågående rutten r_p är en delmängd av. Vi ser även att $r_1 \Leftrightarrow r_5$ vilket medför att mängden av unika rutter, som kan beskriva resterande del av r_p, innehåller två unika rutter och vi får $\boldsymbol{U}(r_p)$=$\{u_1, u_2\}$, där

u_1 =*{Flen, Kumla, Sala, Timrå, Nykvarn}*
u_2 =*{Flen, Kumla, Sala, Timrå, Nykvarn, Heby}*

Problemet är att identifiera den unika rutt hos $\boldsymbol{U}(r_p)$ som med störst sannolikhet beskriver resterande del av den pågående rutten r_p. Vi valde att lösa problemet genom att beräkna sannolikheten, $p(r_p, u)$, för varje unik rutt hos $\boldsymbol{U}(r_p)$ och välja den unika rutt som har högst sannolikhet.

En faktor som påverkar problems svårighet är antalet unika rutter hos mängden $\boldsymbol{U}(r_p)$, ju färre rutter desto enklare blir problemet. Om $\boldsymbol{U}(r_p)$ endast innehåller en unik rutt är det sannolikt att denna beskriver resterande del av r_p även om $p(r_p, u)$ är litet. Om istället antalet unika rutter hos $\boldsymbol{U}(r_p)$ är stort kan det finnas flera unika rutter där $p(r_p, u)$ är stort och vi tvingas välja den unika rutt som har högst sannolikhet. Om $\boldsymbol{U}(r_p)$ inte innehåller några unika rutter är r_p inte delmängd av någon kartlagd rutt och ingen prediktion är möjlig då rutten är okänd.

5.2 Val av algoritm

Det finns en mängd olika algoritmer som kan användas för att beräkna sannolikheten $p(r_p, u)$ för varje unik rutt hos $\boldsymbol{U}(r_p)$. Den enklaste algoritmen vi såg framför oss var att beräkna sannolikheten utifrån hur många gånger varje unik rutt trafikerats. Den unika rutt som trafikerats flest gånger anses då alltid mest sannolik. Problemet med algoritmen är att den enbart tar hänsyn till en enda faktor som påverkar sannolikheten, och förbiser andra faktorer som till exempel avgångstid.

Den enkla algoritmen skulle kunna utökas och ta hänsyn till andra faktorer för att beräkna den betingade sannolikheten med hjälp av Bayes formel, d.v.s. hur vanlig en viss rutt är givet några faktorer, till exempel avgångstid och veckodag. Detta kräver dock kunskap om de faktorer som påverkar sannolikheten. Vi ansåg det svårt att veta vilka faktorer som är av intresse och hur respektive faktor påverkar sannolikheten.

Vi valde att använda lärande beslutsträd eftersom algoritmen analyserar faktorernas inverkan på sannolikheten och automatiskt väljer de faktorer som har mest inverkan. Vidare är lärande beslutsträd enkla att implementera men ändå en av de mest framgångsrika lärande algoritmerna (Russell et al., 2003).

Då $p(r_p, u)$ ska beräknas för varje unik rutt hos $\boldsymbol{U}(r_p)$ valde vi att skapa ett beslutsträd för varje unik rutt hos $\boldsymbol{U}(r_p)$. Ett fall skapas från den pågående rutten r_p och testas (klassificeras) mot varje beslutsträd för att få svar på frågan "Hur stor är sannolikheten att denna unika rutt beskriver resterande del av r_p?". Resultatet från beslutsträden rangordnas beroende på sannolikhet och den unika rutt som har högst sannolikhet väljs.

Huvudstegen för att prediktera resterande delen av ett tågs pågående rutt r_p är:

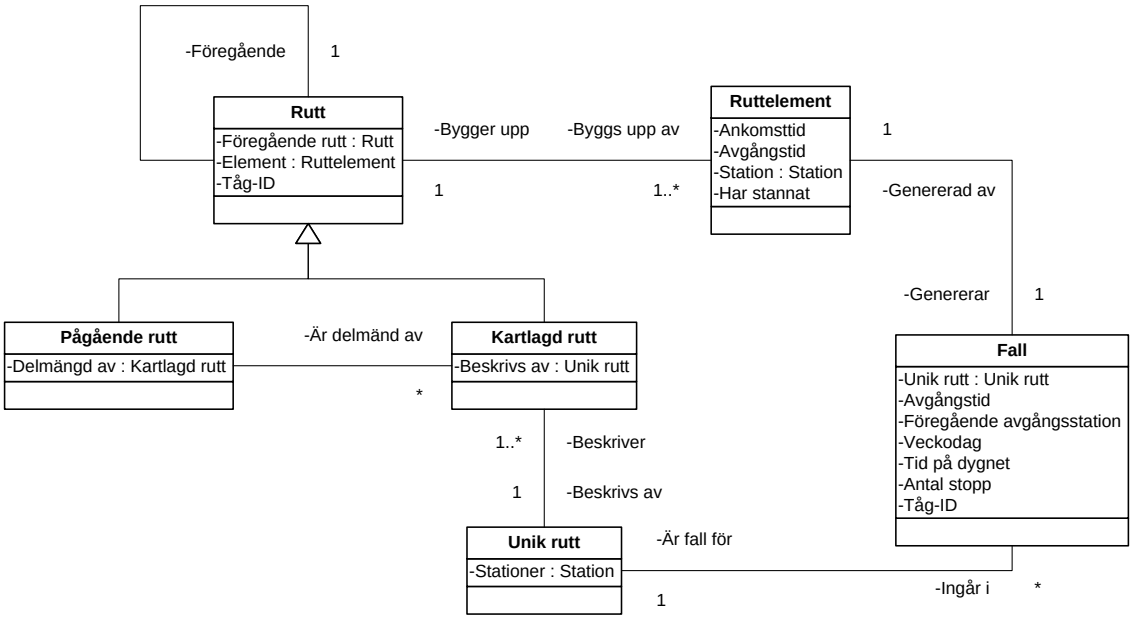
1. Identifiera $\mathbf{K}(r_p)$ och $\mathbf{U}(r_p)$.
2. För varje unik rutt hos $\mathbf{U}(r_p)$
3. Skapa träningsmängd med aktuella fall baserat på $\mathbf{K}(r_p)$.
4. Gör om kontinuerliga värden hos träningsmängden till diskreta värden.
5. Skapa ett beslutsträd baserat på träningsmängden.
6. Skapa ett fall baserat på den pågående rutten r_p.
7. Klassificera fallet mot varje beslutsträd för att beräkna sannolikheten $p(r_p, u)$.
8. Rangordna resultaten från beslutsträden och välj den unika rutt som har högst sannolikhet.

5.2.1 Skapande av träningsmängd

Varje beslutsträd baseras på en träningsmängd som används för att konstruera ett så effektivt beslutsträd som möjligt. Med effektivt beslutsträd menas att trädet korrekt kan klassificera och svara för fall som ännu inte har observerats. Träningsmängden består av ett antal fall, där varje fall beskriver en situation med hjälp av attribut. Attributen som användes för att beskriva situationen för ett fall är:

- *Unik rutt*: Den unika rutt som tåget körde.
- *Avgångstid*: Tid på dygnet då rutten påbörjades, avrundat till minut.
- *Föregående avgångsstation*: Avgångsstationen för tågets föregående rutt.
- *Veckodag*: Veckodag då rutten påbörjades.
- *Tid på dygnet*: Tid på dygnet då tåget passerade en viss station, avrundat till minut.
- *Antal stopp*: Antal stationer som tåget stannat vid när den passerar en viss station.
- *Tåg-ID*: Identitetsnumret för tåget som körde rutten.

Attributvärdena för varje fall baseras på ett ruttelement och klassrelationerna illustreras i klassdiagrammet i Figur 5-3.



Figur 5-3: Klassdiagram för relationen mellan ruttelement och fall.

Fallen hos träningsmängden baseras på ruttelementen hos de kartlagda rutterna $\boldsymbol{K}(r_p)$ samt det sista ruttelementet för den pågående rutten r_p. Ruttelementen hos $\boldsymbol{K}(r_p)$ som träningsmängden baseras på är de ruttelement vars station är samma som för det sista ruttelementet för r_p. När r_p utökas med ytterligare ruttelement kommer därför träningsmängden förändras.

Exempel
Vi antar att r_p{Flen, Kumla} och $\mathbf{K}(r_p) = \{r_1, r_2, r_3; r_4\}$ där de kartlagda rutterna definieras som:

r_1 = *{Flen, Kumla, Sala, Timrå, Nykvarn}*
r_2 = *{Flen, Kumla}*
r_3 = *{Flen, Kumla, Sala, Timrå, Nykvarn}*
r_4 = *{Flen, Kumla, Sala, Timrå, Nykvarn, Heby}*

Vi har tre unika rutter och får $\boldsymbol{U}(r_p) = \{u_1, u_2, u_3\}$, där

u_1 = *{Flen, Kumla, Sala, Timrå, Nykvarn}*
u_2 = *{Flen, Kumla}*
u_3 = *{Flen, Kumla, Sala, Timrå, Nykvarn, Heby}*

Fallen för träningsmängden baseras på ruttelementen hos $\boldsymbol{K}(r_p)$ som innehåller stationen *Kumla* och visas i Tabell 5-1.

Fall	Attribut						
	Unik rutt	*Avgångstid*	*Föregående avgångsstation*	*Veckodag*	*Tid på dygnet*	*Antal stopp*	*Tåg-ID*
1	u_1	12:00:03	Göteborg	Onsdag	12:10:44	1	1234
2	u_2	11:59:02	Malmö	Måndag	12:11:51	1	5312
3	u_1	11:55:22	Malmö	Onsdag	12:02:45	1	1234
4	u_3	17:33:10	Malmö	Fredag	17:44:02	2	1234

Tabell 5-1: Träningsmängdens med fall för ruttelementen hos $\boldsymbol{K}(r_p)$ som innehåller stationen *Kumla*.

Vi har valt att skapa ett beslutsträd för varje unik rutt hos $\boldsymbol{U}(r_p)$, vilket innebär att tre beslutsträd skapas för det här exemplet. Varje beslutsträd baseras på en egen träningsmängd där de unika rutterna för fallen används för att binärt beskriva utfallet, det vill säga, om fallet resulterade i den unika rutten eller inte. I Tabell 5-2 visas träningsmängderna för de tre beslutsträden.

Fall	Träningsmängd för beslutsträd för u_1						Utfall u_1?
	Avgångstid	*Föregående avgångsstation*	*Veckodag*	*Tid på dygnet*	*Antal stopp*	*Tåg-ID*	
1	12:00:03	Göteborg	Onsdag	12:10:44	1	1234	**Ja**
2	11:59:02	Malmö	Måndag	12:11:51	1	5312	Nej
3	11:55:22	Malmö	Onsdag	12:02:45	1	1234	**Ja**
4	17:33:10	Malmö	Fredag	17:44:02	2	1234	Nej

Fall	Träningsmängd för beslutsträd för u_2						Utfall u_2?
	Avgångstid	*Föregående avgångsstation*	*Veckodag*	*Tid på dygnet*	*Antal stopp*	*Tåg-ID*	
1	12:00:03	Göteborg	Onsdag	12:10:44	1	1234	Nej
2	11:59:02	Malmö	Måndag	12:11:51	1	5312	**Ja**
3	11:55:22	Malmö	Onsdag	12:02:45	1	1234	Nej
4	17:33:10	Malmö	Fredag	17:44:02	2	1234	Nej

Fall	Träningsmängd för beslutsträd för u_3						Utfall u_3?
	Avgångstid	*Föregående avgångsstation*	*Veckodag*	*Tid på dygnet*	*Antal stopp*	*Tåg-ID*	
1	12:00:03	Göteborg	Onsdag	12:10:44	1	1234	Nej
2	11:59:02	Malmö	Måndag	12:11:51	1	5312	Nej
3	11:55:22	Malmö	Onsdag	12:02:45	1	1234	Nej
4	17:33:10	Malmö	Fredag	17:44:02	2	1234	**Ja**

Tabell 5-2: Träningsmängder för beslutsträden u1, u2, u3.

I Tabell 5-2 ser vi att attributet *Unik rutt* har tagits bort och istället används för att beskriva om fallens utfall är positiva eller negativa för den unika rutten.

Nya träningsmängder skapas då den pågående rutten utökas med nya ruttelement. Om r_p skulle utökas med ett nytt ruttelement för stationen *Sala*, r_p = *{Flen, Kumla, Sala}*, kommer ***K**(r_p)* reduceras till $\{r_1, r_3, r_4\}$ och ***U**(r_p)* reduceras till $\{u_1, u_3\}$ och två nya beslutsträd skapas. Träningsmängderna för de två beslutsträden kommer nu baseras på de ruttelement hos ***K**(r_p)* som innehåller stationen *Sala*.

5.2.2 Hur beslutsträden används

Ett beslutsträd kan beskrivas som en funktion som klassificerar ett fall *f*. Funktionens resultat är en tvåvärd resultatvektor $v = (j, n)$ där *j* är antalet positiva utfall och *n* antalet negativa utfall för klassificeringen av *f*.

$$f \rightarrow \text{beslutsträd} \rightarrow (\text{j}, \text{n})$$

Resultatvektorn kan användas för att ge den betingade sannolikheten för en resultatvektor beräknas som andelen positiva utfall genom det totala antalet utfall.

$$p(v) = \frac{j}{j+n}$$

Ett fall, f_p, kan genereras från det sista ruttelementet hos den pågående rutten r_p. Eftersom en pågående rutt inte är kartlagd saknar den en koppling till unik rutt och attributet *Unik rutt* får ett ospecificerat värde.

Fall för den pågående rutten r_p						
Unik rutt	*Avgångstid*	*Föregående avgångsstation*	*Veckodag*	*Tid på dygnet*	*Antal stopp*	*Tåg-ID*
?	12:03:27	Malmö	Måndag	13:42:52	1	6321

Tabell 5-3: Exempel på ett fall genererat av det sista ruttelementet hos den pågående rutten.

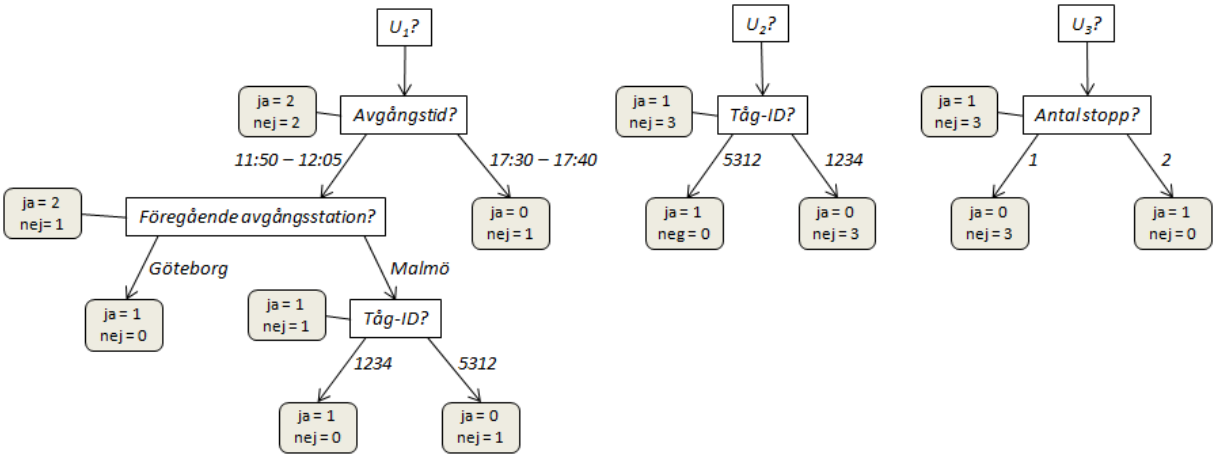
Tabell 5-3 visar ett exempel på f_p för det sista ruttelementet hos r_p. Genom att klassificera falletf_p mot beslutsträden för de unika rutterna kan sannolikheten $p(r_p, u)$ beräknas.
Om f_p i Tabell 5-3 klassificeras mot beslutsträden i Figur 5-4 får vi följande resultatvektorer och sannolikheter:

$$f_p \rightarrow u_1? \rightarrow (1,1) \Rightarrow p(r_p, u_1) = \frac{1}{1+1} = 0.5$$
$$f_p \rightarrow u_2? \rightarrow (1,3) \Rightarrow p(r_p, u_2) = \frac{1}{1+3} = 0.25$$
$$f_p \rightarrow u_3? \rightarrow (0,3) \Rightarrow p(r_p, u_3) = \frac{0}{0+3} = 0$$

Av resultaten från klassificeringen av fallet f_p framgår att u_1 har störst sannolikhet och att u_1 därmed är den unika rutt som med störst sannolikhet beskriver resterade del av tågets pågående rutt.

5.3 Skapande av beslutsträd

Figur 5-4 visar ett exempel på hur beslutsträden för träningsmängderna i Tabell 5-2 kan tänkas se ut.



Figur 5-4: Exempel på beslutsträd baserat på träningsmängderna i Tabell 5-2.

Problemet är att konstruera beslutsträden på sådant sätt att de, så effektivt som möjligt, kan klassificera fall som ännu inte har observerats. En lösning på problemet är att följa principen enligt Ockham's razor och konstruera det "minsta" träd som är konsistent med träningsmängden (Russell et al., 2003). Det är svårt att definiera och konstruera det "minsta" trädet, men det finns förutsättningar för att minimera storleken hos träden och på så sätt närma sig Ockham's razor. Grundidén för att minimera ett träd är att det mest avgörande attributet finns vid trädets rot och därmed testas först, det näst mest avgörande på nästa nivå och så vidare. Med "mest avgörande" menas det attribut som gör störst skillnad i klassificeringen av ett fall. I träningsmängden för u_3 i Tabell 5-2 ser vi t.ex. att attributet *Antal stopp* är ett avgörande attribut som gör stor skillnad i klassificeringen då det delar mängden i enbart positiva och negativa utfall. Samtidigt kan vi se att attributet *Tåg-ID* är ett minde avgörande attribut för u_3 då det delar mängden i delmängder som består av både positiva och negativa utfall.

Vi valde att implementera algoritmen "Decision tree learning algoritm" baserat på (Russell et al., 2003) för att konstruera beslutsträden. Algoritmen avser att minimera trädens storlek och en beskrivning av dess pseudokod är:

funktion TRÄNA-BESLUTSTRÄD(träningsmängd, möjligaAttribut, defaultUtfall)
parametrar:träningsmängd: Mängd av fall
möjligaAttribut: Mängd av användbara attribut
defaultUtfall: Tvåvärd vektor med antal utfall {Antal "Ja", Antal "Nej"}

Om träningsmängd är tom **returnera** defaultUtfall
annars om träningsmängd endast har ett utfall **returnera** utfallet
annars om möjligaAttribut är tom **returnera** ANTAL-UTFALL(träningsmängd)
annars
mestAvgörande = VÄLJ-MEST-AVGÖRANDE-ATTRIBUT(möjligaAttribut, träningsmängd)
träd = Ett nytt beslutsträd med mestAvgörande som rotattribut
def = ANTAL-UTFALL(träningsmängd)
reduceradeAttribut = möjligaAttribut - mestAvgörande
för varje värde v_i hos bästaAttribut **gör**:
$träningsmängd_i$ = {de fall hos träningsmängd där mestAvgörande = v_i}
underTräd = TRÄNA-BESLUTSTRÄD($träningsmängd_i$, reduceradeAttribut, def)
träd[v_i] = underTräd
returnera träd

Funktionen VÄLJ-MEST-AVGÖRANDE-ATTRIBUT har parametrarna *möjligaAttribut*, *träningsmängd* och returnerar det attribut hos *möjligaAttribut* som bidrar med mest information om utfallen hos *träningsmängd*, med andra ord, det attribut som gör störst skillnad i klassificeringen av ett fall.

Funktionen ANTAL-UTFALL har parametern *träningsmängd* och returnerar en tvåvärd vektor som innehåller antalet positiva och negativa utfall, det vill säga antalet "Ja" och antalet "Nej" för utfallen hos *träningsmängd*.

5.3.1 Överträning

Algoritmen TRÄNA-BESLUTSTRÄD skapar alltid en gren för varje värde hos ett attribut som ingår i mängden av möjliga attribut. Om t.ex. attributet *Tåg-ID* är ett möjligt attribut, men egentligen inte har någon inverkan på utfallen, kommer algoritmen ändå skapa en gren för varje tåg-ID. Fenomenet kallas för överträning och uppstår om träningsmängden innehåller irrelevanta attribut. Resultatet av överträning är att träden blir onödigt stora och mindre effektiva (Russell et al., 2003). Genom att ta bort irrelevanta attribut kan överträning undvikas, träden blir mindre och mer effektiva. Ett attribut anses vara irrelevant om det inte är statistiskt signifikant. Vi valde att implementera χ^2-beskärning enligt (Russell et al., 2003), för att avgöra om ett attribut är statistiskt signifikant. Tröskelvärdet för avvikelsen i vår χ^2-beskärning är satt till 5 % som är ett beprövat värde (Russell et al., 2003).

Nedan visas pseudokod för algoritmen TRÄNA-BESLUTSTRÄD-CHI-SQUARE som utökats med χ^2-beskärning.

funktion TRÄNA-BESLUTSTRÄD-CHI-SQUARE (*träningsmängd*, *möjligaAttribut*, *defaultUtfall*)
parametrar:*träningsmängd*: Mängd av fall
möjligaAttribut: Mängd av användbara attribut
defaultUtfall: Tvåvärd vektor med antal utfall {Antal "Ja", Antal "Nej"}

Om *träningsmängd* är tom: **returnera** *defaultUtfall*
annars om *träningsmängd* endast har ett utfall: **returnera** utfallet
irrelevantaAttribut = CHI-SQUARE-BESKÄRNING(*träningsmängd*, *möjligaAttribut*)
möjligaAttribut = *möjligaAttribut* - *irrelevantaAttribut*
om *möjligaAttribut* är tom: **returnera** ANTAL-UTFALL(*träningsmängd*)
annars
mestAvgörande = VÄLJ-MEST-AVGÖRANDE-ATTRIBUT(*möjligaAttribut*, *träningsmängd*)
träd = Ett nytt beslutsträd med *mestAvgörande* som rotattribut
def = ANTAL-UTFALL(*träningsmängd*)
reduceradeAttribut = *möjligaAttribut* - *mestAvgörande*
för varje värde v_i hos *bästaAttribut* **gör**:
$träningsmängd_i$ = {de fall hos *träningsmängd* där *mestAvgörande* = v_i}
underTräd = TRÄNA-BESLUTSTRÄD-CHI-SQUARE ($träningsmängd_i$, *reduceradeAttribut*, *def*)
träd[v_i] = *underTräd*
returnera träd

Funktionen CHI-SQUARE-BESKÄRNING har *träningsmängd* och *möjliga-Attribut* som parametrar och returnerar de attribut hos *möjligaAttribut* som anses irrelevanta baserat på fallen hos *träningsmängd*. De irrelevanta attributen tas bort från mängden av möjliga attribut vilket medför att överträning undviks och trädens storlek minskar.

5.3.2 Hantering av kontinuerliga värden

Träningsmängden innehåller attribut som är kontinuerliga, dessa är *Avgångstid*, *Tid på dygnet* och *Antal stopp* och kan anta en oändlig mängd möjliga värden. De övriga attributen är diskreta och kan endast anta en begränsad mängd värden. Vi valde att hantera *Antal stopp* som ett diskret värde, vilket innebär att trädet får en gren för varje värde som träningsmängden antar för attributet.

Attributen *Avgångstid* och *Tid på dygnet* beskriver en tid på dygnet, avrundat till minuter, de är diskreta men kan ändå anta en stor mängd olika värden. Mängden av möjliga värden är begränsat till antalet minuter på ett dygn vilket är 1440 stycken. I värsta fall skulle ett träd därför kunna innehålla 1440 grenar, en för varje minut på dygnet, för dessa två attribut. Vi har sett att avgångstiderna för samma rutt har en tendens att variera med cirka ±5 minuter, se avsnitt 4.8. Det är därför rimligt att betrakta snarlika avgångstider som samma avgångstid, även om avgången i själva verket påbörjades vid olika tidpunkter på dygnet.

Vi valde att diskretisera värdena för *Avgångstid* och *Tid på dygnet* genom att istället beskriva dem som tidsintervall på dygnet i antal minuter. Metoden är beprövad och har tidigare visat sig ge bra resultat (Simmons et al., 2006). Ett exempel på ett sådant tidsintervall kan vara "14:18 - 14:35" vilket kan användas som värde för de fall hos träningsmängden där *Avgångstid* eller *Tid på dygnet* ligger inom intervallet. Problemet är att finna de lämpliga tidsintervallen och vi implementerade en egen algoritm, GÖR-DISKRET-TIDSINTERVALL, för att lösa problemet.

funktion GÖR-DISKRET-TIDSINTERVALL (*träningsmängd*, *attribut*)
parametrar:*träningsmängd*: Mängd av fall
attribut: Attributet som innehåller tid på dygnet som skall diskretiseras

Om *träningsmängd* endast har ett värde för *attribut*: **returnera** *träningsmängd*
annars
för varje fall f_i hos träningsmängd **gör**:
om utfallet för f_i är positivt
kontinuerligtVärde = f_i [*attribut*]
diskretaVärden = *diskretaVärden* + SKAPA-INTERVALL(*kontinuerligtVärde*)
nästa fall
SAMMANFOGA-ÖVERLAPPANDE-INTERVALL(*diskretaVärden*)
SKAPA-RESTERANDE-INTERVALL(*diskretaVärden*)
för varje fall f_i hos träningsmängd **gör**:
kontinuerligtVärde = f_i [*attribut*]
diskretVärde = HÄMTA-INTERVALL(*kontinuerligtVärde*, *diskretaVärden*)
lägg till *diskretVärde* till f_i
nästa fall
returnera *träningsmängd*

Funktionen SKAPA-INTERVALL har *kontinuerligtVärde* som parameter. Parametervärdet är en tid på dygnet och funktionen returnerar ett tidsintervall som startar 5 minuter innan parametervärdet och slutar 5 minuter efter där start- och sluttid är avrundade till hela minuter. Ett exempel är SKAPA-INTERVALL("14:45") = "14:40-14:50".

Funktionen SAMMANFOGA-ÖVERLAPPANDE-INTERVALL har *diskretaVärden* som parameter och parametervärdet innehåller en mängd av tidsintervall. Funktionen sammanfogar tidsintervall som överlappar varandra. Två intervall i_1 och i_2 överlappar varandra om start- eller sluttid för i_1 faller inom intervallet i_2. Sammanfogningen sker genom att ersätta de två tidsintervallen med ett nytt intervall i_{nytt} där starttiden för i_{nytt} är $\min(i_1.start, i_2.start)$ och sluttiden för i_{nytt} är $\max(i_1.slut, i_2.slut)$.

Ett exempel är SAMMANFOGA-ÖVERLAPPANDE-INTERVALL({"14:40-14:50", "12:00-12:10", "14:38-14:48"}) = {"12:00-12:10", "14:38-14:50"}.

Funktionen SKAPA-RESTERANDE-INTERVALL utökar parametern *diskretaVärden* med de tidsintervall som saknar för att täcka dygnets alla minuter. Ett exempel är SKAPA-RESTERANDE-INTERVALL({"12:00-12:10", "14:38-14:50"}) = {"00:00-11:59", "12:00-12:10", "12:11-14:37", "14:38-14:50", "14:51-23:59"}

Funktionen HÄMTA-INTERVALL har två parametrar, *kontinuerligtVärde* och *diskretaVärden*. Parametervärdet för *kontinuerligtVärde* är en tid på dygnet och *diskretaVärden* är en mängd tidsintervall. Funktionen returnerar det tidsintervall från mängden *diskretaVärden* som tiden för *kontinuerligtVärde* överrensstämmer med.

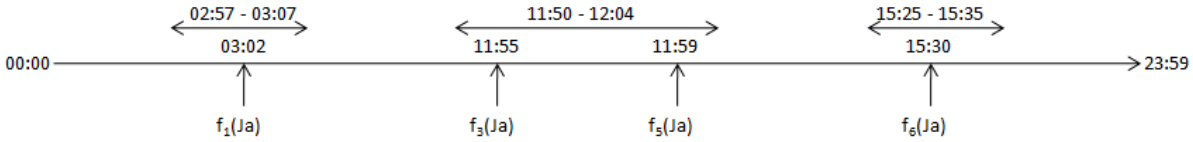
Algoritmen garanterar att funktionsanropet till HÄMTA-INTERVALL alltid returnerar ett tidsintervall eftersom SKAPA-RESTERANDE-INTERVALL har anropats innan funktionen HÄMTA-INTERVALL används.

Vi antar att GÖR-*DISKRET*-TIDSINTERVALL anropas med träningsmängden i Tabell 5-4 och med attributet *Avgångstid* som attribut som för diskretisering.

Fall	Träningsmängd						Utfall u_1?
	Avgångstid	*Föregående avgångsstation*	*Veckodag*	*Tid på dygnet*	*Antal stopp*	*Tåg-ID*	
f_1	03:02:03	Stockholm	Fredag	03:14:22	1	1234	**Ja**
f_2	08:10:32	Malmö	Onsdag	08:20:41	2	3252	Nej
f_3	11:55:22	Göteborg	Onsdag	12:10:44	1	1234	**Ja**
f_4	11:57:02	Malmö	Måndag	12:11:51	1	5312	Nej
f_5	11:59:21	Malmö	Onsdag	12:09:45	1	1234	**Ja**
f_6	15:30:03	Malmö	Onsdag	15:41:45	1	1234	**Ja**
f_7	17:33:10	Malmö	Fredag	17:44:02	2	1234	Nej

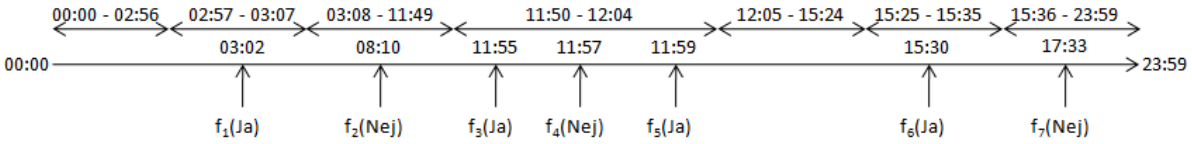
Tabell 5-4: Exempel på träningsmängd där attributet *Avgångstid* skall göras diskret.

Då träningsmängden *innehåller* fyra positiva utfall skapas fyra tidsintervall med funktionen SKAPA-INTERVALL. Två av tidsintervallen överlappar varandra och sammanfogas, vilket ger totalt tre tidsintervall då funktionen SAMMANFOGA-ÖVERLAPPANDE-INTERVALL har anropats, resultatet visas i Figur 5-5.



Figur 5-5: Diskreta tidsintervall skapade för det kontinuerliga värdet för *Avgångstid* efter anrop av SAMMANFOGA-ÖVERLAPPANDE-INTERVALL.

När SKAPA-RESTERANDE-INTERVALL anropas skapas totalt 4 nya intervall för att täcka resterande minuter på dygnet och resultatet illustreras i Figur 5-6.



Figur 5-6: Intervall skapade efter anrop till SKAPA-RESTERANDE-INTERVALL.

I Figur 5-6 ser vi att avgångstiden för varje fall hos träningsmängden överrensstämmer med ett intervall. Genom att lägga till ett nytt attribut till träningsmängden, *Avgångstid (diskret)*, och använda det överensstämmande tidsintervallet som ett diskret värde för varje fall kan träningsmängden uppdateras med diskreta värden för *Avgångstid*.

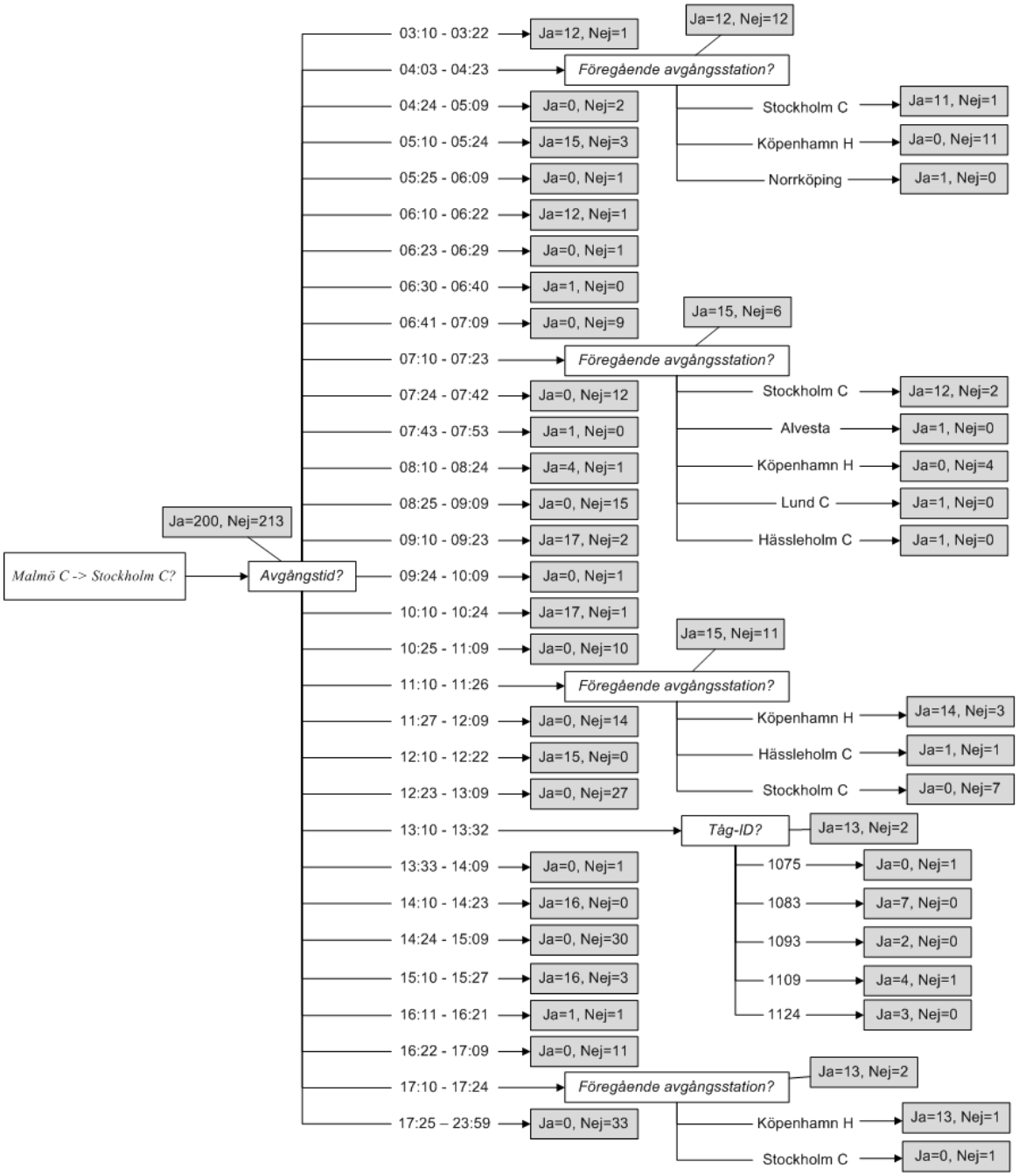
Fall	Träningsmängd							Utfall?
	Avgångstid	*Avgångstid (diskret)*	*Föregående avgångsstation*	*Veckodag*	*Tid på dygnet*	*Antal stopp*	*Tåg-ID*	
f_1	03:02:03	02:57 - 03:07	Stockholm	Fredag	03:14:22	1	1234	**Ja**
f_2	08:10:32	03:08 - 11:49	Malmö	Onsdag	08:20:41	2	3252	Nej
f_3	11:55:22	11:50 - 12:04	Göteborg	Onsdag	12:10:44	1	1234	**Ja**
f_4	11:57:02	11:50 - 12:04	Malmö	Måndag	12:11:51	1	5312	Nej
f_5	11:59:21	11:50 - 12:04	Malmö	Onsdag	12:09:45	1	1234	**Ja**
f_6	15:30:03	15:25 - 15:35	Malmö	Onsdag	15:41:45	1	1234	**Ja**
f_7	17:33:10	15:36 - 23:59	Malmö	Fredag	17:44:02	2	1234	Nej

Tabell 5-5: Exempel på träningsmängd där attributet *Avgångstid* har gjorts diskret och representeras med tidsintervall av det nya attributet *Avgångstid (diskret)*.

Den uppdaterade träningsmängden visas i Tabell 5-5 där både kontinuerliga och diskreta värden visas för *Avgångstid*. Genom att använda det diskreta attributet reduceras antalet olika värden från 7 till 5 vilket medför att trädet blir mindre och därmed förhoppningsvis mer effektivt. En annan fördel med de diskreta värdena är att de kan användas för att klassificera alla fall vars avgångstid överensstämmer med något tidsintervall. Ett träd som istället baseras på de kontinuerliga avgångstiderna kan endast klassificera fall där avgångstiden för fallet exakt stämmer överens med något observerat fall hos träningsmängden.

5.3.3 Exempel på ett beslutsträd

Figur 5-7 visar ett beslutsträd som har tränats med algoritmen TRÄNA-BESLUTSTRÄD-CHI-SQUARE med en träningsmängd bestående av 413 fall. Värdena för attributen *Avgångstid* eller *Tid på dygnet* har diskretiserats till tidsintervall med algoritmen GÖR-DISKRET-TIDSINTERVALL.



Figur 5-7: Beslutsträd som tränats med algoritmen TRÄNA-BESLUTSTRÄD-CHI-SQUARE med en träningsmängd bestående av 413 fall.

Beslutsträdet i Figur 5-7 används för att beräkna sannolikheten för att ett tåg som lämnar Malmö C kommer åka den unika rutten "Malmö C -> Stockholm C". Slutsatser om sannolikheten för att ett tåg kommer rutten kan dras genom att läsa beslutsträdet. Några exempel på slutsatser är:

- Sannolikheten är stor om tåget avgår mellan 03:10 och 03:22.
- Om tåget avgår mellan 04:03 och 04:23 är sannolikheten stor om föregående startstation är Stockholm C, men inte om föregående startstation är Köpenhamn H.
- Sannolikheten är liten om tåget avgår efter 17:25.

5.4 Rangordning av unika rutter baserat på sannolikhet

Resultatet för ett fall som klassificeras mot ett beslutsträd är en tvåvärd resultatvektor $v = (j, n)$. Då fallet för den pågående rutten f_p klassificeras mot beslutsträdet för varje unik rutt hos $\boldsymbol{U}(r_p)$ erhålls ett resultat för varje beslutsträd, vilket ger oss en mängd resultatvektorer. Problemet är att identifiera resultatvektorn med störst sannolikhet. Vi valde att lösa problemet genom att sortera, och rangordna, resultatvektorerna med en bestämd sorteringsordning och välja resultatvektorn med minst rang som den mest sannolika.

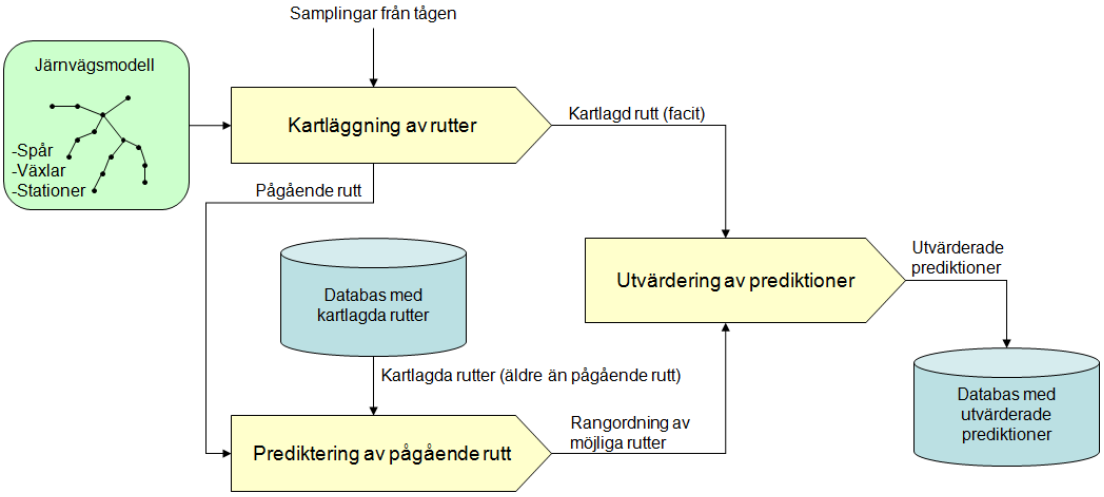
Följande sorteringsordning användes för att rangordna resultatvektorerna:

1. Sannolikheten *p(v)*, störst sannolikhet får lägst rang.
2. För resultat med samma rang: Antalet positiva utfall hos *v*, flest positiva utfall får lägst rang.
3. För resultat med samma rang: Antalet negativa utfall hos *v*, minst antal negativa utfall får lägst rang.
4. För resultat med samma rang: Rangordna slumpmässigt.

Den unika rutten för beslutsträdet med lägst rangordnade resultatvektor väljs som mest sannolik att beskriva resterande del av den pågående rutten r_p.

5.5 Metod för utvärdering av prediktion

De kartlagda rutter som identifierats under tidsperioden 2009-09-01 - 2009-10-31 sparades i en databas, se kapitel 4. Vi valde att utvärdera prediktionsalgoritmen genom att åter kartlägga tågens rutter för samma tidsperiod. Algoritmen för kartläggning av rutter bygger upp tågens pågående rutter. Prediktionsalgoritmen används för att prediktera hur de pågående rutterna kommer avslutas. Beslutsträden genereras med en ny träningsmängd om trädet är äldre än 24 timmar. Trädens träningsmängder baseras på de rutter som kartlagts minst en dag innan den pågående rutten påbörjades. Det finns alltså ingen risk att träningsmängden innehåller den kartlagda rutt som en pågående rutten kommer resultera i. En schematisk bild över metoden illustreras i Figur 5-8.



Figur 5-8: Schematisk bild över metod för utvärdering av prediktionsalgoritmen.

Prediktion av den pågående rutten sker då den pågående rutten utökas med ett nytt ruttelement, vilket inträffar då tåget passerar eller avgår från en ny station. Resultatet av prediktionen är en rangordnad lista med möjliga unika rutter som kan beskriva resterande del av den pågående rutten, den unika rutten med lägst rang anses mest sannolik. När kartläggningen av den pågående rutten avslutats vet vi vilken unik rutt, u_f, som motsvarar den kartlagda rutten. Den unika rutten u_f används som facit för att utvärdera de prediktioner som utförts under uppbygnaden av den pågående rutten. Utvärderingen innebär bland annat att avgöra vilken rangordning u_f hade vid de prediktioner som utförts. Utvärderingarna sparades i en databas där varje utvärdering bland annat innehåller följande information:

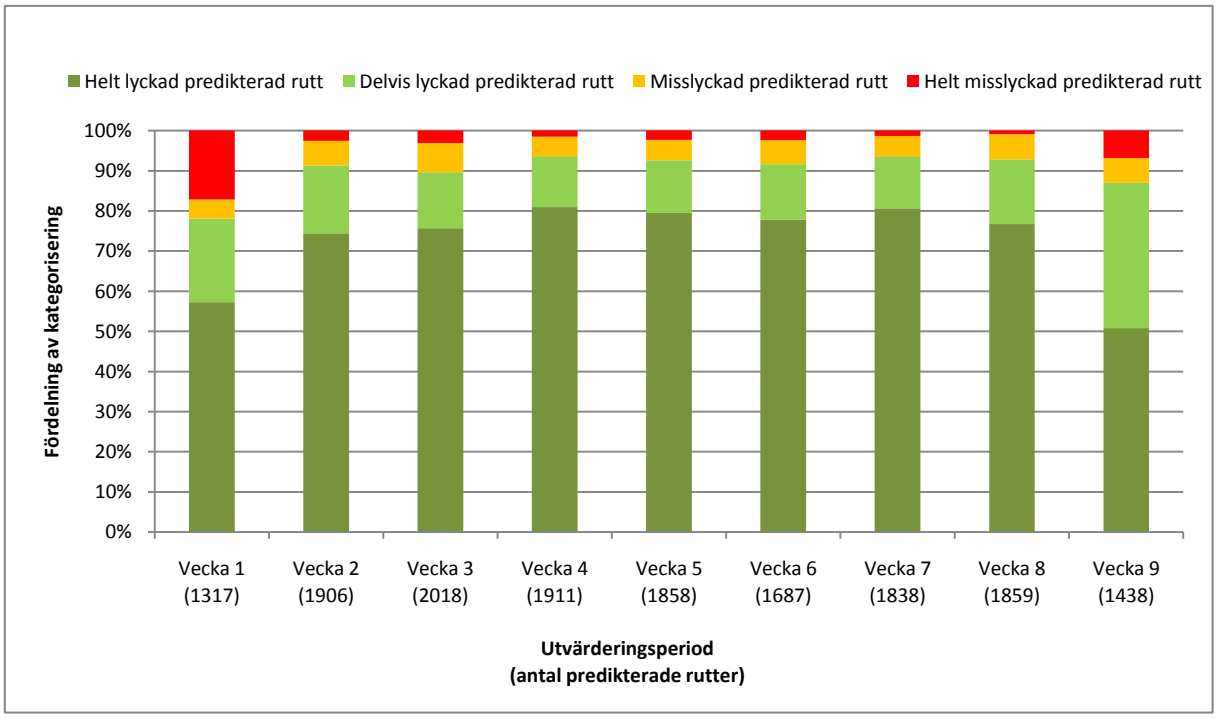
- *Rang*: Rangordning av u_f vid prediktionstillfället.
- *Antal möjligheter*: Antalet unika rutter hos mängden $\boldsymbol{U}(r_p)$ då prediktionen utfördes.
- *Restid:* Antal minuter som tåget färdats på sin rutt då prediktionen utfördes.
- *Total restid*: Ruttens totala tid i antal minuter.

Genom att analysera utvärderingarna från en pågående rutt kan det avgöras hur väl algoritmen lyckades prediktera hur den pågående rutten skulle avslutats. Vi har definierat fyra olika kategorier:

- *Helt lyckad predikterad rutt*: u_f har rang ett för alla utvärderingar av rutten.
- *Delvis lyckad predikterad rutt*: u_f har rang ett för minst en utvärdering av rutten.
- *Misslyckad predikterad rutt*: u_f har inte rang ett för någon utvärdering av rutten.
- *Helt misslyckad predikterad rutt*: u_f fanns inte med i mängden $\boldsymbol{U}(r_p)$.

5.6 Resultat – Analys av ruttprediktering

Vi har valt att presentera resultatet för utvärderingarna veckovis för tidsperioden 2009-09-01 - 2009-10-31 vilket blir nio utvärderingsperioder. Under utvärderingsperioderna användes algoritmen för att prediktera totalt 15 832 pågående rutter och totalt 357 294 prediktioner utfördes. Det som skiljer perioderna åt är storleken hos beslutsträdens träningsmängder då antalet kartlagda rutter ökar dag för dag under veckorna. Den första dagen under vecka 1 är träningsmängderna tomma eftersom antalet kartlagda rutter är noll. Under veckan ökar antalet kartlagda rutter för varje dag som går och träningsmängderna blir större. Den första dagen under vecka 2 baseras träningsmängderna på de rutter som kartlagts under vecka 1 och antalet kartlagda rutter ökar för varje dag under veckan. På samma sätt ökar antalet kartlagda rutter, och därmed träningsmängderna, för vecka 3 och så vidare.



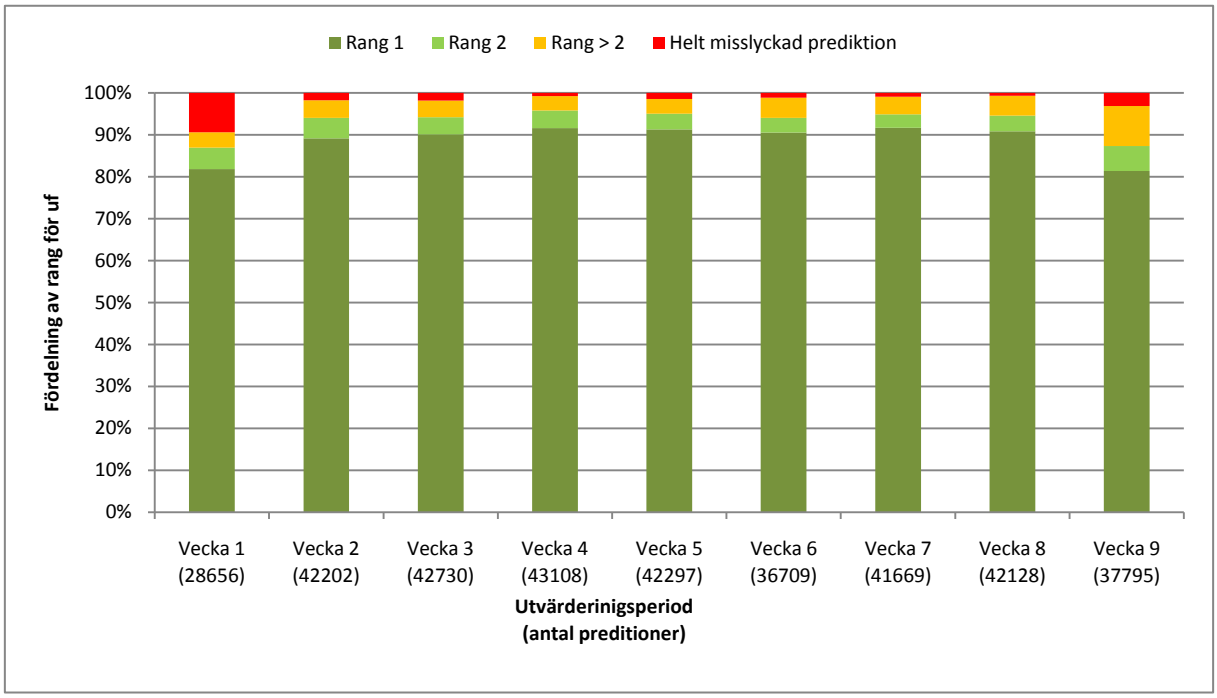
Figur 5-9: Fördelning av hur väl prediktionsalgoritmen lyckades prediktera rutter per utvärderingsperiod.

I Figur 5-9 ser vi antal rutter som predikterats under respektive utvärderingsperiod och hur väl algoritmen har lyckats prediktera rutterna för utvärderingsperioden. Vi ser att flest rutter predikterades under vecka 3 och minst antal predikterades för vecka 1.

Under vecka 1 kategoriseras cirka 17 % av de predikterade rutterna som *Helt misslyckad predikterad rutt*. Resultatet är rimligt då antalet kartlagda rutter under veckans första dag är noll och alla prediktioner under dagen resulterade i att u_f inte fanns med i mängden $\boldsymbol{U}(r_p)$.

Andelen förutsagda rutter som kategoriseras som *Helt lyckad predikterad rutt* ökar från vecka 1 till vecka 4 och är relativt stabilt från vecka 4 fram till och med vecka 8. Under dessa veckor kategoriseras cirka 80 % av de predikterade rutterna som *Helt lyckad predikterad rutt*, och mer än 90 % som *Helt lyckad predikterad rutt* eller *Delvis lyckad predikterad rutt*.

Under vecka 9 minskar andelen rutter som kategoriseras som *Helt lyckad predikterad rutt* till från cirka 80 % till cirka 50 % och andelen *Delvis lyckad predikterad rutt* och *Helt misslyckad predikterad rutt* ökar markant.



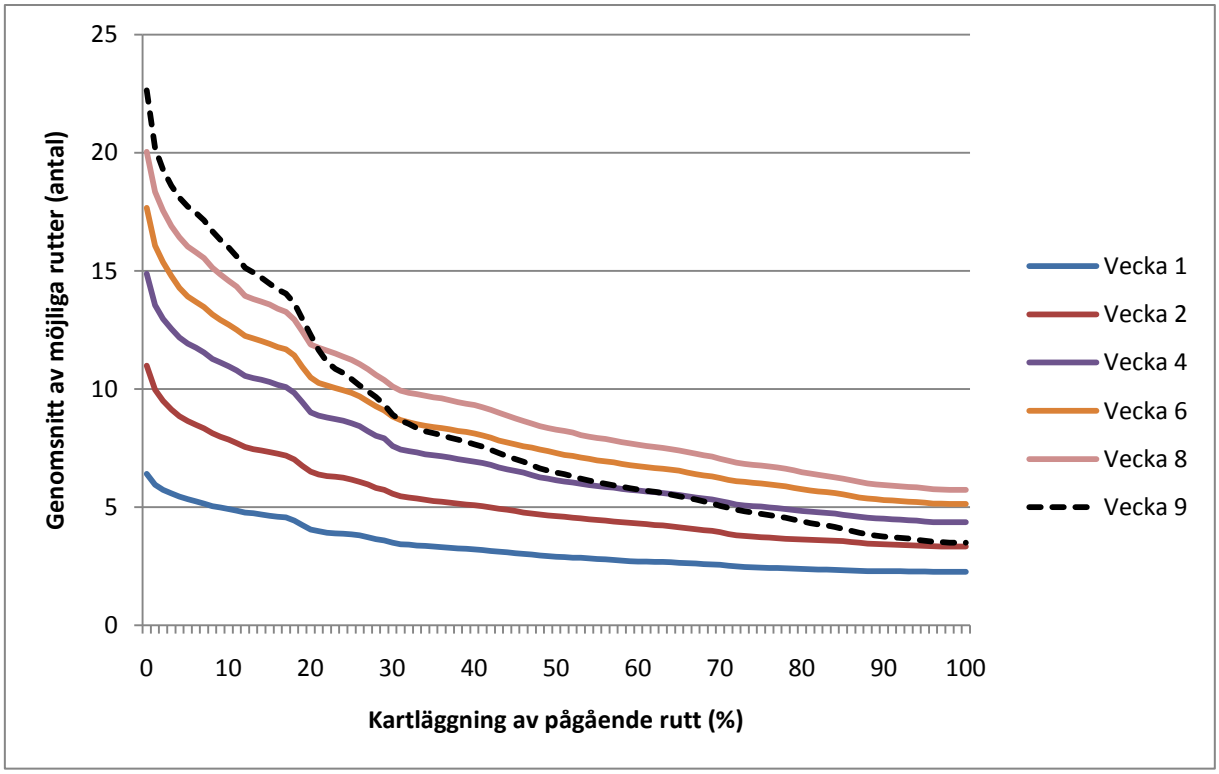
Figur 5-10: Fördelning av rang för u_f för prediktioner som utfördes per utvärderingsperiod.

I Figur 5-9 ser vi antalet prediktioner som utfördes under respektive utvärderingsperiod och hur väl algoritmen lyckades rangordna u_f, den unika rutt som visade sig vara rätt. En prediktion utfördes varje gång tågets pågående rutt utökades med en ny station. Vi ser att flest prediktioner utfördes under vecka 4 och minst antal utfördes under vecka 1.

För veckorna 2 till och med 8 lyckas algoritmen identifiera rätt unik rutt för cirka 90 % av prediktionerna. Under dessa veckor lyckas algoritmen rangordna rätt unik rutt bland de två mest sannolika för cirka 95 % av prediktionerna.

Under vecka 1 är cirka 10 % av prediktionerna helt misslyckade, det innebär att den unika rutten som tågets pågående rutt resulterade i inte fanns med i mängden $\boldsymbol{U}(r_p)$ då prediktionen utfördes. Enligt tidigare resonemang är resultatet rimligt då mängden kartlagda rutter är tom för första dagen under vecka 1 och alla prediktioner misslyckas.

För vecka 9 minskar andelen prediktioner där algoritmen lyckades identifiera rätt unik rutt från 90 % till ca 80 %.

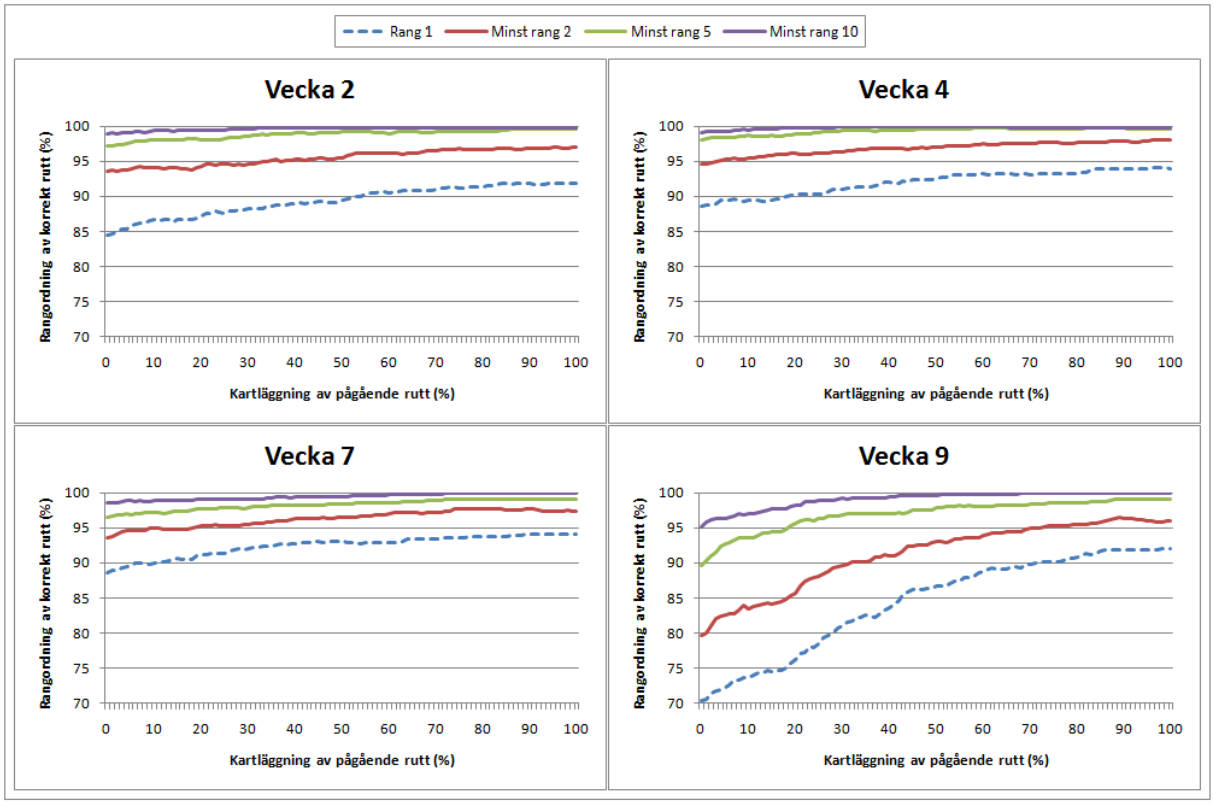


Figur 5-11: Genomsnitt av möjliga unika rutter av procent för pågående rutt.

Figur 5-11 visar genomsnittet för antalet unika rutter hos mängden $\boldsymbol{U(r_p)}$, beroende på hur stor andel av den pågående rutten r_p som är kartlagd för några av utvärderingsperioderna. Diagrammet visar även att antalet unika rutter hos mängden $\boldsymbol{U(r_p)}$ avtar ju större del av r_p som är kartlagd.

Antalet möjliga unika rutter är minst för vecka 1 och ökar kontinuerligt vecka för vecka, vilket innebär att kartläggningsalgoritmen identifierar ungefär lika många nya unika rutter varje vecka. Resultatet kan tyda på en brist i kartläggningsalgoritmen och beskrivs i slutsatserna i avsnitt 4.9.

Kurvorna för vecka 1 till och med vecka 8 är av samma karaktär med en förskjutning i antalet möjliga rutter. Kurvan för vecka 9 är av en annan karaktär vilket kan tyda på att tågen har trafikerat andra rutter för denna utvärderingsperiod eller att enbart vissa tåg har trafikerat järnvägsnätet.



Figur 5-12: Diagram för rangordning av u_f beroende på kartlagd andel för pågående rutter under veckorna 2, 4, 7 och 9.

Figur 5-12 visar rangordningen för u_f beroende på hur stor andel av den pågående rutten r_p som är kartlagd. Under vecka 2 lyckades prediktionsalgoritmen korrekt identifiera u_f vid 84 % av prediktionerna precis då kartläggningen av de pågående rutterna påbörjades. När de pågående rutten kartläggs ökar andelen lyckade prediktioner och vid ruttens slut har andelen lyckade prediktioner ökat till cirka 92 %. Att andelen lyckade prediktioner ökar när den pågående rutten kartläggs är rimligt eftersom antalet möjliga rutter minskar när rutter kartläggs.

För vecka 4 och 7 har andelen lyckade prediktioner vid kartläggningens start ökat till cirka 88 % och ökar sedan till cirka 94 % när kartläggningen av den pågående rutten avslutas.

För vecka 9 ser vi att andelen lyckade prediktioner är cirka 70 % när kartläggningen påbörjas och stiger sedan till cirka 92 % när kartläggningen avslutas.

5.7 Slutsatser – Analys av ruttprediktering

Syftet med analysen var att utreda om tågens pågående rutter kan predikteras med en utvald algoritm. Prediktionsalgoritmen, som baseras på beslutsträd, kan användas för att prediktera resterande del av tågens pågående rutter. Som bäst lyckas algoritmen prediktera cirka 80 % av rutterna som körs under en vecka helt korrekt, det vill säga att varje prediktion som utfördes från att rutten påbörjades till att den avslutades var korrekt. Ruttprediktering utfördes enbart utifrån samplingar från tågen och använde inte någon extern information som t.ex. tidtabeller. Predikteringen är lämplig för mindre kritiska tillämpningar. Några exempel på sådana tillämpningar kan vara:

- Information och reklam till passagerare beroende på tågets destination.
- Möjlighet för Icomeras support att söka efter tåg beroende på destination.
- Dead-reckoning vid längre avbrott i positionsrapportering.

Algoritmens resultat varierar för olika veckor, vilket gör det svårt att dra någon slutsats om hur väl algoritmen lyckas prediktera rutter för tidsperioder som inte analyserats.

Eftersom vi vet att det finns felaktigheter i de kartlagda och pågående rutterna, som är indata för algoritmen, är det svårt för oss att dra slutsatser om felaktig prediktion beror på felaktig indata eller brister hos algoritmen.

En annan slutsats är att större träningsmängd, i form av avslutade rutter, ej garanterar fler lyckade prediktioner. För vecka 9, som har störst träningsmängd, ser vi att andelen lyckade förutsagda rutter är minst. Algoritmen har bäst resultat, med cirka 90 % korrekta prediktioner, för vecka 4 till och med 8, vilket tyder på att algoritmen inte lyckas bättre om träningsmängden baseras på fler kartlagda rutter än tre till fyra veckor bakåt i tiden.

5.8 Diskussion – Analys av ruttprediktering

Algoritmen för kartläggning av rutter har en del brister (se Kap 4) som påverkar predikteringen av rutter negativt. Bristerna medför att mängden kartlagda rutter innehåller en del felaktigt kartlagda rutter. Det innebär även att den pågående rutten, som kartläggs i realtid, ibland är felaktig. Detta påverkar predikteringen på två sätt, del genom att den pågående rutten, som ska predikteras ibland är felaktig, men även att det statistiska underlaget i form av kartlagda rutter är felaktigt. Ungefär 90 % av prediktionerna är korrekta och de 10 % som är felaktiga kan till stor del bero på bristerna i kartläggningsalgoritmen.

I resultatet ser vi att andelen lyckade prediktioner är minst vecka 9, detta trots att flest antal kartlagda rutter finns tillgänga för denna period. Det kan hända att resultatet försämras på grund av att träningsmängderna blir för stora, men det kan även vara så att tågen har börjat trafikera nya rutter för denna utvärderingsperiod. Utvärderingsperioden för vecka 9 baseras på samplingar för dagarna mellan 2009-10-26 och 2009-10-31. Under perioden 2009-09-10 till och med 2009-10-25 var tågtrafiken på Svealandsbanan mellan Södertälje och Läggesta avstängd (Trafikverket, 2010). En hypotes kan vara att tågen började trafikera Svealandsbanan 2009-10-26 vilket medför att nya rutter kartlades. De nya rutterna ansågs inte sannolika av predikteringssalgoritmen som istället ansåg de äldre, mer frekvent trafikerade rutterna som mer sannolika. Problemet uppstår när tågtrafiken förändras och en möjlig lösning kan vara att väga de kartlagda rutterna olika beroende på hur länge sedan de kartlades, eller alternativt endast betrakta till kartlagda rutter som ej är för gamla.

Då problemet är att identifiera rutten med högst sannolikhet kan en enkel statistisk modell ge bättre resultat än beslutsträden. Den enkla modellen skulle kunna användas för att beräkna den betingade sannolikheten för möjliga rutter med Bayes teorem. För att göra detta måste man ta reda på vilka faktorer som är mest avgörande för sannolikheten.

6 Slutsatser och diskussion

Vi har utvecklat en applikation där användaren kan skapa en järnvägsmodell utifrån tågens GPS-positioner. Modellen representerar järnvägsnätets spår, växlar och stationer. Processen för skapandet av järnvägsmodellen är till stor del automatiserad men kräver även en del manuellt arbete av användaren. Vi anser att lösningen vi utvecklat skapar en tillräckligt bra järnvägsmodell för kartläggning av de rutter som tågen trafikerar.

Upplösningen hos järnvägsmodellen som vi skapat är maximalt 100 meter vilket medför att parallella spår som ligger närmare än 100 meter från varandra representeras som ett enda spår. Det innebär att modellen inte kan användas för att till exempel avgöra vilket spår ett tåg står på vid en station.

Algoritmen för att identifiera stationer har en del brister, framför allt identifierar algoritmen platser där det inte finns någon station i verkligheten. Dessa platser kan vara stoppsignaler, mötesplatser eller andra platser där tåg ibland stannar. Vi rekommenderar därför att användaren istället ska definiera vilka stationer som är av intresse och endast låta algoritmen identifiera dessa.

Mer information kan ges utifrån ett tågs GPS-position om den projiceras på vår modell av järnvägsnätet. Vi kan då ge information om vilken station tåget just passerat och vilken station den är på väg till. Modellen av järnvägsnätet skulle kunna utökas för att innehålla information om vart tunnlar finns och vart det är bra respektive dålig täckning för de mobila länkarna, för att underlätta prioritering av länkar. Modellen skulle även kunna utökas med information om stoppsignaler, dubbel- och enkelspår samt hastighet på rälssträckor.

Järnvägsmodellen kan även användas till att implementera en dead reckoning algoritm som tar hänsyn till järnvägens utsträckning och ser till att tågets position alltid rapporteras in på järnvägsrälsen. Denna algoritm skulle även kunna förflytta tåget mer frekvent än vad samplingarna kommer in.

Järnvägsmodellen och samplingarna från tågen kan användas för att analysera hur tågen trafikerar järnvägsnätet och kartlägga tågens rutter. Kartläggningen identifierar rutternas start- och slutstation, samt vilka stationer som passeras och stannas vid längs rutten. De kartlagda rutterna innehåller mönster vilket gör det möjligt att använda dem för att prediktera hur tågen kommer trafikera järnvägsnätet. Rutterna skulle eventuellt även kunna användas för att analysera hur trafiken förhåller sig till de officiella tidtabeller, med syfte att skapa statistik över förseningar med mera.

Det är problematiskt att identifiera rutternas start- och slutstationer vilket gör att kartläggningsalgoritmen ibland kartlägger vissa rutter felaktigt. Det innebär att en del rutter avslutas på ett felaktigt sätt, t.ex. om ett tåg stannar för länge på en station som egentligen inte är ruttens slutstation. Den felaktiga kartläggningen ger negativa konsekvenser för predikteringen.

För att förbättra kartläggningen av rutter föreslår vi en vidareutveckling av algoritmen. Genom att specificera de rutter som är intressanta att kartlägga behöver algoritmen inte identifiera rutterna, utan endast kartlägga hur de intressanta rutterna trafikeras. De rutter som är intressanta skulle man kunna få tillgång till från tågoperatörens tidtabeller. Vi tror att detta skulle förbättra kartläggningens resultat avsevärt.

Vi har visat att det går att prediktera tågens rutter enbart utifrån samplingar från tågen, utan att använda någon extern information som t.ex. tidtabeller. Som bäst predikteras cirka 80 % av rutterna helt korrekt och därför anser vi att predikteringen lämpar sig för mindre kritiska tillämpningar, t.ex. information och reklam till användare beroende på tågens destinationer. Enligt tidigare resonemang är en anledning till att fler rutter inte predikteras korrekt att kartläggnings-algoritmen ibland misslyckas identifiera rutternas start- och slutstationer.

7 Referenser

1. Ashbrook, D. Starner, T (2003) Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, vol. 7, nr 5, ss 275-286.
2. EKF CompactPCI Products (2002) *Production Information CG1-Radio GPS Receiver*, http://www.ekf.de/c/cgps/cg1/cg1pie.pdf
3. Eniro Sverige AB (2009) *Eniro kartor*. http://kartor.eniro.se/ (20 Okt 2009)
4. Froehlich, J. Krumm, J. (2008) Route Prediction from Trip Observations. *Intelligent Vehicle Initiative (IVI) Technology Controls and Navigations Systems*
5. Heyer, L.J., Kruglyak, S., Yooseph, S., (1999) Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Research*, vol 9, ss. 1106-1115.
6. Langley R. (1999) Dilution of Precision. *GPS-World*, vol. 10, nr. 5, ss 52-59.
7. Michels, C. (1997) Latitude/Longitude Distance Calculation. http://jan.ucc.nau.edu/~cvm/latlongdist.html#formats, (27 Apr 2010)
8. Russell, S. Norvig, P. (2003) Learning from observations. I *Artificial Intelligence: A Modern Approach*, 2:a utgåvan, ss. 649-664, New Jersey: Pearson Education Inc. 2003.
9. Simmons, R. Browning, B. Zhang, Y. Sadekar V. (2006) Learning to Predict Driver Route and Destination Intent. *Proceedings of the IEEE ITSC 2006*, 17-20 September, Toronto, Canada.
10. SJ AB (2009) *SJ.se* http://www.sj.se, (20 Okt 2009)
11. Trafikverket (2010) *Deletapper Svelandsbanan*, http://www.trafikverket.se (30 Apr 2010)

8 Appendix

8.1 Identitetsnummer för analyserade tåg

1048	1081	1109	1161	1178
1050	1082	1110	1162	1179
1051	1083	1111	1163	1180
1054	1084	1112	1164	1181
1055	1085	1119	1165	1182
1058	1086	1120	1166	1183
1062	1087	1121	1167	1184
1070	1088	1123	1168	1185
1072	1089	1124	1169	1186
1073	1090	1125	1170	1187
1074	1091	1126	1171	1188
1075	1092	1127	1172	1189
1076	1093	1128	1173	1190
1077	1094	1129	1174	1191
1078	1095	1135	1175	1192
1079	1096	1142	1176	
1080	1107	1160	1177	

8.2 Platser med kända positioner

Agnesberg
Alingsås
Almvik
Alnarp
Alvesta
Alvhem
Arboga
Arholma
Arlanda
Arvika
Baggetorp
Biskopstorp
Bjurhem
Björneborg
Björnkulla
Björnlunda
Bohus
Borlänge
Borås
Brålanda
Bräcke
Brännarp
Bålsta
Båstad norra
Bäckefors
Degerfors
Duved
Ed
Eldsberga
Enafors
Enköping
Erikslund
Eskilstuna
Eslöv
Falkenberg
Falköping
Falun C
Flackarp
Flemingsberg
Flen
Floby
Fredrikstad
Frillesås
Frändefors
Fränsta
Furusund
Glyxnäs
Gnesta
Gubbero
Gävle
Göteborg C
Hagalund (D)
Halden
Hallandsåsen
Hallsberg
Halmstad C
Hamra
Hasselfors
Heberg
Helsingborg C
Herrljunga
Horred
Huddinge
Hudiksvall
Härnösand
Hässleholm C
Högboda
Högsjö
Höör
Järna
Järpen
Jönköping
Kallhälls station
Karlstad C
Kastrup Lufthavn
Katrineholm C
Kil
Killsmo
Kolmården
Kornsjö
Kristinehamn
Krokom
Kumla
Kungsbacka
Kållered
Kävlinge
Köpenhamn H
Köping
Laholm väster
Landskrona Östra
Laxå
Lindalen
Lindome
Linköping C
Linköping Depot
Ljusdal
Lund C
Malmsjö
Malmö C
Malmö vagnhall
Mjölby
Moss
Mölnbo
Mölndal Nedre
Nol
Norrköping
Norrsund
Nyckelsjön
Nygård
Nykvarn
Nyköping
Nässjö C
Olskroken
Osby
Pepparholm
Pressebo
Pålsboda
rattvik
Råskogen
Rödlöga
Sandviken station
Sarpsborg
Skattkärr
Skebokvarn
Skåre
Skälebo
Sköldinge
Skövde C
Sparreholm
Stjärnhov
Stockholm C
Stockholm Syd
Stolpstugan
Storlien
Strängnäs
Strömstad
Strömtorp
Stöde
Sundbyberg
Sundsvall
Surte
Svartå
Svågetorp
Söderhamn
Södertälje Syd
Tierp
Torebo
Torpshammar
Tranås
Trollhättan C
Tyllered
Tälle
Töreboda
Uddevalla C
Undersåker
Upphärad
Uppsala
Vagnhärad
Vallkärra
Varberg C
Vaxholm
Vegeholm
Vejbyslätt
Velanda
Vingåker
Vretstorp
Vårgårda
Värö
Väse
Västerås
Västerås Depot
Åkarp Norra
Åmotfors
Ånge
Åre
Årnes
Årstabron
Åsa
Älmhult
Älvsjö
Älvängen
Ängelholm
Ödåkra
Ölme
Örebro C
Östansjö
Östanå Färjeläge
Östersund
Öxnered

8.3 Identifierade stationer

Alingsås - Station
Alvesta - Station
Arboga - Station
Arvika - Huvudstation
Avesta - Station
Bollnäs - Station
Borlänge - Station
Bräcke - Station
Bålsta - Station
Degerfors - Station
Duved - Huvudstation
Enköping - Station
Eskilstuna - Station
Falköping - Station
Falun C - Station
Flemingsberg - Station
Flen - Station
Furuvik - Station
Gnesta - Station
Gävle - Station
Göteborg C - Huvudstation
Hagalund (D) - Depå
Hallsberg - Station
Halmstad C - Station
Heby - Station
Herrljunga - Station
Hudiksvall - Station
Härnösand - Huvudstation
Härnösand depå - Depå
Hässleholm C - Station
Iggesund - Station
Jakobsberg - Station
Järsö - Station
Jönköping - Huvudstation
Jönköping öster - Station
Kallhälls station - Station
Karlstad C - Station
Kastrup Lufthavn - Station
Katrineholm C - Station
Kil - Station
Knivsta - Station
Kolbäck - Station
Kolmården - Station
Kristinehamn - Station
Kumla - Station
Kungsbacka - Station
Kungsör - Station
Kvicksund - Station
Köpenhamn H - Huvudstation
Köping - Station
L. Sundby - Station
Linköping C - Station
Linköping Depot - Depå
Ljusdal - Station
Lund C - Station
Malmsjö - Station
Malmö C - Huvudstation
Malmö Syd - Station
Malmö vagnhall - Depå
Mjölby - Station
Morgongåva - Station
Munktorp - Station
Mölndal Nedre - Station
Norrköping - Station
Nykvarn - Station
Nyköping - Station
Nässjö C - Station
Ockelbo - Station
Olskroken - Depå
Persborg - Station
Ransta - Station
Rotebro - Station
Sala - Station
Sandviken station - Huvudstation
Skutskär - Station
Skövde C - Station
Stenstorp - Station
Stockholm C - Huvudstation
Stockholm Syd - Station
Strängnäs - Station
Sundbyberg - Station
Sundsvall - Station
Söderhamn - Station
Södertälje Syd - Station
Tierp - Station
Tillberga depå - Depå
Timrå - Station
Tranås - Station
Töreboda - Station
Uddevalla C - Huvudstation
Uppsala - Station
Vagnhärad - Station
Vara - Station
Varberg C - Station
Vingåker - Station
Värnersborg - Station
Västerås - Station
Västerås Depot - Depå
Ånge - Station
Örbyhus - Station
Örebro C - Station
Örestad station - Station
Östersund - Station