KLASSIFISERING AV BEVEGELSESMØNSTER HOS ORIENTERINGSLØPERE

CLASSIFYING MOTIONTYPES IN ORIENTEERING

JENS TORBJØRN SAGBERG

UNIVERSITETET FOR MILJØ- OG BIOVITENSKAP
INSTITUTT FOR MATEMATISKE REALFAG OG TEKNOLOGI, IMT
MASTEROPPGAVE 30 STP. 2012

Sammendrag

Med de siste års utvikling av kommunikasjonsteknologi, GNSS-mottakere og -sendere er orienteringssporten på vei ut av skogen og inn i hjemmet. Direkte overføring av løpernes posisjon fra GNSS mottakere via mobilnettet til en server og deretter videre ut på storskjerm og internett, gjør at man nå kan følge løpernes bevegelser fra start til mål. Med et stort antall løpere som følges kan det bli vanskelig å få med seg alt. Jeg har derfor i denne oppgaven forsøkt å finne en metode for å kunne klassifisere bevegelsesmønsteret til orienteringsløpere, for automatisk å kunne oppdage når løperne har problemer med å finne posten.

Jeg har valgt å benytte et såkalt *sliding window*, glidende vindu, en algoritme som behandler de til enhver tid *n* siste observasjoner. Når en ny observasjon kommer til fjernes den eldste, og nye beregninger foretas. For å klassifisere dataene valgte jeg å se på forholdet mellom løpt distanse på de *n* siste observasjonene og avstanden fra den første til den siste observasjonen i vinduet.

For å finne en verdi for hva som bør klassifiseres som bom, ble det samlet inn data med GNSS-mottakere. Den samme løypen ble løpt fem ganger, og under innsamlingen av data ble det lagt inn en del typiske bommer som ofte går igjen i orientering. Under analysen av testdataene fant jeg at forholdet mellom luftavstand og løpt distanse falt til under 0,8 når løperen bommet. For å få systemet følsomt nok ble vinduet satt til fem observasjoner, noe som tilsvarer cirka 30 sekunders løping, avhengig av GNSS-enhetens oppdateringsfrekvens.

For å teste om metoden også fungerte på andre datasett benyttet jeg data fra et norgescupløp. Data fra mange større orienteringskonkurranser ligger tilgjengelig på internett for de som ønsker å se dette i ettertid. GNSS-dataene er også mulig å laste ned for egne analyser. Kontrolldataene avdekket noen svakheter med algoritmen. Siden løperne ofte skal endre retning på postene, er det avgjørende, for å unngå falske positive utslag, at dette tas hensyn til. For å registrere løperne på postene ble det satt en buffer på 17 meter rundt postens koordinater, dersom løperen var innenfor denne avstanden ble det antatt at posten var besøkt. Dessverre kan det virke som om terrenget i kontrollområdet stilte andre krav til postregistreringen, noe som ga et stort antall falske positive utslag. En annen svakhet, som ikke ble avdekket siden kontrollterrenget hadde en del likhetstrekk med testterrenget, er hva som skjer når det på grunn av hindringer i terrenget ikke er mulig for løperne å holde en like rett linje over kortere tidsperioder.

80 av 97 bom i kontrolldatasettet ble oppdaget. 96 av 176 utslag i kontrolldataene ble feilaktig klassifisert som bom. Hele 93 av disse var umiddelbart etter postpassering og bør kunne fjernes med en bedre metode for registrering på postene.

Abstract

Due to the rapid evolution in communication technique and GNSS receivers and transmitters, the possibility to follow the big competitions in the sport of orienteering is now getting available in the many homes spread around the world. Live broadcast of the runner's position collected by GNSS receivers, transmitted through mobile broadband, computed in a server and then sent out on the internet and television, makes it possible for everyone to follow all the action going on in the forest. With the increasing amount of information available, a method to automatically sort out the most interesting moments is needed. I will therefore, in this paper, try to find an algorithm that can classify the runner's pattern of motion, and let us know when their behavior is now longer as expected.
To analyze the data, I have chosen to use a *sliding window*.This is a method that takes only in consideration the *n* last observations, and leaving out the oldest observation as a new one gets available. To classify the motion I have looked at the factor between the total distance run during the last *n* observation and the distance between the first and last observation in the window.

The test data was collected with GNSS watches. A course was run five times, including some normal mistakes made by orienteers. This dataset was then used to find the factor representing normal behavior and the limit for what should be classified as a mistake. During this analysis, I found that an appropriate value for mistakes would be when the factor exceeds 0.8. Also the size of the window was tested, leaving me with the best result for a window stretching about 30 seconds.

To verify the method, I used data from 37 runners in a national competition. This revealed some weaknesses in the algorithm. First, there were a lot of false positive results; most of these immediately after a control. With the runners often changing the direction when heading for the next control, I made a method to find out when a runner had found a control. With a much slower terrain in the control area than in the test area it seems like the runners have been confirmed at a control too soon, making the wanted change of direction at the control classified as unwanted motion, a mistake. A more sophisticated algorithm to decide whether or not a runner have passed the given control, would eliminate most of the false positive results. Another big weakness, unfortunately not revealed due to similarities between the test area and the control area, is the possibility that terrain formations, vegetation or other obstacles forcing runners into motion, in this paper, classified as unwanted.

Innholdsfortegnelse

1. Innledning

Orientering har i mange år vært en idrett mest for utøverne. Orienteringens egenart har, da konkurransene i all hovedsak har foregått ute i skogen, på mange måter utelukket mulighetene for publikum til å følge løperne mer enn ved start, målgang og ved eventuelle passeringer, i tillegg til passeringstider, via radio, fra arrangøren ute på postene. Med teknologiens fremskritt med stadige mindre og lettere GNSS-enheter og etterhvert også GNSS-trackere, har det blitt mulig å følge dramatikken ute i skogen direkte via GPS-tracking på storskjerm i målområdet, eller via internett og på TV hjemme i stuen.Mens det for inntil et par år siden kun var mulig å følge de aller beste på sin ferd gjennom skogen, øker nå antallet løpere som blir utstyrt med GNSS utstyr. Med mange løpere i skogen på en gang, kan det være vanskelig for produsent, speaker og publikum å få med seg alt som skjer og det er fort gjort å gå glipp av avgjørende øyeblikk og interessante bommer.

1.2.Problemstilling og målsetting

-Hvordan kan man utnytte tilgjengelig teknologi for å klassifisere bevegelsesmønsteret til en orienteringsløper?
-Hvordan kan man best avgjøre hva som er et uønsket bevegelsesmønster for løperen, kun ved hjelp av løperens posisjon?

Heldigvis finnes det verktøy for å håndtere store datamengder, og jeg vil i denne oppgaven forsøke å finne en metode for å klassifisere bevegelsesmønsteret til en orienteringsløper, slik at eventuelle bommer kan detekteres automatisk og publikum ikke går glipp av de avgjørende øyeblikkene, mens de skjer.

2. Teori

2.1. Orientering

2.1.1. Historikk

Idretten orientering startet i det svenske militæret på slutten av 1800-tallet. Den første sivile orienteringskonkurransen ble gjennomført i Nordmarka i 1897. Norgesmesterskap ble første gang arrangert i 1937. I 1945 ble Norges Orienteringsforbund stiftet, og i 1961 det internasjonale orienteringsforbundet. I 1966 ble det første VM arrangert i Finland, og deretter hvert annet år (med unntak av at det ble arrangert både 1978 og 1979) frem til 2003. Nå arrangeres VM hvert år[1].

2.1.2.Om Orientering

Grunnidéen i orientering er selvvalgt vei i ukjent terreng. Målet er å ta seg fra start til mål raskest mulig, med hjelp av kun kart og kompass. På kartet er det en løype med et antall poster man må besøke i riktig rekkefølge. Løypen er markert på kartet med sirkler rundt detaljen posten befinner seg på, sammenbundet med streker og nummerert i rekkefølgen postene skal besøkes. I terrenget er postene markert med oransje-hvite flagg. Flaggets eksakte plassering på detaljen er dessuten beskrevet i postbeskrivelsen[1]. For at løperen skal være trygg på at han er på riktig post, er det en unik kode på hver av postene. Denne koden står rett etter postnummeret i postbeskrivelsen. Postbeskrivelsen kan også inneholde annen informasjon, som f.eks. løypelengde, at det finnes drikke på enkelte poster (markert med et lite drikkebeger i den siste kolonnen) og obligatoriske strekk med merket løype underveis.

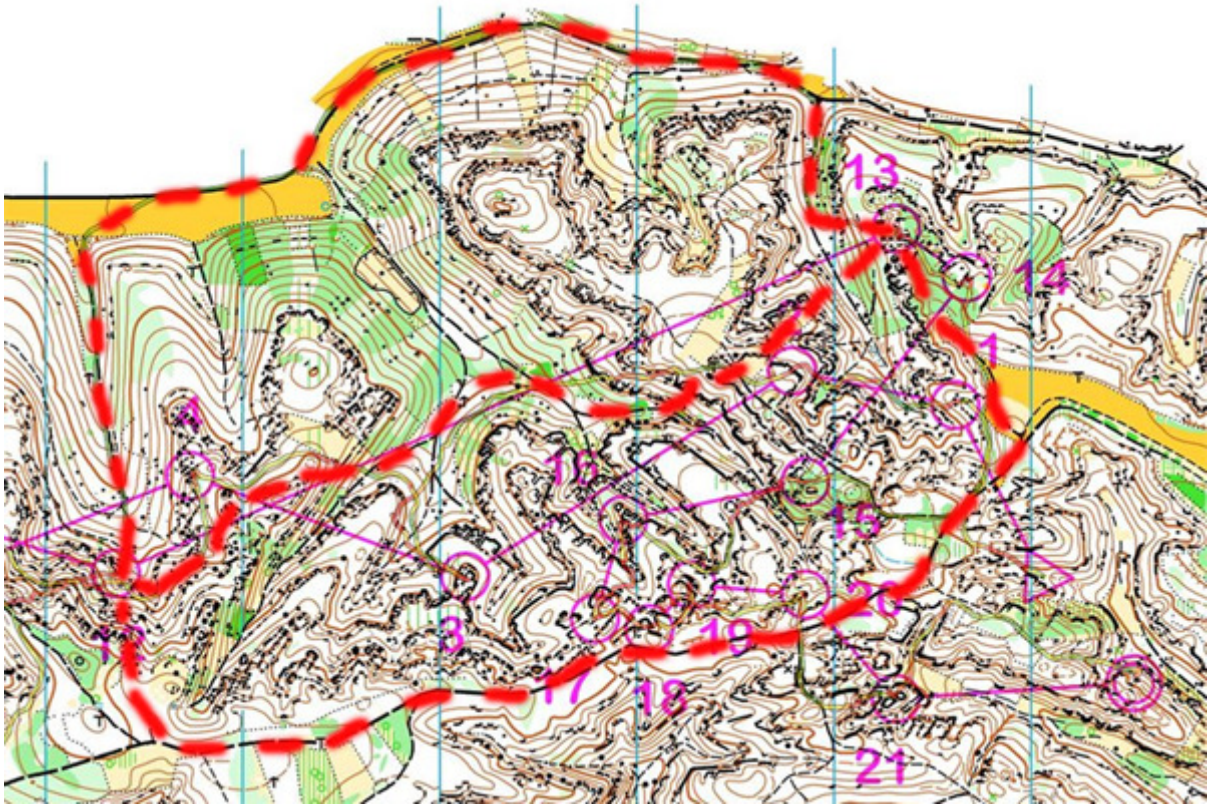
Bilde 2.1. Kart fra et orienteringsløp. Starten er markert med en trekant, postene med sirkler og nummerert og mål med en sirkel inne i en annen. Oppe til høyre kan man se postbeskrivelsen. Her finner løperen informasjon om kodesiffer, hvilken detalj posten befinner seg på, informasjon om detaljen samt hvor på detaljen flagget er plassert.

2.1.3. Løypelegging

Generelt for alle disiplinene er ønsket om variasjon, både i avstanden mellom postene, terrengtype og hvor intensiv selve orienteringen er. Variasjon stiller krav til løpernes allsidighet og deres evne til å tilpasse fart og orienteringsteknikk til utfordringene de har blitt gitt. Ulike typer strekk er:

-Veivalgsstrekk
-Kartlesningsstrekk
-Avledningsstrekk
-Transportstrekk

Veivalgsstrekkene er lagt med tanke på at utøverne skal vurdere hvilken vei som er raskest. Enkle veivalgsstrekk består ofte av kun å vurdere hvilken vei er kortest, mens på mer avanserte strekk blir løperen nødt til å vurdere flere elementer, som avstand, høydeforskjell, løpbarhet og vanskelighetsgrad, opp mot hverandre. Normalt sett er veivalgsstrekk lange, men det er også mulig å lage veivalgsproblematikk på kortere strekk. Store høydeforskjeller og områder med nedsatt løpbarhet er gode elementer for å tvinge løperne bort fra streken mellom postene og inn i utfordringen med å finne den raskeste veien til neste post. Et godt veivalgsstrekk inneholder ikke bare et enkelt valg mellom den høyre eller venstre stien, men gir mulighet for 3-4 hovedvarianter, med flere mindre vei-/trasé- valg underveis. I terrengområder med ekstreme hindringer i forkant av posten kan det til og med være aktuelt å løpe forbi posten, for å ta bakveien.



Bilde 2.2. Eksempel på terreng som innbyr til krevende veivalgsstrekk. Vegetasjon, ekstreme høyder kombinert med flate platåer, upasserbare stenformasjoner og lettløpte stier gir løypeleggeren mange muligheter å legge løyper med mange gode veivalg. 3 mulige veivalg fra post 12 til 13 inntegnet.

Kartlesningsstrekklegger opp til intensiv orientering, gjerne i områder med detaljert terreng. For å holde høy hastighet i på slike strekk, blir utøveren tvunget til å sile ut overflødig informasjon, både fra kartet og terrenget og kun benytte sikre, tydelige detaljer for å navigere seg frem i terrenget.

Avledningsstrekk benyttes til å knytte sammen resten av strekkene, unnvike for spisse vinkler inn og ut av postene (så ikke løperne møter hverandre og blir ledet inn i posten) og for å lede løperne utenom enkelte områder (viltlommer, privat mark mm). Det gjør ingenting om slike strekk også er utfordrende orienteringsteknisk, men hovedmålet er å skape en helhet i løypen.

Det vil ofte være deler av terrenget som ikke er spesielt interessant, verken med tanke på veivalg eller kartlesing. Istedenfor å lage postplukk med en mengde uinteressant kortstrekk kan løypeleggeren velge å dra løperne raskest mulig gjennom området. Med en vanskelig post i enden kan det dessuten bli mye bom for løpere som ikke setter ned igjen farten etter å ha hatt enkel orientering en stund. [2]

2.1.4. Løpsdisipliner

I orientering konkurreres det i flere ulike disipliner.
-Den tradisjonelle langdistansen, tidligere kalt klassisk distanse. Vinnertid ca. 90-100 minutter for herrer, 70-80 minutter for damer
-Mellomdistanse. Vinnertid 30-35 min
-Sprint. Vinnertid 12-15 minutter
-Stafett. Varierende antall etapper og etappelengde
-Nattorientering. Vinnertid i mesterskap ca. 70 minutter for herrer og 55-60 for damer
-Ultra-langdistanse. Vinnertid 140-150 min for herrer, 110-120 for damer.

I tillegg til løpstiden skiller øvelsene seg fra hverandre gjennom ulik karakter på terrengvalg og ikke minst utfordringene løypeleggeren gir løperne.

Langdistanser bør helst arrangeres i skikkelig real skog, der utøverne blir nøst til å ta seg frem tøft krevende terreng, gjerne kupert. Det internasjonale orienteringsforbundet, IOF har definert langdistansen som den fysisk utholdende øvelsen, der evnen til å ta ut gode veivalg og gjennomføre disse er viktigere enn nøyaktig orientering inn mot postene. Løypen kan godt inneholde noen mer tekniske partier for å teste løpernes evne til å skifte teknikk.

Mellomdistansen er den tekniske øvelsen. Finorientering og evnen til å holde konsentrasjonen er viktige elementer. Også mellomdistansen bør inneholde veivalgsstrekk, men disse bør ikke komme på bekostning av teknisk krevende orientering. Terrenget bør være detaljrikt, gjerne med kupering, mens helst ikke altfor god sikt.

Den siste av de internasjonale mesterskapsøvelsene, sprint, blir som regel arrangert i byer eller bynære områder. Kjennetegnet for sprinten er høy hastighet. I sprintøvelsen testes løpernes evne til, i stor fart, å oversette kartet til terrenget og ta ut veivalg. Spesielt i urbane sprintkonkurranser handler det mye om å avgjøre hvilken vei rundet en hindring som er den raskeste. Jo mer komplekst miljøet er, jo vanskeligere blir det å avgjøre hva som er raskest. Om det i tillegg kommer høydeforskjeller inn i bildet, kan det fort tapes noen avgjørende sekunder. [2]

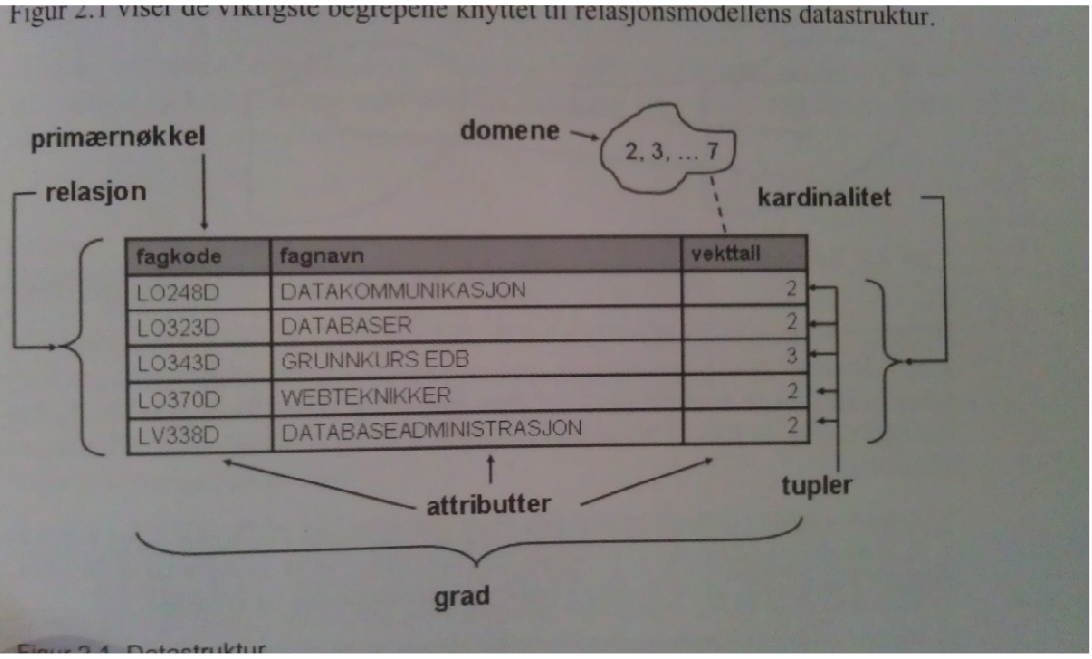
3. Programvare og programmering

3.1.Databaser

Databaser brukes for å håndtere store mengder informasjon. Relasjonsdatabaser er i dag den klart dominerende modellen. For å kunne kalle seg en relasjonsdatabase er det en del krav som må oppfylles. Edgar Codd har laget en liste med 12 punkter, hvorav to regnes som helt avgjørende.

- Databasen skal av brukeren oppfattes som tabeller og ikke noe annet
- Databasen skal ha operasjonene seleksjon, projeksjon og forening

Det er foreløpig ingen systemer som oppfyller alle de 12 punktene til Codd, men en god del systemer oppfyller de to nevnte krav [3].



Bilde 3.1. Bildet viser oppbyggingen av en databasetabell (fra Databaser).

Som bildet over viser består en databasetabell av et antall elementer. En rad i tabellen kalles en tuppel, mens en kolonne er et attributt. Antallet tupler gir tabellens kardinalitet, mens

antallet attributter bestemmer graden. En, eller en kombinasjon av, attributter må være entydige. Dersom man entydig kan identifisere en tuppel ved hjelp av et attributt, kalles denne primærnøkkel, alternativt kan man kombinere flere attributter til en sammensatt nøkkel. En enkel løsning på dette er å ha en egen kolonne, kun for eksklusivt å kunne identifisere tuppelen. De fleste databasesystemer har mulighet for å implementere en automatisk inkrementerende verdi, slik at hver tuppel får sin egen id.
Et annet viktig element i relasjonsdatabaser er fremmednøkkel. Fremmednøkkelen i en tabell er et attributt som peker til primærnøkkelen i en annen tabell og på den måten binder tabellene sammen.
For å kunne søke i databasens ulike innhold, er det nødvendig at hvert attributt tilordnes en datatype. Typiske datatyper er tekst/string, tall/int, dato og boolean (sant eller falskt).Dette muliggjør søk etter f.eks. alle som er eldre enn 32 år som heter Knut.
Databaser er ofte plassert på eksterne servere slik at flere kan utnytte dataene som ligger der gjennom spørringer. Dersom det kommer for mange spørringer for tett på hverandre blir det kø i databasen. Når man programmerer mot databaser er det derfor viktig å ta høyde for at det kan komme mange spørringer dersom programmet kjøres av flere klienter eller brukere samtidig.

3.2.Geografiske databaser

Databaser benyttes ikke lenger kun til lagring og behandling av enkle data. I dag benyttes databaser til å håndtere det meste, deriblant nettsider, bilder, musikk, avansert design og kart. For å kunne behandle mer kompliserte typer data enn det som kan representeres i en standard database, trenger man oftest en egen utvidelse til databasen. Flere av de store databaseleverandørene tilbyr slike utvidelser. IBM DB2, Oracle, Microsoft SQL server, MySQL og PostgreSQL er blant leverandørene av geografiske databasesystemer. For å behandle dataene i denne oppgaven har jeg benyttet PostgreSQL med utvidelsen PostGIS, så fokuserer naturlig nok på dette systemet.

PostgreSQL er et objektbasert relasjonsdatabasesystem basert på åpen kildekode. Systemet er gratis å laste ned fra www.postgresql.org. PostgreSQL kan benyttes på alle større operativsystemer, støtter de vanlige datatypene i tillegg til binære filtyper som bilder, lyd og video. PostgreSQL støtter også et antall programmeringsspråk, deriblant Java, Perl, Python, C, C++ og PHP, slik at det er mulig å programmere mot databasen.

3.3.PostGIS

Som nevnt behøves det ofte en utvidelse til databasesystemene for at de skal kunne håndtere geografiske datatyper. Utvidelsen til PostgreSQL heter PostGIS, og kan lastes ned fra http://postgis.refractions.net. PostGIS følger spesifikasjonene utarbeidet av Open Geospatial Consortium, OGC, for Simple Features SQL. Disse standardene er utarbeidet for å kunne utnytte databasers muligheter på internett. Mange har benyttet seg av denne muligheten til å gjøre data fra databaser tilgjengelig, deriblant Mapserver, en plattform bygget på åpen kildekode, for presentasjon av geografiske data og interaktive kart og OpenJUMP, som er et geografisk informasjonssystem (GIS) med PostGIS som database. [4]

Under er det listet noen av de geografiske datatypene som er støttet i PostGIS:
Point (X Y)
Linestring (X Y, X Y, X Y)

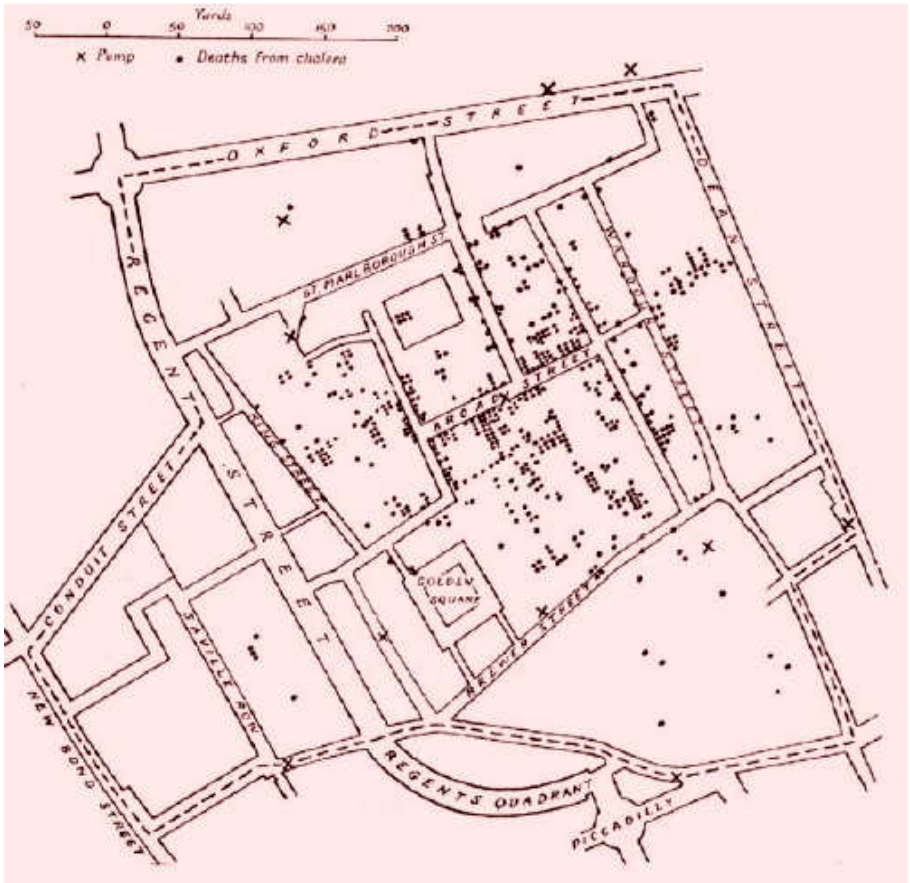
Polygon ((X Y, X Y, X Y, X Y), (X Y, X Y, X Y, X Y))
Multipoint (XY, X Y)
Multilinestring
Multipolygon

Lagring av data er en ting, men de virkelige fordelene med geografiske databaser ligger i spørringene man kan foreta. Blant funksjonene i PostGIS kan nevnes f.eks.:

- ST_distance- returnerer avstanden mellom to punkt
- ST_buffer- lager en buffer i gitt størrelse rundt geometrien
- ST_Within- Returnerer sann, dersom hele geometri A er inne i geometri B

For en mer fullstendig oversikt over funksjoner,
http://postgis.refractions.net/docs/reference.html#Spatial_Relationships_Measurements

3.4.GIS



Bilde 3.2. Et av de første GIS, fra 1854. Dr. John Snow plottet koleradødsfall og vannpumper i London på et kart, og så en klar konsentrasjon av dødsfall rundt en av vannpumpene ().

Geografiske informasjonssystemer, slik vi kjenner dem i dag, har kun eksistert i ca. 30 år. Frem til datamaskinenes gjennombrudd på 80-tallet foregikk det meste av GIS-analyse på papirkart. Dataene kunne organiseres på flere tynne ark eller plastfolier, slik at man ved å kombinere de ulike lagene enkelt kunne visualisere flere fenomener[6].

I dag er GIS et kraftig verktøy for å lagre, behandle og analysere geografiske data. I bunnen av et GIS ligger det som nevnt et geografisk databasesystem. Noen bygger på fritt tilgjengelige systemer, slik som OpenJUMP, mens for eksempel ArcInfo og ArcView (begge fra ESRI) først og fremst benytter seg av ESRIs eget system, ESRI *Spatial Database Engine* (SDE).

Om med papirversjonene av GIS behandles ofte dataene i lag. Fordelen med GIS på datamaskiner er at det er enkelt og raskt å slå av og på visning av lagene, i tillegg til at databasen som ligger i grunnen gjør det mulig å koble data sammen og utfør spørringer på disse. For eksempel kan man koble et lag med markslag sammen med et lag med informasjon om tomtegrenser for å finne ut hvem som eier jordbruksareal.

3.5.Java

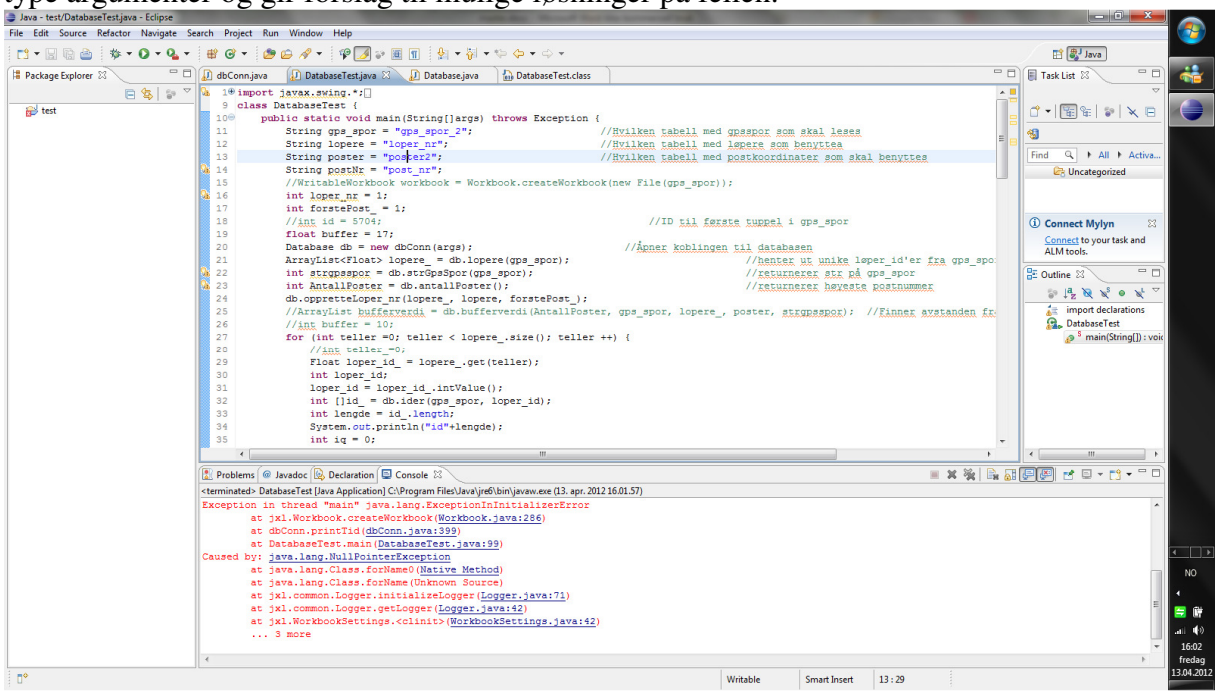
I denne oppgaven valgte jeg å benytte Java for å programmere mot PostGIS databasen. Java er et objektorientert dataprogrammeringsspråk. Et av kjennetegnene til objektorientert programmering er at man lager et antall enheter, eller moduler, som tar seg av hver sine deler av programmet, for deretter å sette disse sammen for å oppnå ønsket funksjonalitet.

Java er et stort og tungt programmeringsspråk. Det finnes et stort antall ferdige moduler, eller klasser som de også kalles. Disse er gjerne pakket i biblioteker som må importeres før de kan benyttes i programmeringen. Mange av disse klassene følger med programmet, men det er også mulig å programmere egne, eller benytte seg av klasser andre har laget og gjort tilgjengelig.For å kunne programmere mot PostGIS må man hente klassen JDBC (Java Database Connectivity) og importere denne i programmet.

Som i databaser er man avhengig av å angi hva slags datatype ulike variabler er. Tall, tekst og lister behandles ulikt og kan ikke blandes. I Java har man muligheten til å endre datatype gjennom såkalt *casting*[7].

3.6.Eclipse

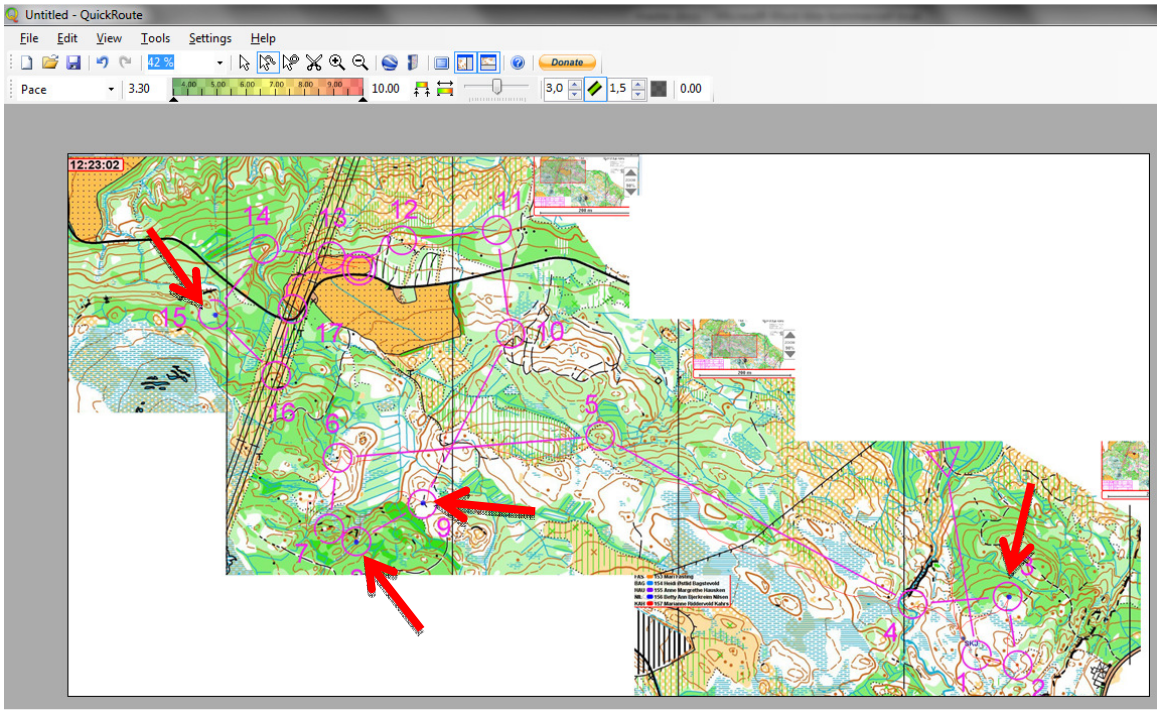
For å lette programmeringen finnes det utviklingsmiljøer med en del innebygde funksjoner og verktøy. Eclipse er et slikt verktøy utviklet for programmering i Java. Et slikt miljø gjør programmeringen mer oversiktlig, blant annet gjennom å gi reserverte ord egne fargekoder. Eclipse holder også oversikten over og sier ifra om enkelte metoder tar inn feil antall eller type argumenter og gir forslag til mulige løsninger på feilen.



Bilde 3.3. Skjermdump av Eclipse. Programmeringskoden i det øvre vinduet i midten, feilmeldinger i det nedre.

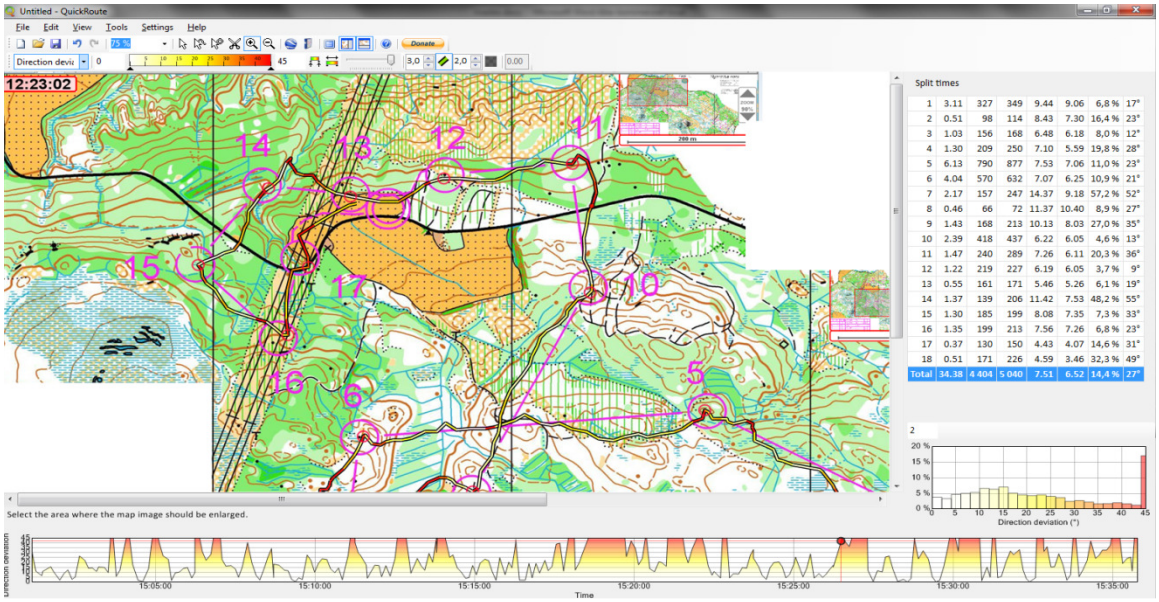
3.7.QuickRoute

QuickRoute (http://www.matstroeng.se/quickroute/en/) er et verktøy for analyse av GNSS-spor, utviklet av og for orienteringsløpere. GNSS data kobles til et kart over det aktuelle området gjennom å dra minst tre punkter på GNSS-sporet til rett sted på kartet.



Bilde 3.4. Skjermdump fra QuickRoute. Pilene peker til de fire punktene som ble brukt for å georeferere kartet.

Etter at kartet er koblet med GNSS dataene kan man legge inn mellomtider på postene for å se hvor mye lenger enn luftlinje man ar løpt, se på fargekoder på sporet for å se farten i ulike faser av løpet eller hvor stort avvik man har i retning mot neste mellomtid.



Bilde 3.5. Hvitt er rett mot neste mellomtid, rødt er stort avvik i vinkel. Tabellen til høyre gir informasjon om løpt distanse, avstand mellom postene, anvendt tid, hastighet og vinkel.

3.8.GNSS

3.8.1. Generelt

Det første satellittbaserte navigasjonssystemet, TRANSIT, var klar til bruk i 1967. Satellittnavigering ble i første rekke utviklet for militært bruk, men ble fort også populært for kommersielle og private aktører. I dag er det to systemer som er fullt operative, det amerikanske GPS og russiske GLONASS, mens Europa og Kina snart har sine systemer operative. Selv om mange benytter forkortelsen GPS om alt som har med satellittnavigasjon å gjøre, er den riktige betegnelsen GNSS (Global Navigation Satellite Systems), GPS er som nevnt den amerikanske versjonen av teknologien.

For å beregne posisjon ved hjelp av satellitter må mottakeren ha kontakt med minimum 3 satellitter. Gjennom å måle tiden det tar for signaler å gå fra satellitt til mottaker, kan man beregne mottakerens posisjon. Satellittene sender ut signal i form av en repeterende kode, og ved å måle hvor i koden mottakeren er på et gitt tidspunkt, kan man beregne tiden signalet har brukt fra satellitt til mottaker. For å oppnå høyere nøyaktighet på målingene begynte man å måle på selve bærebølgen til signalet. Bærebølgen er sinusformet og ved å måle hvor i svingningen man er får man meget nøyaktige målinger, < 1cm. Siden bærebølgen ikke er spesielt lang, er man nødt til å måle uavbrutt over lengre tid for å finne ut hvilken svingning man befinner seg i. GPS utstyret brukt for innsamling av data til denne oppgaven er ikke laget for slike nøyaktige målinger, og jeg går derfor ikke nærmere inn på hvordan man løser denne flertydigheten. Også andre faktorer spiller inn på nøyaktigheten til målinger utført med GNSS:

- Atmosfæriske forstyrrelser
- Feil i satellittklokke
- Feil i mottakerklokke
- Multipath

3.8.2.Atmosfæriske forstyrrelser

På sin ferd gjennom atmosfæren påvirkes signalets hastighet, avhengig av trykk, fuktighet og temperatur, noe som påvirker tiden signalet bruker fra satellitt til mottager. For å korrigere for disse forstyrrelsene kan man anta at atmosfærens påvirkning er tilnærmet lik innenfor et begrenset geografisk område, og plassere en ekstra mottager på et kjent punkt i nærheten av der man måler.

3.8.3.Klokkefeil

Selv om klokkene i GNSS er meget nøyaktige er det små feil her som påvirker resultatet. For å fjerne disse feilene fra beregningene benyttes flere mottakere og flere stellitter for å differensiere bort feilene. Ved å benytte relative avstander istedenfor absolutte, blir slike feilkilder eliminert.

3.8.4.Multipath

Multipath er den feilkilden som er mest relevant for denne oppgaven. Som navnet antyder oppstår feilen ved at signalet kan ta flere veier fra satellitt til mottaker. Signalet kan reflekteres av fysiske hindringer og dermed ta en lengre vei. Dette er særlig et problem i

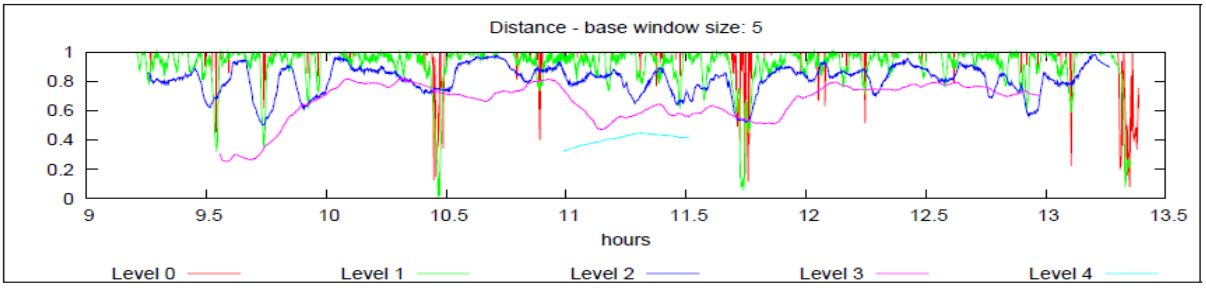
urbane strøk, der høye bygninger skjermer for fri sikt mellom satellitt og mottaker, men også i skogen kan signalet bli forstyrret av trær og fjellformasjoner. Problemet med multipath er en av grunnene til at jeg har valgt ikke å analysere GNSS spor fra sprintkonkurranser[8] og [9].

3.8.5.GNSS tracker

En GNSS-tracker består av, i tillegg til en GNSS-mottaker, en sender som kan videreformidle posisjonen. Trackerne som er benyttet til å samle inn kontrolldata i oppgaven sender posisjonen sin via mobilnettverket til en server, der de behandles og kombineres med et georeferert kart, for publikum til å følge med på. I og med at mobildekningen ikke alltid er god langt ute i skogen, er det fullt mulig at trackeren i perioder mister kontakten med serveren, noe som fører til at det ikke kommer noen nye oppdateringer på GNSS-mottakerens posisjon. Alle dataene er i ettertid tilgjengelig på leverandørens hjemmeside, http://www.gpsseuranta.net/, for gjennomsyn, analyse eller nedlastning.Det har ikke vært mulig å få tak i systemspesifikasjonen til GNSS-enheten i disse trackerne, men den benytter seg sannsynligvis av kodemålinger og har dermed en nøyaktighet på, i beste fall, 7-10 meter, avhengig av mottakerforhold. Enhetene har vært stilt inn på en oppdateringssekvens på 7 sekunder. Under innsamling av testdataene ble det benyttet utstyr med tilsvarende nøyaktighet, men med såkalt smartsampling, der enheten lagrer posisjonen når den føler det behøves, det vil si når man endrer retning eller hastighet [10].

3.9.Tidligere studier

Håvard Tveite har i en studie testet ulike metoder for å klassifisere bevegelse. Han har i sin studie sett på forskjellen i bevegelsesmønstre til en orienteringsløper og en ungdoms tilfeldige fysiske aktivitet. Flere ulike metoder ble testet, og alle viste at orienteringsløperen i stor grad beveget seg direkte fra A til B. Ved å se på 10 og 10 etterfølgende observasjoner, viste det seg at løperen kun sporadisk beveget seg mer enn 20 % (0,8 på figuren under) lenger enn avstanden fra første til siste observasjon [11].



Bilde 3.6. Viser avstand A-B / løpt distanse mellom A og B, med ulikt antall observasjoner mellom A og B, level 0 færrest, level 4 flest.[11]

4. Metode

4.1.Hva er bom:

En bom i orientering kan være så mangt. Det vanligste er nok å rote bort noen sekunder på de siste meterne inn mot posten. Andre varianter er såkalte parallellfeil, der løperen mener å ha full kontroll samtidig som han egentlig er på feil åsrygg eller tar av i et stikryss for tidlig. Andre varianter er å løpe 180 grader feil ut av en post eller overse en post på kartet og løpe rett til den neste. For en orienteringsløper er det sjelden vanskelig å vite at man har vært på bomtur, og i mange tilfeller merker man det også nesten med en gang. Å finne en absolutt metode for å sjekke om en løper er der han ønsker og tror derimot, er en utfordring. I motsetning til et annet bruksområde for GNSS, bilnavigasjon, er det ikke i orientering noen forhåndsdefinert rute mellom A og B, noe som gjør problemstillingen rundt det å avdekke avvik fra ønsket rute, siden denne kun eksisterer i løperens hode, atskillig mer innviklet. Under vil jegpresentere noen mulige metoder for automatisk å kunne avdekke om en løper er ute på bomtur, der jeg diskuterer fordeler, ulemper og problemer knyttet til de ulike fremgangsmåtene.

4.1.1.Tidstap

Ved å benytte såkalte strekktider, tiden mellom to poster, kan man få et varsel om at en løper bruker unormalt lang tid til en post, noe som ofte skyldes bomming. Med gode metoder for å avgjøre når løperen er på en post vil denne metoden være relativt enkel å implementere. F.eks. kan den, til enhver tid, beste strekktiden lagres i en variabel, og når en løpers tid på det aktuelle strekket overstiger denne med noen prosent, kan man anta at det er en bom. Prosentsatsen kan selvfølgelig varieres fra løper til løper, f.eks. basert på strekktider tidligere i løpet. Mats Troeng har en slik funksjon i sitt program for analyse av strekktider, WinSplits. Problemer med denne metoden oppstår dersom en løper oppnår en meget god tid på et strekk, slik at resten av startfeltet blir registrert med bom. Det finnes også andre svakheter ved en algoritme kun basert på tidsbruk. Som jeg vil diskutere senere i kapittelet er det ikke alltid like enkelt å avgjøre hvorvidt en løper har vært på posten, og dermed heller ikke så lett å vite tidspunktet for en postpassering. Enda en ulempe er at man ikke nødvendigvis vil fange opp bommen mens den foregår. Dersom avstanden mellom to poster er relativt stor, kan en løper gjøre en feil tidlig på strekket, men denne vil ikke fanges opp av systemet før etter at bestetiden på strekket er overskredet, i verste fall mange minutter etter at feilen ble gjort.

4.1.2.Forhåndsdefinerte ruter

Orienteringens egenart med fritt og ofte omgående veivalg mellom postene utelukker muligheten for å sjekke løperens posisjon opp mot forhåndsdefinerte ruter. En mulighet kan være å definere et område løperne bør være innenfor, eventuelt lage buffere rundt de mest aktuelle og antatt beste veivalgene. Denne metoden gir mye forarbeidde, siden det er mange strekk som må grundig analyseres for å avdekke alle sannsynlige veivalg og deretter få lagt inn rutene i databasen. Metoden vil dessuten ikke være spesielt god på å avdekke mindre bom nærme posten. Muligens egner denne seg bedre som et verktøy for veivalgsanalyse enn deteksjon av bom.

4.1.3.Avstand til posten

I de aller fleste tilfeller vil en situasjon der løperen beveger seg vekk fra neste post være ensbetydende med at han er på villspor. Likevel vil ikke dette alene være noen god indikasjon på en bom. Det er ikke umulig at den raskeste ruten mellom to poster innebærer at løperne i perioder øker avstanden mellom seg og posten de er på vei mot. Også mindre hindringer, som trevelt, tette busker og skrenter, kan tvinge løperne til å løpe omveier, som i sin tur innebærer

at avstanden til posten øker. Også unøyaktigheten i GNSS kan i visse tilfeller gi utslag. En algoritme som bygger på dette prinsippet må i det minste ta for seg mer enn den siste posisjonen fra trackeren for å avgjøre om det er snakk om en bom, eller bare en hindring som må passeres.

4.1.4.Liten eller ingen bevegelse

En vanlig reaksjon for en løper som har bommet, er at han stopper opp og forsøker å finne ut hvor han er. Ved å regne ut avstanden mellom etterfølgende posisjoner fra trackeren og tidspunktet for disse, kan man finne hastigheten til løperen. En løper som står stille, eller beveger seg meget langsomt er muligens ute og bommer. Det er selvfølgelig mulig at terrenget i området umuliggjør stor hastighet, så algoritmen kan med fordel forsterkes gjennom å se på hastigheten til andre løpere som har vært i det samme området. Iblant kan trackeren falle ut, noe man eventuelt må ta høyde for i implementeringen av metoden.

4.1.5.Retningsforandringer

En annen, ikke helt uvanlig reaksjon når man mister kontrollen er å løpe rundt uten noen god plan for å forsøke finne et sikkert holdepunkt. Det kan derfor være mulig å se på stadige retningsforandringer for å avdekke en bom. Retningsforandringer kan også skyldes såkalte mikroveivalg, og i en del typer terreng er det overhodet ikke mulig å unngå å løpe uten stadige retningsforandringer.

4.1.6.Løpt distanse i forhold til netto forflytting

Dette blir en variant av retningsforandring, men istedenfor å se på momentane endringer, velger man å se på de n antall siste observasjonene eller observasjonene over et gitt tidsrom. Deretter kan man finne en faktor for hvor konstant retning løperen har holdt, gjennom å se på summen av avstand mellom de n siste observasjonene i vinduet og avstanden mellom den første og siste observasjonen i vinduet. Når en ny observasjon blir tilgjengelig, dropper man den eldste, og får dermed et sliding window med konstant størrelse. Her kan størrelsen på vinduet være utslagsgivende for om en bom blir riktig klassifisert og hvor raskt den oppdages.

4.1.7.Samme område over lengre tideller flere besøk

Ofte skjer en bom nære posten, og løperen vet at han er i nærheten. Et søk etter posten vil derfor trolig foregå i et rimelig avgrenset område og løperen kan godt vende tilbake til et punkt han har vært tidligere for å gjøre et nytt forsøk. En metode som sjekker om løperen har vært innenfor et begrenset område over lengre tid, eller kommer tilbake til et område han allerede har besøkt, kan være en god indikasjon på bom.

4.2.Oppsummering av metodene

Av metodene nevnt ovenfor synes jeg metoden med sliding window skiller seg ut som den mest interessante å gå videre med.Metoden virker robust med tanke på å få med seg de fleste bommer, selv om den ikke vil gi utslag før løperen ikke lenger beveger seg i en rimelig stabil retning.

4.3.Godkjent kvittering

På grunn av flere forhold, som unøyaktighet i GNSS og oppdateringsfrekvens, vil det ikke være mulig kun å sammenligne løperens posisjon med postens koordinater for å avgjøre om posten er besøkt eller ikke. Ved å lage en buffer rundt posten kan man benytte en metode for å sjekke om en løper er i nærheten av posten. Hvor stor denne bufferen bør være krever nærmere undersøkelser. Bufferen bør være liten nok til at løperen med største sannsynlighet har vært på posten, samtidig som feilkilder og oppdateringsfrekvens ikke fører til at løpere ikke blir registrert når de har vært på posten. Tett skog og dårlig sikt vil ytterligere skjerpe kravene til størrelsen på bufferen. F.eks. kan multipathproblematikken slå inn i kuperte

terreng. I tillegg risikerer man at en løper passerer nærme posten, innenfor bufferen, uten å oppdage den og situasjonen dermed ikke bli fanget opp som en bom.
Det er også mulig at trackeren faller ut i kortere tidsrom, og dersom dette skjer i nærheten av en post, vil ikke systemet få med seg at løperen har vært innom og deretter er på vei mot neste post. En mulighet her er også å sjekke sporet mot den neste posten i løypen, for muligens luke ut falske positive utslag.

4.4.Programmet

Programsnutten som blir benyttet under analysen av GNSS dataene er skrevet i Java. I tillegg til standardbiblioteket benyttes også klassepakkene jxl.jar for å lese fra og skrive til Excel og aktuell variant av postgresql.jdbc.jar for å koble til og kommunisere med postGIS databasen. Dataene ligger lagret på en PostgreSQL server med PostGis-utvidelse. Alle dataene som blir brukt i denne oppgaven er opprinnelig i en eller annen form for XML, men siden de ulike leverandørene av GNSS-mottakerne har valgt noe forskjellig navn på merkelappene, har det vært enklest å åpne filene i Excel. Excel lager en kolonne for hver merkelapp, så det er enkelt å hente ut dataene derifra med klipp og lim.

Bok2

	A	B	C	D	E	F	G	H	I
1	version	creator	ns1:metadata	ns1:name	ns1:cmt	lat	lon	ns1:ele	ns1:time
2	1,1	SportTracks 2.1				38,37807846	-7,900689125	326,9855957	2011-03-10T10:54:57Z
3	1,1	SportTracks 2.1				38,37808609	-7,900650978	326,5048828	2011-03-10T10:55:00Z
4	1,1	SportTracks 2.1				38,37810516	-7,900588512	326,9855957	2011-03-10T10:55:04Z
5	1,1	SportTracks 2.1				38,37818909	-7,900490284	327,9467773	2011-03-10T10:55:09Z
6	1,1	SportTracks 2.1				38,37827682	-7,900314331	327,9467773	2011-03-10T10:55:15Z
7	1,1	SportTracks 2.1				38,37834167	-7,900146008	325,5435791	2011-03-10T10:55:20Z
8	1,1	SportTracks 2.1				38,37842941	-7,900041103	325,0629883	2011-03-10T10:55:24Z
9	1,1	SportTracks 2.1				38,3785553	-7,899980545	324,1015625	2011-03-10T10:55:28Z
10	1,1	SportTracks 2.1				38,37866974	-7,899865627	321,6983643	2011-03-10T10:55:32Z
11	1,1	SportTracks 2.1				38,37876892	-7,899764538	319,2950439	2011-03-10T10:55:36Z
12	1,1	SportTracks 2.1				38,37887192	-7,899628162	317,8530273	2011-03-10T10:55:41Z
13	1,1	SportTracks 2.1				38,37897873	-7,899544716	317,3724365	2011-03-10T10:55:45Z
14	1,1	SportTracks 2.1				38,37907791	-7,899524689	316,4110107	2011-03-10T10:55:49Z
15	1,1	SportTracks 2.1				38,37919235	-7,899503708	315,9304199	2011-03-10T10:55:53Z
16	1,1	SportTracks 2.1				38,37927246	-7,899458408	315,9304199	2011-03-10T10:55:56Z
17	1,1	SportTracks 2.1				38,37932968	-7,899218082	315,4498291	2011-03-10T10:56:02Z
18	1,1	SportTracks 2.1				38,37936401	-7,899053097	315,4498291	2011-03-10T10:56:06Z
19	1,1	SportTracks 2.1				38,37939453	-7,898963451	315,9304199	2011-03-10T10:56:09Z
20	1,1	SportTracks 2.1				38,37941742	-7,898841381	315,4498291	2011-03-10T10:56:13Z

Bilde 4.1. Eksempel på en .gpx fil åpnet i Excel. De aktuelle kolonnene er lat, lon og ns1:time.

Jeg har valgt å benytte tre tabeller i PostgreSQL for å gjøre analysene, hvorav to fylles med informasjon på forhånd, mens den siste henter dataene sine fra tabellen med alle observasjonene samt oppdateres av programmet etter hvert. Den største inneholder alle GNSS-dataene, lagret som "posisjon"av typen POINT og "tidspunkt" av typen Timestamp without timezone, samt løper-ID og ID serial (Integer).

Tabellen opprettes med kommandoen:
-Create table TABELLEN (id serial NOT NULL, loper_id integer, tidspunkt timestamp without time zone);
-Select AddGeometrycolumn('TABELLEN', 'posisjon', 4326, 'POINT', 2);

Select AddGeometrycolumn gjør tabellen mottakelig for stedlige data. SRID, i dette tilfellet 4326 forteller hvilket referansesystem som skal benyttes, POINT er type data som settes inn mens 2 viser til antallet dimensjoner i dataene.

Data settes inn med kommandoen:
INSERT INTO TABELLEN (loper_id, posisjon, tidspunkt) values(1, ST_GeomFromText ('Point(Longitude Latitude)', 4326),' 2011-03-10 10:25:47Z');

Loper_id forteller hvilken løper sine data som kommer. Posisjon og tidspunkt er koordinatene til løperen ved det gitte tidspunktet.
Den andre tabellen inneholder postnummer og koordinater for alle postene. Postkoordinatene fant jeg ved å studere trackingen i QuickRoute i kombinasjon med strekktidslisten for noen av løperne.

Tabellen ble opprettet og fylt med kommandoene:
-create table POSTER(post_nr int);
-select addGeometryColumn('POSTER', 'post_geom', 4326, 'point', 2);
-INSERT INTO poster (post_nr, post_geom) values(1, ST_GeomFromText ('Point(longitude latitude)',4326));

Den siste tabellen opprettes når programmet kjøres og inneholder alle løperne som finnes i tabellen med GNSS-sporene og hvilken post den enkelte er på vei mot.Mens de to andre tabellene har blitt fylt med geografisk informasjon før programmet starter å kjøre, vil den siste tabellen bli fylt og endret mens programmet går igjennom alle dataene i den første tabellen. Denne tredje tabellen inneholder heller ikke noe geografisk informasjon. For å være sikker på at det ikke ligger noe i tabellen fra før, starter jeg med å slette all info fra tabellen med kommandoen:
-Delete from"+lopere+";

Deretter hentes alle løperne som finnes i den første tabellen ut med kommandoen,
-select distinct loper_id from GPSspor

Og legges inn i tabellen lopere med kommandoen:
-"INSERT INTO" + lopere+"loper_id, pa_vei_til) values(loper_id, postnr)";

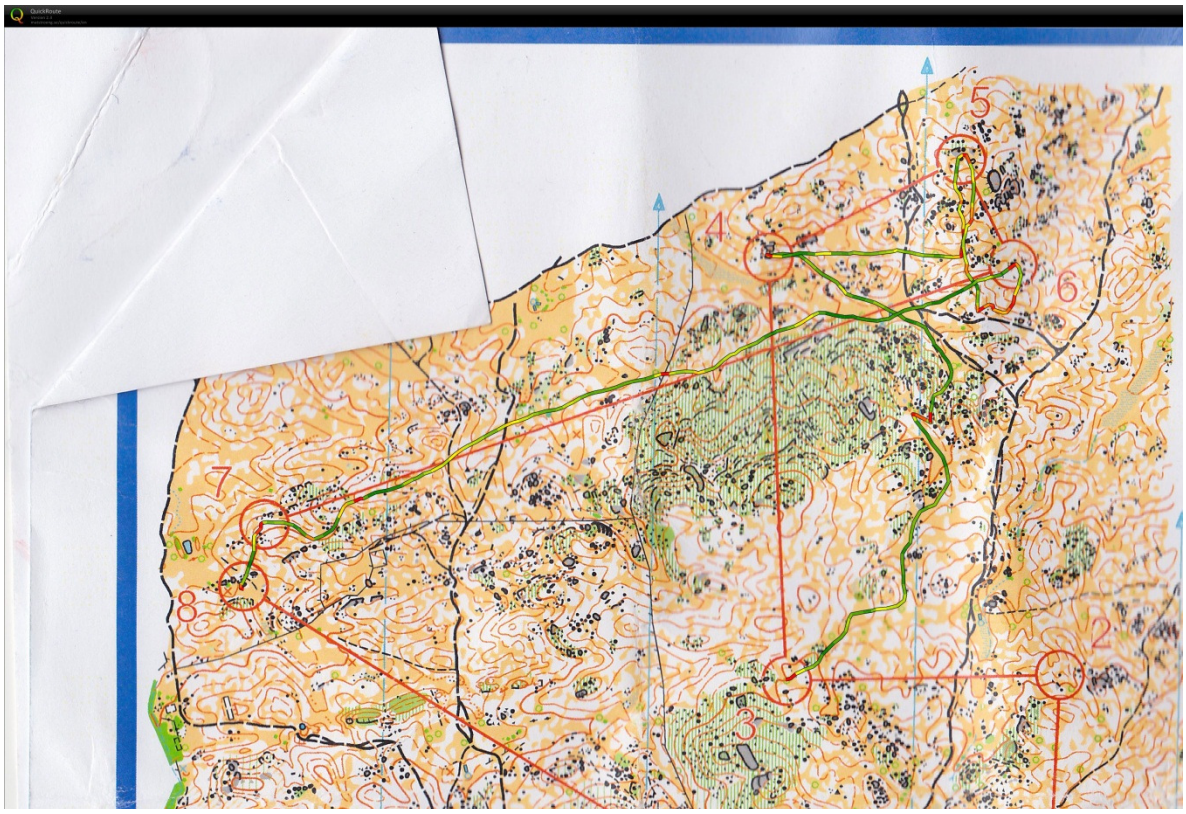
Etter at alle løperne er funnet og lagt inn i tabellen tar programmet for seg løperne en etter en. Først sjekkes det hvilken post løperen er på vei til, hvor langt det er igjen til posten og om denne avstanden er innenfor buffersonen rundt posten. Dersom det er tilfellet oppdateres tabellen med hvilken post løperen er på vei mot. Hvis løperen ikke er innenfor buffersonen beregnes det hvor langt løperen har løpt siden forrige observasjon. Programmet summerer opp de n antall siste observasjonene og dividerer med avstanden mellom den siste og den n-siste observasjonen. Denne faktoren vil ligge mellom 0 og 1, og sier noe om hvor mye lenger enn luftlinje løperen har løpt. Faktoren sjekkes deretter opp mot en satt grenseverdi for hva som kan regnes som bomfri oppførsel. For å unngå at meget små bommer gir utslag kreves det to etterfølgende verdier under grensen for å gi utslag. Til slutt skriver programmet alle resultatene til en .xls fil for hver løper, til bruk i videre analyser. Denne inneholder alle de beregnede faktorene med fire ulike vindusstørrelser, henholdsvis 5, 10, 20 og 30 observasjoner. I tillegg skrives det også en fil der det kun skrives ut de resultatene som anses som bom. For at verdien skal skrives til det andre regnearket må det være to påfølgende verdier som er lavere enn grenseverdien.

Jeg har også laget en metode for å finne den minste mulige verdien for buffer. Denne metoden finner den korteste avstanden for alle løpere til alle poster, og tar vare på den høyeste verdien. Siden dette ikke vil være mulig i en sanntidsanalyse der dataene kommer inn underveis, har jeg valgt å benytte en buffer på 17 meter.

5. Analyse

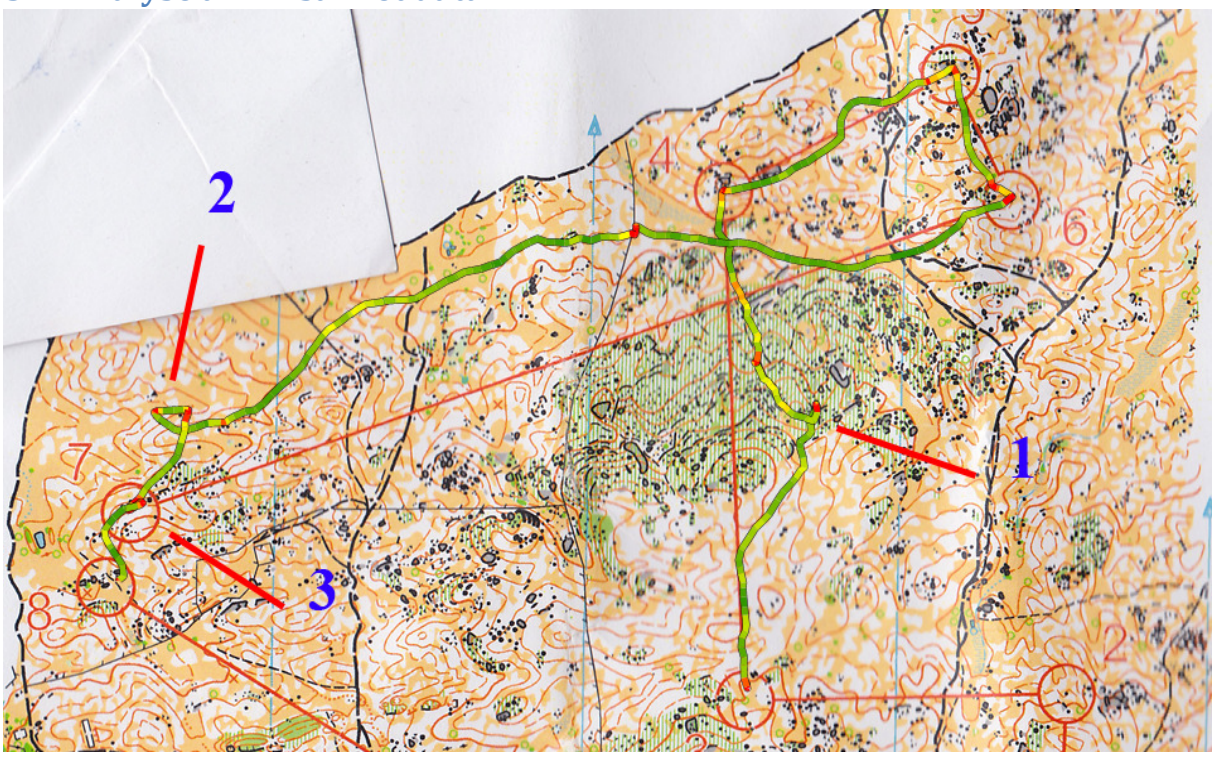
5.1.Datainnsamling

GNSS-dataene brukt til testing av metodene ble samlet inn spesielt med tanke på de utfordringene som ventet. En kortere orienteringsløype, som inneholdt elementer med flere av utfordringene diskutert tidligere, ble løpt fem ganger. Tre av rundene ble løpt med tanke på å skape et bilde av hva som kjennetegner GNSS-sporene til løpere som ikke bommer. I de to siste rundene ble det bevisst lagt inn ulike former for bom for å få testet metoden for deteksjon og finne en grenseverdi for hva som må kunne regnes som bom. Grenseverdien for dette området ble bestemt utfra samtlige 5 runder.

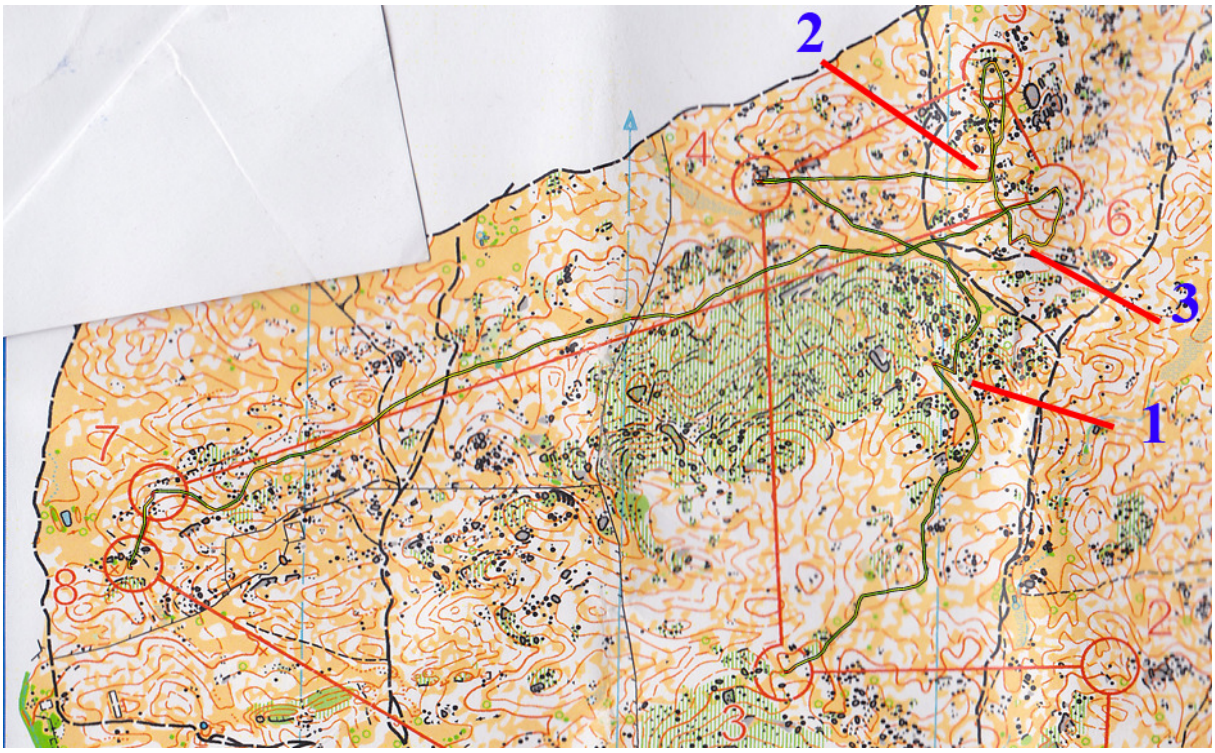


Bilde 5.1. En av gjennomløpningene av testløypen.

5.2. Analyse av innsamlet data



Bilde 5.2. Første gjennomløping av testløypen med 3 bommer som bør detekteres.

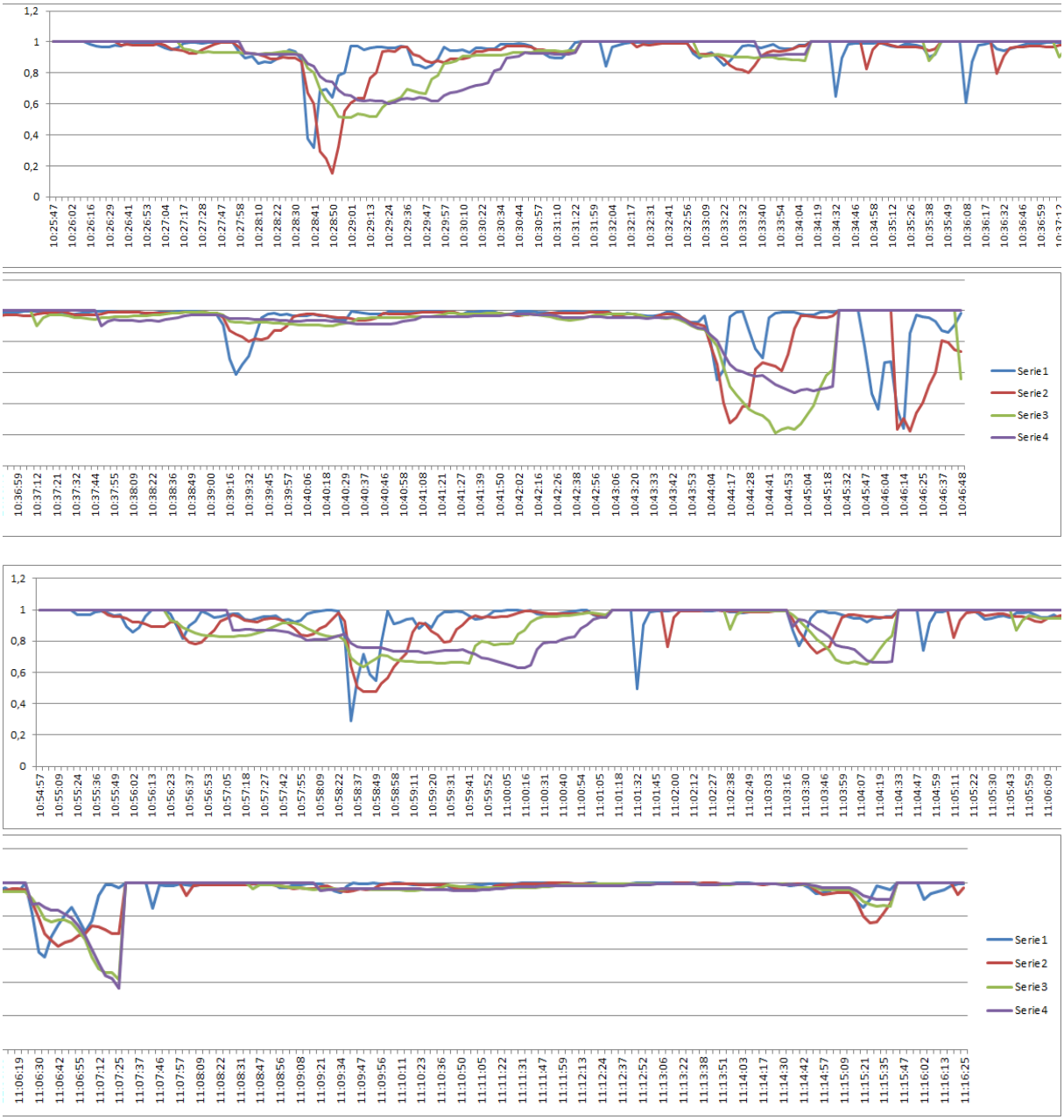


Bilde 5.3. Andre gjennomløping av testløypen med 3 bommer som bør detekteres.

Tidspunktene for bommene er:

Runde 1
11:28
11:44
11:46

Runde 2
11:58
12:03
12:06



Bilde 5.2. Grafene viser de to gjennomløpningene med innlagte bommer. Blå graf har et vindu på 5, rød 10, grønn 20 og fiolett 30 observasjoner.

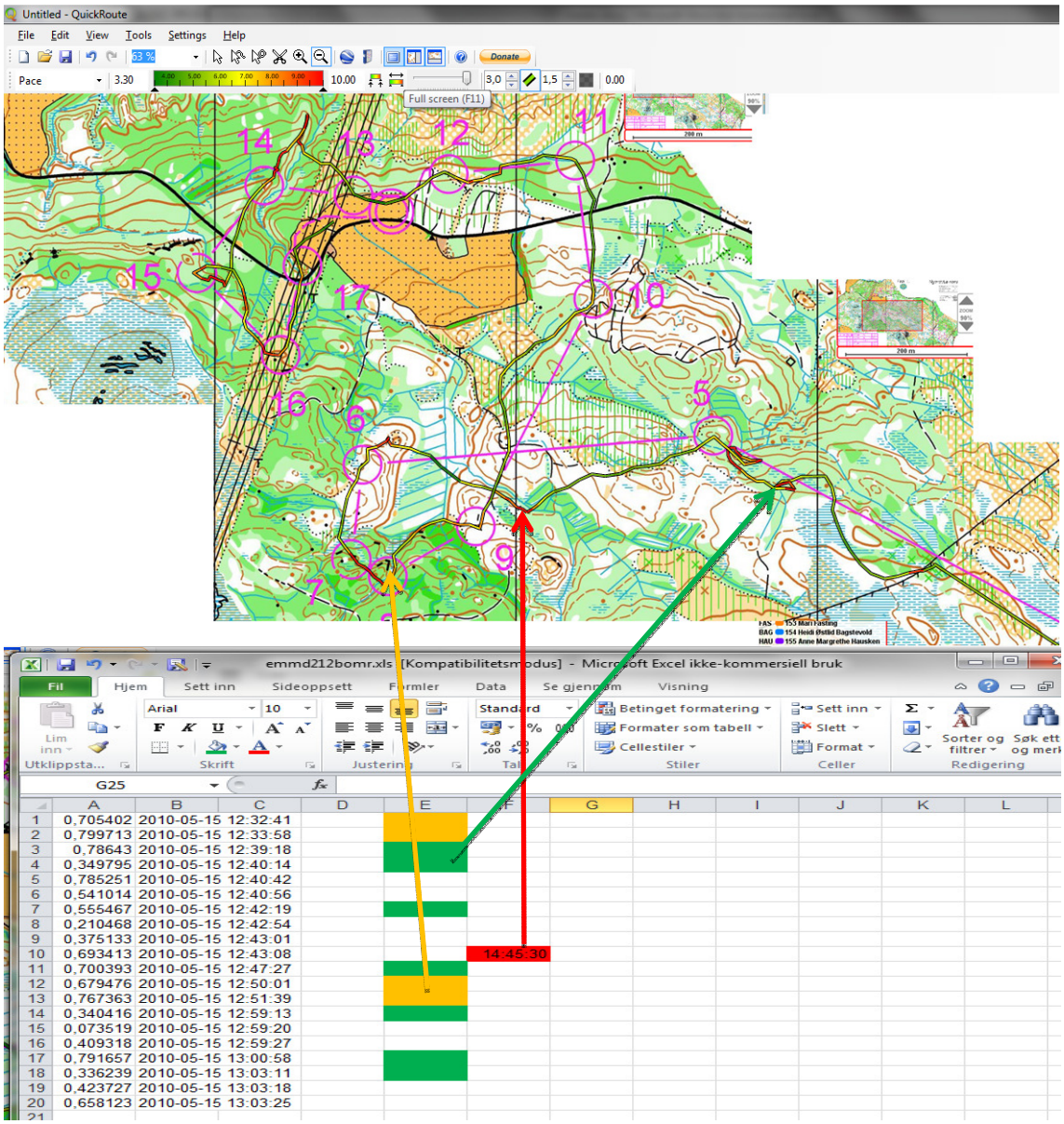
Grafene og tabellene viser at jo større vinduet man analyserer er, jo lenger tid tar det før en bom blir oppdaget. Med vindu som dekker de 20 og 30 siste observasjoner, forsvinner til og med hele strekk fra resultatene. De to siste (5 og 10 siste observasjoner) virker å reagere omtrent like kjapt og får med seg alle bommene. I tillegg til bommene kan man også se ytterligere et antall utslag. Ett skyldes kryssing av et gjerde der løperen har måttet løpe langs gjerdet(90 grader på løpsretningen) for å finne et passe passeringspunkt, mens de andre er på vei ut av enkelte poster. Det sistnevnte skyldes at bufferen for posten er satt litt for stor, slik at løperen registreres med godkjent passering av posten for tidlig og dermed får man en kortere periode der vinduet dekker både inn- og utløping av posten. Med spiss vinkel mellom postene vil naturlig nok dette slå ut på faktoren.

Bommene der løperen virkelig er ute og leter kan man se at faktoren faller helt ned mot 0,2, men for å fange opp den andre bommen på den andre gjennomløpningen må grenseverdien settes nærmere 0,8. En grenseverdi på 0,8 gir ingen falske positive utslag på vinduet med 5 observasjoner, mens vinduet som tar for seg de 10 siste observasjoner får to utslag på den andre gjennomløpningen. Et rett etter start og et rett før posten merket på kartet som nummer 7. Jeg velger derfor å benytte en grenseverdi på 0,8 i kombinasjon med et vindu på fem observasjoner.

6. Resultater

Etter å ha fastsatt grenseverdien for bomfri adferd testet jeg denne på GNSS data fra et orienteringsløp i Norge. Dataene har jeg fått hjelp av Jan Kocbach til å hente fra GPSSeuranta (leverandøren av GPS tracking ved de fleste orienteringsløp) sin nettside. Totalt ble algoritmen testet på 37 løpere fordelt på to løyper, 16 damer og 21 herrer.

Excelarkene med varslete bommer ble deretter kontrollert visuelt mot GNSS-sporet i QuickRoute.



Bilde 6.1. GNSS sporet til en av løperne, visualisert i QuickRoute og tilhørende excel arket. Grønn er bommer som er oppdaget, oransje er falske positive, dvs. de er feilaktig klassifisert som bom, mens rødt er bom som ikke har blitt fanget opp av algoritmen. De røde er fylt inn manuelt i ettertid. En del utsalg har ikke fått noen fargekode, som skyldes at de er direkte knyttet til en tidligere hendelse.

	Ant bom varslet	Ant bom riktig varslet	Ant falske varsler	Ant bom som ikke ble varslet
Damer	123	45	78	9
Herrer	53	35	18	8
Totalt	176	80	96	17

Tabell 6.1. Tabellen viser resultatet av testen.

Totalt fant testen altså 176 tilfeller der grenseverdien på 0,8 ble underskredet. 96 av disse viste seg likevel ikke å være bom, mens 17 tilfeller jeg oppfatter som bom, ga av ulike årsaker ikke stort nok utslag til å bli klassifisert som bom. Av de 96 falske utslagene var 93 rett etter godkjent kvittering på en post.

Jeg valgte også å finne gjennomsnittsfaktoren for testområdet og kontrollområdet, for å se om det var noen vesentlig forskjell i hvor rett linje løperne holder i ulikt terreng. Gjennomsnittsfaktoren for testfeltet var 0,968, mens den for kontrolldataene var på 0,930.

Gjennomkjøringen av de to klassene tok henholdsvis 54 min for damene og 58 min for herrene. På den tiden gikk programmet gjennom 5512 og 5760 observasjoner, noe som gir et tidsforbruk på ca. 0,6 sekunder per observasjon.

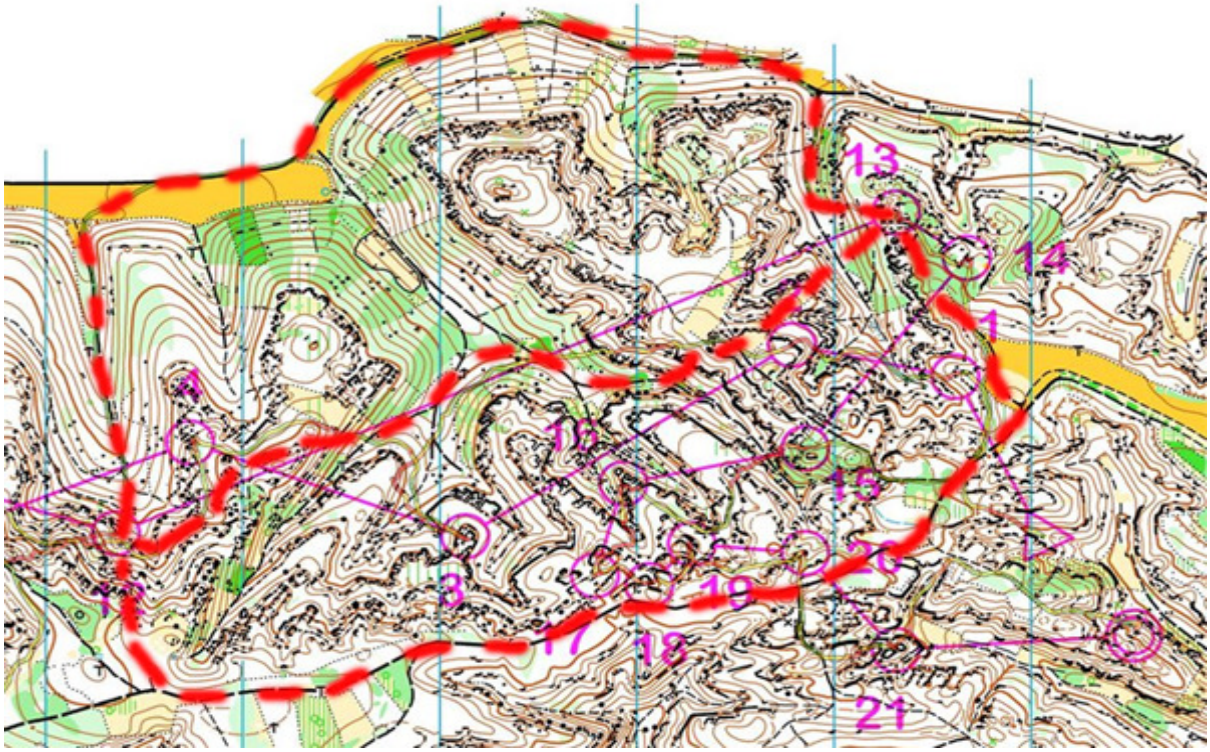
Jeg hadde planlagt å teste data fra flere løp og ulike terrengtyper, deriblant sprint i bymiljø, men det har oppstått en feil i programmet jeg ikke har klart å finne ut av.

7. Diskusjon

Til tross for at det er en tydelig forskjell på gjennomsnittsfaktoren i testterrenget og kontrollterrenget, fungerte det brukbart med samme grenseverdi for de to områdene. Av 97 faktiske bom ble 80 oppdaget, noe som gir en prosent på 82,5. Jeg fant tre årsaker til at bommene ikke ble riktig klassifisert; betingelsen om minst to etterfølgende verdier under 0,8, faktoren har ligget på så vidt over 0,8 og bommen har ikke blitt oppdaget i det hele tatt fordi løperen har justert seg inn over lengre tid.

Den store mengden falske positive utslag skyldes i all hovedsak at løperen har blitt registrert på posten for tidlig, med andre ord har bufferen vært for stor, slik inngangen til posten blir med i beregningen i starten av neste strekk. Dersom vinkelen mellom postene er spiss, er dette nok til å få en faktor på mindre enn 0,8, med påfølgende utslag i testen. Dersom man ser bort ifra disse nevnte falske positive var 96,4 % av utslagene korrekte. For å kvitte seg med det store antallet falske positive utslag må man finne en bedre måte for å registrere at en løper har vært på posten. En mulighet er å vente med å godkjenne kvittering på posten til løperen både er innenfor den satte bufferen og avstanden til posten øker. Det bør også tas hensyn til at GNSS-trackeren til tider kan miste kontakten med serveren, eller ha for dårlig GNSS-signaler til at denne får en observasjon innenfor bufferen, slik at løperen kan komme inn igjen i systemet, f.eks. ved å sjekke om løperen har vært innom bufferen til en senere post dersom avstanden til neste post øker over lengre tid.

Som nevnt var det en liten, men tydelig, forskjell i gjennomsnittsverdien for den beregnede faktor i testterrenget og kontrollterrenget. I begge løypene var det stort sett rett på som var beste veivalg, så forskjellen skyldes nok i stor grad at det var lite undervegetasjon og vesentlig mer åpen skog i terrenget testdataene ble samlet inn. I en helt annen terrengtype der løperne blir tvunget til å følge trangere passasjer og blir styrt mer av terrenget er det ikke sikkert metoden vil fungere like bra, eventuelt må grenseverdien justeres.



Bilde 7.1. Eksempel på terreng der løperne blir tvunget til å løpe med mye retningsforandringer.

Bilde 7.2. Sprint arrangeres ofte i urbane områder. Med mage korte strekk og mye retningsforandring også underveis på strekkene, egner nok den testede metoden seg dårlig til denne disiplinen.

En annen bekymring er tidsbruken til analysen. Metoden er på ingen måte optimalisert og gjør unødvendig mange beregninger. Med en kjøretid som den jeg hadde under testing, forutsatt at trackerne har en oppdateringsfrekvens på 7 sekunder, vil det ikke være mulig å håndtere mer enn 7/0,6=11 løpere ad gangen. Under testkjøringen har jeg sittet med mobilt bredbånd, så det kan antas at det største tidsforbruket er ved spørringer mot serveren. For å unngå problemer med etterslep i beregningene vil det derfor være anbefalt å la et sliktprogram kjøre sentralt på en server nærmest mulig databasen.

Selv om algoritmen viste seg å fungere tilfredsstillende, er det store rom for forbedringer, ikke minst med tanke på å takle andre typer terreng.Algoritmen jeg har testet er best egnet til å oppdage bom der løperen har oppdaget feilen og endrer retning, enten for å ta posten eller finne ut hvor man er. For å oppdage bom før løperen selv vet at det bærer galt avgårde, kan man sjekke om andre løpere tidligere har vært i det samme området. Dette kan også gjøres ved å forhåndsdefinere ruter eller soner det vil være naturlig å holde seg innenfor.

8. Konklusjon

Med de siste års teknologiske utvikling har det åpnet seg mange muligheter for orienteringssporten til å nå ut til publikum med nyvinninger, blant annet muligheten til å følge løperne hele veien, også ute i skogen. Som med så mye annet når man et punkt der mengden informasjon blir for stor og man må velge hva som er mest interessant. Det mest interessante i orientering er å få med seg hvor i løpet det blir avgjort, når skiller de beste seg fra de nest beste. Derfor har jeg forsøkt å finne en metode for å oppdage endringer i bevegelsesmønsteret til løperen og automatisk oppdage oppførsel som skiller seg fra det normale for en orienteringsløper, og klassifisere dette som en bom. Resultatene viser at orienteringsløpere, så lenge de ikke bommer, i stor grad holder en tilnærmet rett linje over kortere tidsrom. Dersom løperen, i de testede terrengtypene, løper mer enn 20 % lenger enn luftlinje over en periode på cirka 30 sek, kan dette i de aller fleste tilfeller anses som en bom som gir tidstap.

Selv om metoden ga gode resultater for de to terrengtypene den ble testet på, vil det nok være nødvendig med en del forbedringer for å få en universell metode for å klassifisere bevegelsesmønsteret til orienteringsløpere som bom eller ikke, uavhengig av terrengtype. Først og fremst må det utvikles en bedre algoritme for å avgjøre hvorvidt løperen har vært på posten, i tillegg til metoder for å skille bom fra ikke-bom i terrengtyper som tvinger løperen til et bevegelsesmønster som, ifølge denne oppgaven, klassifiseres som bom.

Referanser

[1]http://no.wikipedia.org/wiki/Orientering, 25.4.2011
[2]Løypeleggerboka, Bernt O Myrvold, 2007, Akilles – idrettsforlaget
[3] Databaser, Kjell Toft Hansen & Tore Mallaug, 2003, Stiftelsen TISIP & Gyldendal Akademisk
[4] http://postgis.refractions.net/documentation/
[5] Geographic Information Analysis, David O'Sullivan & David J. Urwin, 2003, John Wiley & sons, INC

[6] http://www.gis.unbc.ca/courses/geog300/lectures/lect17/index.php
[7] Programmering i Java, Else Lervik og Vegard B. Havdal, 2001, Gyldendal Akademisk
[8] GPS Satellite Surveying, Alfred Leick, 2004, John Wiley & Sons
[9] http://no.wikipedia.org/wiki/NAVSTAR_Global_Positioning_System
[10] http://static.garmincdn.com/pumac/Forerunner_910XT_OM_NO.pdf
[11] Investigations into Simple Statistics forClassifying Motion from Space-TimeTrajectories ofGeographical Point Objects, HåvardTveite, Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences

Vedlegg 1.

De tre javafilene somer brukt I analysedelen. I DatabaseTest.java defineres inputvariablene, samt statiske verdier om buffer. Filen inneholder også *main* som er den styrende metoden i java, som kaller opp andre metoder for å behandle inndataene. Database.java definerer de ulike metodene, hvilke variabler og datatyper de forskjellige metodene forventer og hva de skal returnere. db_Conn.java inneholder alle metodene som bearbeider inndataene.

DatabaseTest.java

```
importjavax.swing.*;
import java.io.*;
importjava.sql.ResultSet;
importjava.util.*;

importjxl.*;
importjxl.write.*;
importjxl.write.Number;
classDatabaseTest {
	publicstaticvoid main(String[]args) throws Exception {
		String gps_spor = "gps_spor_2";
//Hvilken tabell med gpsspor som skal leses
		String lopere = "loper_nr";
//Hvilken tabell med løpere som benyttea
		String poster = "poster2";
//Hvilken tabell med postkoordinater som skal benyttes
		String postNr = "post_nr";
		intloper_nr = 1;
		intforstePost_ = 1;
		float buffer = 17;
		Database db = newdbConn(args);
//Åpner koblingen til databasen
		ArrayList<Float> lopere_ = db.lopere(gps_spor);
//henter ut unike løper_id'er fra gps_spor databasen.
		intstrgpsspor = db.strGpsSpor(gps_spor);
//returnerer str på gps_spor
		intAntallPoster = db.antallPoster();
//returnerer høyeste postnummer
		db.oppretteLoper_nr(lopere_, lopere, forstePost_);

		//ArrayList bufferverdi = db.bufferverdi(AntallPoster, gps_spor, lopere_,
		poster, strgpsspor);

//Finner avstanden fra observasjonen nærmest postene
		for (int teller =0; teller < lopere_.size(); teller ++) {

			Float loper_id_ = lopere_.get(teller);
			int loper_id;
			loper_id = loper_id_.intValue();
			int []id_ = db.ider(gps_spor, loper_id);
			int lengde = id_.length;
			System.out.println("id"+lengde);
			intiq = 0;
			ArrayList<Float>avstandTilNestePost = newArrayList<Float>();
			ArrayList<Float> avstand_ = newArrayList<Float>();
	//lager liste med avstand mellom alle GPS observasjonene
			ArrayList<Float> avstand5 = newArrayList<Float>();
		//lager liste med avstand mellom GPS observasjoner med mellomrom = 10
			ArrayList<Float> avstand10 = newArrayList<Float>();
		//lager liste med avstand mellom GPS observasjoner med mellomrom = 10
			ArrayList<Float> avstand20 = newArrayList<Float>();
		//lager liste med avstand mellom GPS observasjoner med mellomrom = 20
			ArrayList<Float> avstand30 = newArrayList<Float>();
		//lager liste med avstand mellom GPS observasjoner med mellomrom = 20
			ArrayList<Float> Faktor5 = newArrayList<Float>();
		//Finner faktor for løpt distanse/avstand A-B (10)
			ArrayList<Float> Faktor10 = newArrayList<Float>();
		//Finner faktor for løpt distanse/avstand A-B (10)
			ArrayList<Float> Faktor20 = newArrayList<Float>();
		//Finner faktor for løpt distanse/avstand A-B (10)
			ArrayList<Float> Faktor30 = newArrayList<Float>();
		//Finner faktor for løpt distanse/avstand A-B (10)
```

```
                ArrayList<String> Tid = newArrayList<String>();
                ArrayList<Float> hopp_ = newArrayList<Float>();
                hopp_.add((float) 5);
                hopp_.add((float) 10);
                hopp_.add((float) 20);
                hopp_.add((float) 30);
                for (int teller2 =0; teller2 < lengde; teller2++)   {
                        intpost_nr = db.nestePost(lopere, loper_id);
        //Finner hvilken post løperen skal til neste gang
                        System.out.println(loper_id);
                        int id = id_[iq];
                        Float avstanden = db.avstandNestePost(post_nr, gps_spor, poster,
                        id);
                //Finner avstanden fra siste observasjon til neste post
                        avstandTilNestePost.add(avstanden);
                        System.out.println("postnummer "+post_nr);
                        intbesokt_ = db.besokt(avstandTilNestePost, buffer, post_nr,
                        loper_id);
                //Sjekker om nestePost er besøkt og oppdaterer løpertabellen med ny nestepost
                        if (besokt_ != 0) {
                                avstand_.clear();
                                avstand5.clear();
                                avstand10.clear();
                                avstand20.clear();
                                avstand30.clear();
                        }
                        else {
                                Float avstanden_ = db.avstand(gps_spor, id);
                //lager liste med avstand mellom alle GPS observasjonene

                        avstand_.add(avstanden_);
                        Float avstandHopp5 = db.avstandHopp(5, gps_spor, id);
                //lager liste med avstand mellom GPS observasjoner med mellomrom = 5
                        avstand5.add(avstandHopp5);
                        Float avstandHopp10 = db.avstandHopp(10, gps_spor, id);
                //lager liste med avstand mellom GPS observasjoner med mellomrom = 10
                        avstand10.add(avstandHopp10);
                        Float avstandHopp20 = db.avstandHopp(20, gps_spor, id);
                //lager liste med avstand mellom GPS observasjoner med mellomrom = 20
                        avstand20.add(avstandHopp20);
                        Float avstandHopp30 = db.avstandHopp(30, gps_spor, id);
                //lager liste med avstand mellom GPS observasjoner med mellomrom = 10
                        avstand30.add(avstandHopp30);
                        Float Faktor5_ = db.faktor(avstand_, avstand5, 5);
                //Finner faktor for løpt distanse/avstand A-B (10)
                                Faktor5.add(Faktor5_);
                                Float Faktor10_ = db.faktor(avstand_, avstand10, 10);
                        //Finner faktor for løpt distanse/avstand A-B (10)
                                Faktor10.add(Faktor10_);
                                Float Faktor20_ = db.faktor(avstand_, avstand20, 20);
                        //Finner faktor for løpt distanse/avstand A-B (10)
                                Faktor20.add(Faktor20_);
                                Float Faktor30_ = db.faktor(avstand_, avstand30, 30);
                        //Finner faktor for løpt distanse/avstand A-B (10)
                                Faktor30.add(Faktor30_);
                                String tid_ = db.Tid(id, gps_spor);
                                Tid.add(tid_);
                                iq=iq+1;
                        }
                db.print(Faktor5, gps_spor, 5);
                db.print(Faktor10, gps_spor, 10);
                db.print(Faktor20, gps_spor, 20);
                db.print(Faktor30, gps_spor, 30);
                    db.printTid(Faktor5, gps_spor, loper_id, Tid, Faktor10, Faktor20,
                    Faktor30, hopp_);
                db.printTid_ (Faktor5, gps_spor, loper_id, Tid, hopp_);

                }
        }
        db.KobleNedForbindelsen();
    //siste som skal gjøres
    }
}
```

Database.java

```
importjava.rmi.*;
importjava.util.*;
importjxl.*;
importjxl.write.*;
importjxl.write.Number;interface Database extends Remote {
	intantallPoster () throws Exception;
	intnestePost (String loper_nr, int loper_id) throws Exception;
	intstrGpsSpor (String gps_spor) throws Exception;
	int [] ider(String gps_spor, int loper_id) throws Exception;
	intbesokt(ArrayList<Float>avstandTilNestePost, float buffer, int post_nr,
intloper_nr)throws Exception;
	ArrayList<Float>bufferverdi (intAntallPoster, String gps_spor, ArrayList<Float>lopere_,
String poster, intstrGpsSpor) throws Exception;
	Float avstandNestePost (int post_nr, String gps_spor, String poster, int id) throws
Exception;
	ArrayList<Float>lopere(String gps_spor) throws Exception;
	Float avstand(String gps_spor, int id) throws Exception;
	Float avstandHopp(inthopp, String gps_spor, int id) throws Exception;
	String Tid(int id, String gps_spor) throws Exception;
	voidoppretteLoper_nr(ArrayList<Float>lopere_, String lopere, intforstePost)throws
Exception;
	Float faktor(ArrayList<Float>avstand_, ArrayList<Float>faktor_, inthopp) throws
Exception;
	void print (ArrayList<Float> Faktor10, String gps_spor, inthopp) throws Exception;
	voidprintTid (ArrayList<Float> Faktor5, String gps_spor, int loper_id,
ArrayList<String>Tid, ArrayList<Float> Faktor10, ArrayList<Float> Faktor20, ArrayList<Float>
Faktor30, ArrayList<Float>hopp_) throws Exception;
	voidprintTid_ (ArrayList<Float> Faktor5, String gps_spor, int loper_id,
ArrayList<String>Tid, ArrayList<Float>hopp_) throws Exception;
	voidKobleNedForbindelsen() throws Exception;
	voidexprint (ArrayList<Float> Faktor10, String gps_spor, int j) throws Exception;
}
```

db_Conn.java

```
importjava.sql.*;
importjava.util.*;
importjava.rmi.*;
importjava.rmi.ServerError.*;
import java.io.*;
importjava.lang.*;
importjxl.write.Number;
importjxl.write.WritableSheet;
importjxl.write.WritableWorkbook;
importjxl.*;
importjxl.write.*;

classdbConn implements Database {
	private Connection conn;
	private Statement setning;

	// Koble til databasen

	publicdbConn(String argv[]) throws ClassNotFoundException, SQLException {
		Class.forName("org.postgresql.Driver"); // laster inn JDBC drivere
		Connection conn = DriverManager.getConnection(
				"jdbc:postgresql://SERVER", "BRUKERNAVN",
				"PASSORD"); // Kobler opp til databasen
		DatabaseMetaDatadbmd = conn.getMetaData(); // Henter MetaData for å

	// sjekke om koblingen er

	// vellykket
		System.out.println("Connection to " + dbmd.getDatabaseProductName()
				+ " " + dbmd.getDatabaseProductVersion() + " successful.\n");
		setning = conn.createStatement();
	}

	// sletter gammelt innhold i loper_nr og legger inn nye løpere

	public voidoppretteLoper_nr(ArrayList<Float> lopere_, String lopere,
			int forstePost) throwsException {
		String sqlsetning = "delete from " + lopere + ";";
		setning.executeUpdate(sqlsetning);
		int j = 0;
		for (int i = 0; i < lopere_.size(); i++) {
			String sqlsetn = "INSERT INTO " + lopere
					+ " (loper_id, pa_vei_til) values(" + lopere_.get(j) + ","
					+ forstePost + ");";
			setning.executeUpdate(sqlsetn);
			j = j + 1;
		}
	}

	// Henter ut unike løper_ID fra aktuell gps spor tabell

	publicArrayList<Float>lopere(String gps_spor) throws Exception {
		ArrayList<Float>lopere_ = new ArrayList<Float>();
		String sqlsetning = "Select distinct loper_id from " + gps_spor + ";";
		ResultSet res = null;
		try {
			res = setning.executeQuery(sqlsetning);
			while (res.next()) {
				Float loperId = res.getFloat("loper_id");
				lopere_.add(loperId);
			}
		} finally {
			if (res != null)
				res.close();
		}
		return lopere_;
	}

	// henter ut ant tupler i gps_spor

	public int strGpsSpor(String gps_spor) throws Exception {
		intstrGps_spor = 0;
		String sqlsetning = "select id from " + gps_spor + ";";
```

```
		ArrayList<Float>antGpsSpor = new ArrayList<Float>();
		ResultSet res = null;
		try {
			res = setning.executeQuery(sqlsetning);
			while (res.next()) {
				Float antallet = res.getFloat("id");
				antGpsSpor.add(antallet);
			}
		} finally {
			if (res != null)
				res.close();
		}
		strGps_spor = antGpsSpor.size();
		returnstrGps_spor;
}

// Henter ut alle IDer som er i bruk.
// LEGGE TIL BETINGELSE MED LØPER ID
public int[] ider(String gps_spor, int loper_id) throws Exception {
		ArrayList<Float>idFloat = new ArrayList<Float>();
		String sqlsetning = "Select id from " + gps_spor + " where loper_id = "
				+ loper_id + ";";
		ResultSet res = null;
		try {
			res = setning.executeQuery(sqlsetning);
			while (res.next()) {
				Float ide = res.getFloat("id");
				idFloat.add(ide);
			}
		} finally {
			if (res != null)
				res.close();
		}
		floatidx = 0;
		int i = 0;
		int[] id = new int[idFloat.size()];
		int k = id.length;
		for (int j = 0; j < k; j++) {
			idx = idFloat.get(i);
			id[i] = (int) idx;
			i = i + 1;
		}
		return id;
}

// Sjekker hvilken post løperen er på vei til

public int nestePost(String loper_nr, int loper_id) throws Exception {
		String sqlsetn = "Select pa_vei_til from " + loper_nr
				+ " where loper_id =" + loper_id + ";";
		int post_nr = 0;
		ResultSet res = null;
		try {
			res = setning.executeQuery(sqlsetn);
			while (res.next()) {
				int post_nr_ = res.getInt("pa_vei_til");
				post_nr = post_nr_;
			}
		} finally {
			if (res != null)
				res.close();
		}
		returnpost_nr;
}

// Henter ut siste postnummer fra tabellen poster

public int antallPoster() throws Exception {
		intAntallPoster = 0;
		String sqlsetn = "Select post_nr from poster;";
		ResultSet res1 = null;
		ArrayList<Float>antallet = new ArrayList<Float>();
		try {
			res1 = setning.executeQuery(sqlsetn);
			while (res1.next()) {
				intantall = res1.getInt("post_nr");
```

```
                                        antallet.add((float) antall);
                                }
                        } finally {
                                if (res1 != null)
                                        res1.close();
                        }
                        AntallPoster = antallet.size();
                        returnAntallPoster;
}

// Lager liste for minste avstand mellom gps observasjon og nærmeste post.

public ArrayList<Float> bufferverdi(int AntallPoster, String gps_spor,
                                ArrayList<Float> lopere_, String poster, int strGpsSpor)
                                throwsException {
                        float id = 0;
                        int str = 0;
                        str = lopere_.size();
                        ArrayList<Float> avstand_2 = newArrayList<Float>();
                        for (int teller = 1; teller <= AntallPoster; teller++) {
                                int i = 0;
                                ArrayList<Float>avstandTilNestePost = newArrayList<Float>();
                                for (int str_ = 0; str_ <str; str_++) {
                                        id = lopere_.get(i);
                                String sqlsetning = "Select ST_Distance(ST_Transform(p1.posisjon,32632),
                                ST_Transform(p2.post_geom,32632)) from "
                                                                + gps_spor
                                                                + " p1, "
                                                                + poster
                                                                + " p2 where p1.loper_id = "
                                                                + id
                                                                + " and p2.post_nr = " + teller + ";";
                                        ResultSet res = null;
                                        id = id + 1;
                                        i = i + 1;
                                        try {
                                                res = setning.executeQuery(sqlsetning);
                                                while (res.next()) {
                                                        Float avstanden = res.getFloat("ST_Distance");
                                                        avstandTilNestePost.add(avstanden);
                                                }
                                        } finally {
                                                if (res != null)
                                                        res.close();
                                        }
                                }
                                Collections.sort(avstandTilNestePost);
                                float minste_avstand = avstandTilNestePost.get(0);
                                avstand_2.add(minste_avstand);
                        }
                        return avstand_2;
}

// Finner avstanden mellom GPS observasjon og neste post

public Float avstandNestePost(int post_nr, String gps_spor, String poster,
                                int id) throws Exception {
                        System.out.println("avstandNestePost");
                        float avstanden = 0;
                        String sqlsetning = "Select ST_Distance(ST_Transform(p1.posisjon,32632), ST_Transform(p2.post_geom,32632))
                        from "
                                                + gps_spor
                                                + " p1, "
                                                + poster
                                                + " p2 where p1.id = "
                                                + id
                                                + " and p2.post_nr = " + post_nr + ";";
                        ResultSet res = null;
                        try {
                                res = setning.executeQuery(sqlsetning);
                                while (res.next()) {
                                        avstanden = res.getFloat("ST_Distance");
                                }
                        } finally {
                                if (res != null)
                                        res.close();
```

```
		}
		System.out.println("avstand" + avstanden);
		return avstanden;
	}

	// Sjekker om løperen har vært innom nestePost og oppdaterer løpertabellen;

	public int besokt(ArrayList<Float>avstandTilNestePost, float buffer,
			int post_nr, int loper_id) throws Exception {
		int i = avstandTilNestePost.size() - 1;
		float avst = avstandTilNestePost.get(i);
		intbesokt_ = 0;
		if (i >= 2) {
			if (avst< buffer) {
				besokt_ = 1;
				post_nr = post_nr + 1;
				String sqlsetning = "UPDATE loper_nr SET pa_vei_til ="
						+ post_nr + " where loper_id =" + loper_id + ";";
				setning.executeUpdate(sqlsetning);
			} else {
				besokt_ = 0;
			}
		} else {
			besokt_ = 0;
		}
		returnbesokt_;
	}

	// Finner avstanden mellom 2 påfølgende GPS observasjoner

	public Float avstand(String gps_spor, int id) throws Exception {
		Float avstanden_ = null;
		int id2 = id + 1;
		String sqlsetning = "Select ST_Distance(ST_Transform(p1.posisjon,32632), ST_Transform(p2.posisjon,32632)) from
"
				+ gps_spor
				+ " p1, "
				+ gps_spor
				+ " p2 where p1.id ="
				+ id
				+ "and p2.id = " + id2 + ";";
		ResultSet res = null;
		try {
			res = setning.executeQuery(sqlsetning);
			while (res.next()) {
				avstanden_ = res.getFloat("ST_Distance");
			}
		} finally {
			if (res != null)
				res.close();
		}
		return avstanden_;
	}

	// Finner avstanden mellom 2 GPS observasjoner med X antall observasjoner
	// mellom

	public Float avstandHopp(int hopp, String gps_spor, int id)
			throws Exception {
		Float avstandHopp_ = null;
		int id2 = id + hopp;
		String sqlsetning = "Select ST_Distance(ST_Transform(p1.posisjon,32632), ST_Transform(p2.posisjon,32632)) from
"
				+ gps_spor
				+ " p1, "
				+ gps_spor
				+ " p2 where p1.id ="
				+ id
				+ "and p2.id = " + id2 + ";";
		ResultSet res = null;
		try {
			res = setning.executeQuery(sqlsetning);
			while (res.next()) {
				avstandHopp_ = res.getFloat("ST_Distance");
			}
		} finally {
```

```
                        if (res != null)
                                res.close();
                }
                returnavstandHopp_;
}

// Finner faktoren mellom avstand og faktor

public Float faktor(ArrayList<Float> avstand_, ArrayList<Float> faktor_,
                        inthopp) throws Exception {
                Float fakt = null;
                Float faktor10 = null;
                for (int teller = 0; teller < avstand_.size() - hopp + 1; teller++) {
                        Float sum = (float) 0;
                        for (int i = 0; i < hopp; i++) {
                                sum = sum + avstand_.get(i + teller);
                        }
                        fakt = faktor_.get(teller);
                        faktor10 = fakt / sum;
                }
                return faktor10;
}

public String Tid(int id, String gps_spor) throws Exception {
                String sqlsetning = "Select Tidspunkt from " + gps_spor + " where id ="
                                + id + ";";
                String tid = null;
                ResultSet res = null;
                try {
                        res = setning.executeQuery(sqlsetning);
                        while (res.next()) {
                                tid = res.getString("Tidspunkt");
                        }
                } finally {
                        if (res != null)
                                res.close();
                }
                return tid;
}

// Sender resultatene til fil

public void print(ArrayList<Float> Faktor10, String gps_spor, int hopp)
                        throws Exception {
                try {
                        FileWriteroutFile = new FileWriter(gps_spor + ".txt", true);
                        PrintWriter out = new PrintWriter(outFile);
                        out.write("hopp" + hopp + ", ");
                        for (int i = 0; i < Faktor10.size(); i++) {
                                Float sjekk = Faktor10.get(i);
                                if (sjekk != null) {
                                        String sjekk_ = Float.toString(sjekk);
                                        out.write(sjekk_ + ", ");
                                } else {
                                        out.write("1,");
                                }
                        }
                        out.write("\n");
                out.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
}

public void exprint(ArrayList<Float> Faktor10, String gps_spor, int j)
                        throws Exception {
                for (int i = 0; i < Faktor10.size(); i++) {
                        Float fakt = Faktor10.get(i);
                        if (fakt != null) {
                                Number number = new Number(0, i, fakt);
                        } else {
                                Number number = new Number(0, i, 1);
                        }
                }
}
```

```
public voidprintTid(ArrayList<Float> Faktor5, String gps_spor,
                    int loper_id, ArrayList<String> Tid, ArrayList<Float> Faktor10,
                    ArrayList<Float> Faktor20, ArrayList<Float> Faktor30,
                    ArrayList<Float>hopp_) throws Exception {
          try {
                    FileWriteroutFile = new FileWriter(gps_spor + ".txt", true);
                    PrintWriter out = new PrintWriter(outFile);
                    out.write("loper_id" + loper_id + ", ");
                    for (int i = 0; i < Faktor5.size(); i++) {
                              out.write(Tid.get(i) + ", ");
                    }
                    out.write("\n");
                    out.close();
          } catch (IOException e) {
                    e.printStackTrace();
          }
          try {
                    intl_id = loper_id - 1;
                    WritableWorkbook workbook = Workbook.createWorkbook(new File(
                                        gps_spor + l_id + ".xls"));
                    int k = 5;
                    int j = 0;
                    WritableSheet sheet_ = workbook.createSheet("løper id" + loper_id,
                                        l_id);
                    for (int i = 0; i < Faktor5.size(); i++) {
                              Float fakt = Faktor5.get(i);
                              if (fakt != null) {
                                        Number number = new Number(j, i, fakt);
                                        sheet_.addCell(number);
                              } else {
                                        Number number = new Number(j, i, 1);
                                        sheet_.addCell(number);
                              }
                    }
                    k = 10;
                    j = 1;
                    for (int i = 0; i < Faktor10.size(); i++) {
                              Float fakt = Faktor10.get(i);
                              if (fakt != null) {
                                        Number number10 = new Number(j, i, fakt);
                                        sheet_.addCell(number10);
                              } else {
                                        Number number10 = new Number(j, i, 1);
                                        sheet_.addCell(number10);
                              }
                    }
                    k = 20;
                    j = 2;
                    for (int i = 0; i < Faktor20.size(); i++) {
                              Float fakt = Faktor20.get(i);
                              if (fakt != null) {
                                        Number number20 = new Number(j, i, fakt);
                                        sheet_.addCell(number20);
                              } else {
                                        Number number20 = new Number(j, i, 1);
                                        sheet_.addCell(number20);
                              }
                    }
                    k = 30;
                    j = 3;
                    for (int i = 0; i < Faktor30.size(); i++) {
                              Float fakt = Faktor30.get(i);
                              if (fakt != null) {
                                        Number number30 = new Number(j, i, fakt);
                                        sheet_.addCell(number30);
                              } else {
                                        Number number30 = new Number(j, i, 1);
                                        sheet_.addCell(number30);
                              }
                    }
                    j = 4;
                    for (int i = 0; i <Tid.size(); i++) {
                              String tid = Tid.get(i);
                              if (tid != null) {
                                        Label tid_ = newLabel(j, i, tid);
                                        sheet_.addCell(tid_);
```

```
                                        } else {
                                                Label tid_ = newLabel(j, i, "1");
                                                sheet_.addCell(tid_);
                                        }
                                }
                                workbook.write();
                                workbook.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }

                public void printTid_(ArrayList<Float> Faktor5, String gps_spor,
                                int loper_id, ArrayList<String>Tid, ArrayList<Float>hopp_)
                                throws Exception {
                        try {
                                intl_id = loper_id - 1;
                                WritableWorkbook workbook = Workbook.createWorkbook(new File(
                                                gps_spor + l_id + "bom.xls"));
                                Float faktm = (float) 1;
                                int k = 5;
                                int l = 0;
                                int j = 0;
                                WritableSheet sheet_ = workbook.createSheet("løper id" + loper_id,
                                                l_id);
                                for (int i = 0; i < Faktor5.size(); i++) {
                                        Float fakt = Faktor5.get(i);
                                        String tid = Tid.get(i);
                                        if (fakt != null) {
                                                if (fakt< 0.8) {
                                                        if (faktm != null) {
                                                                if (faktm< 0.8) {
                                                                        Number number = new Number(j, l, fakt);
                                                                        sheet_.addCell(number);
                                                                        Label tid_ = new Label(j + 1, l, tid);
                                                                        sheet_.addCell(tid_);
                                                                        l++;
                                                                } else {
                                                                }
                                                        } else {
                                                        }
                                                } else {
                                                }
                                        } else {
                                        }
                                        faktm = fakt;
                                }
                                workbook.write();
                                workbook.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }

                // Kobler ned forbindelsen

                public voidKobleNedForbindelsen() throwsException {
                        if (setning != null)
                                setning.close();
                        if (conn != null)
                                conn.close();
                        System.out.println("avsluttet");
                }
}
```

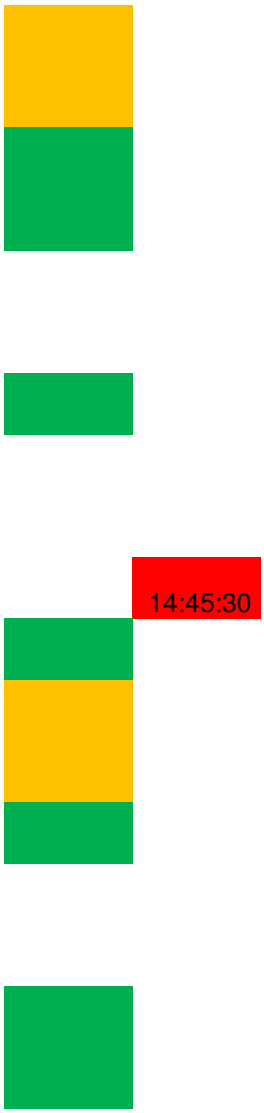
Vedlegg 2.

Legger også ved noen utskrifter av Excel arkene som programmet har skrevet ut. De første er utskrifter av det som programmet oppfattet som bom. Første kolonne er faktoren, mens den neste er tidspunktet. Fargekoden har jeg lagt til manuelt i ettertid. Grønn viser til at påvist om var korrekt. Gul betyr at påvist bom ikke var bom likevel. Rød angir at det burde ha vært oppdaget en bom ved det tidspunktet. Det siste dokumentet er en utskrift av alle faktorene for en løper, beregnet med et vindu på henholdsvis 5, 10, 20 og 30 observasjoner, samt tidspunktet.

	2010-05-15
0,581705	12:40:36
	2010-05-15
0,623737	12:40:43
	2010-05-15
0,358172	12:43:03
	2010-05-15
0,464745	12:43:10
	2010-05-15
0,741315	12:43:17
	2010-05-15
0,135661	12:50:37
	2010-05-15
0,369667	12:50:44
	2010-05-15
0,716776	12:51:26
	2010-05-15
0,643343	12:55:37
	2010-05-15
0,715403	12:55:37
	2010-05-15
0,612006	12:57:08
	2010-05-15
0,615315	12:57:50
	2010-05-15
0,777187	12:57:57
	2010-05-15
0,191175	12:58:25
	2010-05-15
0,651993	12:58:32
	2010-05-15
0,440214	13:11:00
	2010-05-15
0,40585	13:11:07
	2010-05-15
0,749181	13:11:14
	2010-05-15
0,511995	13:12:45

-Klassifisering av bevegelsesmønster hos orienteringsløpere-

	2010-05-15
0,705402	12:32:41
	2010-05-15
0,799713	12:33:58
	2010-05-15
0,78643	12:39:18
	2010-05-15
0,349795	12:40:14
	2010-05-15
0,785251	12:40:42
	2010-05-15
0,541014	12:40:56
	2010-05-15
0,555467	12:42:19
	2010-05-15
0,210468	12:42:54
	2010-05-15
0,375133	12:43:01
	2010-05-15
0,693413	12:43:08
	2010-05-15
0,700393	12:47:27
	2010-05-15
0,679476	12:50:01
	2010-05-15
0,767363	12:51:39
	2010-05-15
0,340416	12:59:13
	2010-05-15
0,073519	12:59:20
	2010-05-15
0,409318	12:59:27
	2010-05-15
0,791657	13:00:58
	2010-05-15
0,336239	13:03:11
	2010-05-15
0,423727	13:03:18
	2010-05-15
0,658123	13:03:25

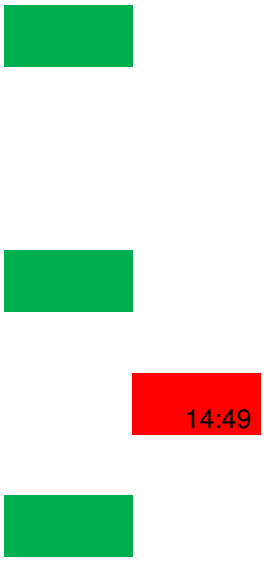


-Klassifisering av bevegelsesmønster hos orienteringsløpere-

	2010-05-15
0,78741	12:48:27
	2010-05-15
0,546434	12:49:21
	2010-05-15
0,692326	12:49:29
	2010-05-15
0,591393	12:50:27
	2010-05-15
0,539236	12:51:50
	2010-05-15
0,241908	12:51:58
	2010-05-15
0,349272	12:52:05
	2010-05-15
0,424665	13:03:50
	2010-05-15
0,621639	13:04:01
	2010-05-15
0,423387	13:05:59
	2010-05-15
0,627266	13:07:41
	2010-05-15
0,719661	13:07:49
	2010-05-15
0,785695	13:15:57
	2010-05-15
0,030591	13:16:17
	2010-05-15
0,443435	13:16:26

-Klassifisering av bevegelsesmønster hos orienteringsløpere-

	2010-05-15
0,373999	12:34:52
	2010-05-15
0,637589	12:35:27
	2010-05-15
0,692766	12:35:34
	2010-05-15
0,757936	12:35:41
	2010-05-15
0,576124	12:46:52
	2010-05-15
0,254305	12:46:59
	2010-05-15
0,183741	12:47:06
	2010-05-15
0,638467	12:47:13
	2010-05-15
0,721354	12:58:59



	2010-05-15	
0,341553	13:04:31	
	2010-05-15	
0,613993	13:04:38	
	2010-05-15	
0,17366	13:06:09	
	2010-05-15	
0,497434	13:06:16	
	2010-05-15	
0,768779	13:06:23	
	2010-05-15	
0,256784	13:09:31	
	2010-05-15	
0,732393	13:09:38	
	2010-05-15	
0,695599	13:20:28	
	2010-05-15	
0,668332	13:20:35	
	2010-05-15	
0,666254	13:34:34	

				2010-05-15
1	1	1	1	12:36:03
				2010-05-15
1	1	1	1	12:36:10
				2010-05-15
1	1	1	1	12:36:17
				2010-05-15
1	1	1	1	12:36:24
				2010-05-15
0,968989	1	1	1	12:36:31
				2010-05-15
0,967763	1	1	1	12:36:38
				2010-05-15
0,97046	1	1	1	12:36:45
				2010-05-15
0,992236	1	1	1	12:36:52
				2010-05-15
0,996394	1	1	1	12:36:59
				2010-05-15
0,983849	0,957812	1	1	12:36:59
				2010-05-15
0,977529	0,95983	1	1	12:37:06
				2010-05-15
0,970672	0,966888	1	1	12:37:13
				2010-05-15
0,962854	0,980942	1	1	12:37:20
				2010-05-15
0,966034	0,981436	1	1	12:37:27
				2010-05-15
0,966327	0,974383	1	1	12:37:34
				2010-05-15
0,928814	0,954853	1	1	12:37:41
				2010-05-15
0,947007	0,952157	1	1	12:37:48
				2010-05-15
0,939954	0,948332	1	1	12:37:55
				2010-05-15
0,964981	0,941917	1	1	12:38:02
				2010-05-15
0,94716	0,919945	0,942621	1	12:38:09
				2010-05-15
0,968986	0,922045	0,945072	1	12:38:16
				2010-05-15
0,969579	0,946968	0,950289	1	12:38:23
				2010-05-15
0,966697	0,943207	0,953455	1	12:38:30
				2010-05-15
0,943371	0,953823	0,949873	1	12:38:37
				2010-05-15
0,961171	0,949891	0,942912	1	12:38:44
				2010-05-15
0,968221	0,962184	0,940158	1	12:38:51
				2010-05-15
0,950472	0,949055	0,940108	1	12:38:58
				2010-05-15
0,950193	0,952688	0,939205	1	12:39:05
				2010-05-15
0,965728	0,952985	0,941334	1	12:39:12
				2010-05-15
0,975773	0,964737	0,936019	0,946714	12:39:19

0,984788	0,97111	0,942073	0,950504	2010-05-15 12:39:19
0,99509	0,973175	0,954823	0,955353	2010-05-15 12:39:26
0,990183	0,972718	0,955328	0,957268	2010-05-15 12:39:33
0,99049	0,978935	0,963866	0,955704	2010-05-15 12:39:40
0,972991	0,972729	0,958884	0,949587	2010-05-15 12:39:47
0,957063	0,949147	0,955321	0,93965	2010-05-15 12:39:54
0,963799	0,956162	0,952691	0,940475	2010-05-15 12:40:01
0,952867	0,954771	0,953756	0,940632	2010-05-15 12:40:08
1	1	1	1	2010-05-15 12:40:15
1	1	1	1	2010-05-15 12:40:22
1	1	1	1	2010-05-15 12:40:29
1	1	1	1	2010-05-15 12:40:36
0,623737	1	1	1	2010-05-15 12:40:43
0,888501	1	1	1	2010-05-15 12:40:50
0,84193	1	1	1	2010-05-15 12:40:57
0,904276	1	1	1	2010-05-15 12:41:04
0,962669	1	1	1	2010-05-15 12:41:11
0,967984	0,800582	1	1	2010-05-15 12:41:18
1	1	1	1	2010-05-15 12:41:25
1	1	1	1	2010-05-15 12:41:32
1	1	1	1	2010-05-15 12:41:39
1	1	1	1	2010-05-15 12:41:39
0,822826	1	1	1	2010-05-15 12:41:46
0,990472	1	1	1	2010-05-15 12:41:53
0,980945	1	1	1	2010-05-15 12:42:00
0,913465	1	1	1	2010-05-15 12:42:07
0,870889	1	1	1	2010-05-15 12:42:14
0,878832	0,771981	1	1	2010-05-15 12:42:21
0,887737	0,859327	1	1	2010-05-15 12:42:28
0,793507	0,867445	1	1	2010-05-15 12:42:35

0,829687	0,881397	1	1	2010-05-15 12:42:42
1	1	1	1	2010-05-15 12:42:49
1	1	1	1	2010-05-15 12:42:56
1	1	1	1	2010-05-15 12:43:03
1	1	1	1	2010-05-15 12:43:10
0,741315	1	1	1	2010-05-15 12:43:17
0,94594	1	1	1	2010-05-15 12:43:24
0,938113	1	1	1	2010-05-15 12:43:31
0,963535	1	1	1	2010-05-15 12:43:38
0,966213	1	1	1	2010-05-15 12:43:45
0,977953	0,827598	1	1	2010-05-15 12:43:52
0,986181	0,894916	1	1	2010-05-15 12:43:58
0,985785	0,918784	1	1	2010-05-15 12:43:58
0,994214	0,947393	1	1	2010-05-15 12:44:05
0,993021	0,974422	1	1	2010-05-15 12:44:12
1	1	1	1	2010-05-15 12:44:19
1	1	1	1	2010-05-15 12:44:26
1	1	1	1	2010-05-15 12:44:33
1	1	1	1	2010-05-15 12:44:40
0,983341	1	1	1	2010-05-15 12:44:47
0,979986	1	1	1	2010-05-15 12:44:54
0,982161	1	1	1	2010-05-15 12:45:01
0,977699	1	1	1	2010-05-15 12:45:08
0,953489	1	1	1	2010-05-15 12:45:15
0,981215	0,956279	1	1	2010-05-15 12:45:22
0,981304	0,954578	1	1	2010-05-15 12:45:29
0,988486	0,944852	1	1	2010-05-15 12:45:36
0,99207	0,953945	1	1	2010-05-15 12:45:43
0,992691	0,959759	1	1	2010-05-15 12:45:50
0,999446	0,980275	1	1	2010-05-15 12:45:57

0,997378	0,985551	1	1	2010-05-15 12:46:04
0,990778	0,989658	1	1	2010-05-15 12:46:11
0,989757	0,990368	1	1	2010-05-15 12:46:18
0,984291	0,988853	1	1	2010-05-15 12:46:18
0,986407	0,991532	0,951151	1	2010-05-15 12:46:25
0,986832	0,991794	0,957685	1	2010-05-15 12:46:32
0,993772	0,991243	0,956686	1	2010-05-15 12:46:39
0,993912	0,990795	0,966337	1	2010-05-15 12:46:46
0,993028	0,988892	0,971409	1	2010-05-15 12:46:53
0,983654	0,984806	0,982398	1	2010-05-15 12:47:00
0,984504	0,984387	0,984687	1	2010-05-15 12:47:07
0,995615	0,983714	0,986212	1	2010-05-15 12:47:14
0,99332	0,977655	0,981609	1	2010-05-15 12:47:21
0,987904	0,974762	0,975478	1	2010-05-15 12:47:28
0,987428	0,974696	0,974428	0,959305	2010-05-15 12:47:35
0,97835	0,975584	0,975165	0,963153	2010-05-15 12:47:42
0,977093	0,983996	0,974963	0,962882	2010-05-15 12:47:49
0,975416	0,984653	0,973378	0,966737	2010-05-15 12:47:56
0,982643	0,98411	0,974062	0,969022	2010-05-15 12:48:03
0,982984	0,981816	0,976532	0,975874	2010-05-15 12:48:10
0,943998	0,962093	0,961616	0,965355	2010-05-15 12:48:17
0,900771	0,938243	0,954541	0,963131	2010-05-15 12:48:24
0,912654	0,939427	0,956728	0,963266	2010-05-15 12:48:31
0,900219	0,940115	0,958919	0,963092	2010-05-15 12:48:38
0,900933	0,941834	0,959322	0,962725	2010-05-15 12:48:38
0,959419	0,942453	0,959906	0,961818	2010-05-15 12:48:45
0,984877	0,936903	0,961526	0,959363	2010-05-15 12:48:52
0,992121	0,939902	0,963288	0,960146	2010-05-15 12:48:59
0,985786	0,933419	0,95928	0,956805	2010-05-15 12:49:06
0,986651	0,935238	0,958436	0,958659	2010-05-15 12:49:13

0,97458	0,939041	0,950921	0,951642	2010-05-15 12:49:20
0,964962	0,963114	0,941965	0,943084	2010-05-15 12:49:27
0,974585	0,964547	0,933635	0,937764	2010-05-15 12:49:34
0,962986	0,954312	0,921035	0,932897	2010-05-15 12:49:41
0,990727	0,960208	0,918148	0,931844	2010-05-15 12:49:48
0,976195	0,948416	0,908572	0,922484	2010-05-15 12:49:55
0,976577	0,944434	0,902002	0,923225	2010-05-15 12:50:02
0,935644	0,922395	0,868108	0,901067	2010-05-15 12:50:09
0,909118	0,894865	0,846065	0,882335	2010-05-15 12:50:16
0,941355	0,921473	0,848634	0,878299	2010-05-15 12:50:23
0,711257	0,857366	0,83101	0,871514	2010-05-15 12:50:30
0,135661	0,591227	0,787198	0,829186	2010-05-15 12:50:37
0,369667	0,449045	0,757063	0,80747	2010-05-15 12:50:44
0,899537	0,297879	0,726949	0,785362	2010-05-15 12:50:51
0,928812	0,237352	0,709746	0,777269	2010-05-15 12:50:58
0,890493	0,360766	0,67604	0,76867	2010-05-15 12:50:58
1	1	1	1	2010-05-15 12:51:05
1	1	1	1	2010-05-15 12:51:12
1	1	1	1	2010-05-15 12:51:19
1	1	1	1	2010-05-15 12:51:26
0,923434	1	1	1	2010-05-15 12:51:33
0,960833	1	1	1	2010-05-15 12:51:40
0,964597	1	1	1	2010-05-15 12:51:47
0,97136	1	1	1	2010-05-15 12:51:54
0,96734	1	1	1	2010-05-15 12:52:01
0,95941	0,943005	1	1	2010-05-15 12:52:08
0,960398	0,959043	1	1	2010-05-15 12:52:15
0,973435	0,962075	1	1	2010-05-15 12:52:22
0,991633	0,96831	1	1	2010-05-15 12:52:29
0,991593	0,974548	1	1	2010-05-15 12:52:36

				2010-05-15
0,992869	0,97445	1	1	12:52:43
				2010-05-15
0,988088	0,970119	1	1	12:52:50
				2010-05-15
0,94228	0,950399	1	1	12:52:57
				2010-05-15
0,929013	0,954462	1	1	12:53:04
				2010-05-15
0,891986	0,913084	1	1	12:53:11
				2010-05-15
0,922793	0,904185	0,898525	1	12:53:18
				2010-05-15
0,979035	0,917321	0,896221	1	12:53:18
				2010-05-15
0,946953	0,925375	0,90267	1	12:53:25
				2010-05-15
0,91771	0,921245	0,917527	1	12:53:32
				2010-05-15
0,967736	0,911215	0,925254	1	12:53:39
				2010-05-15
0,970723	0,917955	0,92428	1	12:53:46
				2010-05-15
0,978791	0,933629	0,923394	1	12:53:53
				2010-05-15
0,978589	0,9399	0,934222	1	12:54:00
				2010-05-15
0,977712	0,939087	0,944798	1	12:54:07
				2010-05-15
0,98465	0,97643	0,943986	1	12:54:14
				2010-05-15
0,978271	0,973164	0,939911	0,928107	12:54:21
				2010-05-15
0,992323	0,983532	0,943124	0,930115	12:54:28
				2010-05-15
0,993567	0,982111	0,942317	0,931412	12:54:35
				2010-05-15
0,988421	0,976539	0,936695	0,936953	12:54:42
				2010-05-15
0,988698	0,97618	0,93384	0,941206	12:54:49
				2010-05-15
0,970224	0,968629	0,924151	0,935424	12:54:56
				2010-05-15
0,972797	0,979773	0,928372	0,934891	12:55:02
				2010-05-15
0,961707	0,979835	0,936924	0,940029	12:55:10
				2010-05-15
1	1	1	1	12:55:16
				2010-05-15
1	1	1	1	12:55:23
				2010-05-15
1	1	1	1	12:55:30
				2010-05-15
1	1	1	1	12:55:37
				2010-05-15
0,715403	1	1	1	12:55:37
				2010-05-15
0,97181	1	1	1	12:55:44
				2010-05-15
0,971411	1	1	1	12:55:51

0,981637	1	1	1	2010-05-15 12:55:58
0,982344	1	1	1	2010-05-15 12:56:05
0,991269	0,869976	1	1	2010-05-15 12:56:12
0,992363	0,974917	1	1	2010-05-15 12:56:19
0,991901	0,975803	1	1	2010-05-15 12:56:26
0,99278	0,987006	1	1	2010-05-15 12:56:33
1	1	1	1	2010-05-15 12:56:40
1	1	1	1	2010-05-15 12:56:47
1	1	1	1	2010-05-15 12:56:54
1	1	1	1	2010-05-15 12:57:01
0,612006	1	1	1	2010-05-15 12:57:08
0,635086	1	1	1	2010-05-15 12:57:15
0,744491	1	1	1	2010-05-15 12:57:22
0,9446	1	1	1	2010-05-15 12:57:29
0,869101	1	1	1	2010-05-15 12:57:36
0,721433	0,641779	1	1	2010-05-15 12:57:43
1	1	1	1	2010-05-15 12:57:50
1	1	1	1	2010-05-15 12:57:57
1	1	1	1	2010-05-15 12:57:57
1	1	1	1	2010-05-15 12:58:04
0,820737	1	1	1	2010-05-15 12:58:11
0,132324	1	1	1	2010-05-15 12:58:18
0,191175	1	1	1	2010-05-15 12:58:25
0,651993	1	1	1	2010-05-15 12:58:32
0,950273	1	1	1	2010-05-15 12:58:39
0,962154	0,548136	1	1	2010-05-15 12:58:46
0,925689	0,664245	1	1	2010-05-15 12:58:53
0,901581	0,676507	1	1	2010-05-15 12:59:00
0,879419	0,751628	1	1	2010-05-15 12:59:07
0,906854	0,825012	1	1	2010-05-15 12:59:14

0,919996	0,796503	1	1	2010-05-15 12:59:21
0,95144	0,765196	1	1	2010-05-15 12:59:28
0,955676	0,785082	1	1	2010-05-15 12:59:35
0,941741	0,852722	1	1	2010-05-15 12:59:42
0,941741	0,915169	1	1	2010-05-15 12:59:49
0,958969	0,934402	0,665212	1	2010-05-15 12:59:56
0,994469	0,932837	0,724504	1	2010-05-15 13:00:03
1	1	1	1	2010-05-15 13:00:10
1	1	1	1	2010-05-15 13:00:17
1	1	1	1	2010-05-15 13:00:17
1	1	1	1	2010-05-15 13:00:24
0,90915	1	1	1	2010-05-15 13:00:31
0,993103	1	1	1	2010-05-15 13:00:38
0,98981	1	1	1	2010-05-15 13:00:45
0,974985	1	1	1	2010-05-15 13:00:52
0,97281	1	1	1	2010-05-15 13:00:59
0,973769	0,930642	1	1	2010-05-15 13:01:06
0,994187	0,978552	1	1	2010-05-15 13:01:13
0,991239	0,980888	1	1	2010-05-15 13:01:20
0,974854	0,974009	1	1	2010-05-15 13:01:27
0,953988	0,962825	1	1	2010-05-15 13:01:34
0,969128	0,964312	1	1	2010-05-15 13:01:41
0,977663	0,962748	1	1	2010-05-15 13:01:48
0,955836	0,958767	1	1	2010-05-15 13:01:55
0,957701	0,962151	1	1	2010-05-15 13:02:02
0,967866	0,95967	1	1	2010-05-15 13:02:09
0,955355	0,961722	0,944329	1	2010-05-15 13:02:16
0,940822	0,959478	0,95717	1	2010-05-15 13:02:23
0,988232	0,961166	0,955621	1	2010-05-15 13:02:30
0,991587	0,956457	0,954317	1	2010-05-15 13:02:37

0,969946	0,953569	0,952677	1	2010-05-15 13:02:37
0,963915	0,956498	0,951711	1	2010-05-15 13:02:44
0,959573	0,947601	0,943638	1	2010-05-15 13:02:51
0,953982	0,973667	0,93914	1	2010-05-15 13:02:58
1	1	1	1	2010-05-15 13:03:05
1	1	1	1	2010-05-15 13:03:12
1	1	1	1	2010-05-15 13:03:19
1	1	1	1	2010-05-15 13:03:26
0,931377	1	1	1	2010-05-15 13:03:33
0,967055	1	1	1	2010-05-15 13:03:40
0,988229	1	1	1	2010-05-15 13:03:47
0,996551	1	1	1	2010-05-15 13:03:54
0,995926	1	1	1	2010-05-15 13:04:01
0,974195	0,94688	1	1	2010-05-15 13:04:08
0,969695	0,967932	1	1	2010-05-15 13:04:15
0,968153	0,978275	1	1	2010-05-15 13:04:22
0,949625	0,973443	1	1	2010-05-15 13:04:29
1	1	1	1	2010-05-15 13:04:36
1	1	1	1	2010-05-15 13:04:43
1	1	1	1	2010-05-15 13:04:50
1	1	1	1	2010-05-15 13:04:57
0,954637	1	1	1	2010-05-15 13:04:57
0,997618	1	1	1	2010-05-15 13:05:04
0,998227	1	1	1	2010-05-15 13:05:11
0,997979	1	1	1	2010-05-15 13:05:18
0,960332	1	1	1	2010-05-15 13:05:25
0,930624	0,920201	1	1	2010-05-15 13:05:32
0,925848	0,948977	1	1	2010-05-15 13:05:39
0,917395	0,925329	1	1	2010-05-15 13:05:46
0,983017	0,920897	1	1	2010-05-15 13:05:53

0,980238	0,917078	1	1	2010-05-15 13:06:00
0,981936	0,918905	1	1	2010-05-15 13:06:07
0,906218	0,910499	1	1	2010-05-15 13:06:14
0,93426	0,920755	1	1	2010-05-15 13:06:20
1	1	1	1	2010-05-15 13:06:27
1	1	1	1	2010-05-15 13:06:34
1	1	1	1	2010-05-15 13:06:41
1	1	1	1	2010-05-15 13:06:48
0,907843	1	1	1	2010-05-15 13:06:55
0,941365	1	1	1	2010-05-15 13:07:02
0,99144	1	1	1	2010-05-15 13:07:09
0,989574	1	1	1	2010-05-15 13:07:16
0,96421	1	1	1	2010-05-15 13:07:16
0,96305	0,93338	1	1	2010-05-15 13:07:23
1	1	1	1	2010-05-15 13:07:30
1	1	1	1	2010-05-15 13:07:37
1	1	1	1	2010-05-15 13:07:44
1	1	1	1	2010-05-15 13:07:51
0,925609	1	1	1	2010-05-15 13:07:58
0,917981	1	1	1	2010-05-15 13:08:05
0,907164	1	1	1	2010-05-15 13:08:12
0,992415	1	1	1	2010-05-15 13:08:19
0,988524	1	1	1	2010-05-15 13:08:26
0,980017	0,924505	1	1	2010-05-15 13:08:33
0,966763	0,916915	1	1	2010-05-15 13:08:40
0,961579	0,912897	1	1	2010-05-15 13:08:47
0,946117	0,972125	1	1	2010-05-15 13:08:54
1	1	1	1	2010-05-15 13:09:01
1	1	1	1	2010-05-15 13:09:08
1	1	1	1	2010-05-15 13:09:15

1	1	1	1	2010-05-15 13:09:22
0,973255	1	1	1	2010-05-15 13:09:29
0,977822	1	1	1	2010-05-15 13:09:36
0,986949	1	1	1	2010-05-15 13:09:36
0,991783	1	1	1	2010-05-15 13:09:43
0,992168	1	1	1	2010-05-15 13:09:50
0,98988	0,964044	1	1	2010-05-15 13:09:57
0,984622	0,974363	1	1	2010-05-15 13:10:04
0,955867	0,966767	1	1	2010-05-15 13:10:11
0,960011	0,971966	1	1	2010-05-15 13:10:18
0,959989	0,966416	1	1	2010-05-15 13:10:25
0,967505	0,946519	1	1	2010-05-15 13:10:32
1	1	1	1	2010-05-15 13:10:39
1	1	1	1	2010-05-15 13:10:46
1	1	1	1	2010-05-15 13:10:53
1	1	1	1	2010-05-15 13:11:00
0,40585	1	1	1	2010-05-15 13:11:07
0,749181	1	1	1	2010-05-15 13:11:14
0,974596	1	1	1	2010-05-15 13:11:21
0,985086	1	1	1	2010-05-15 13:11:28
0,993491	1	1	1	2010-05-15 13:11:35
0,996824	0,725346	1	1	2010-05-15 13:11:42
0,982944	0,869723	1	1	2010-05-15 13:11:49
0,98016	0,977209	1	1	2010-05-15 13:11:56
0,972882	0,97641	1	1	2010-05-15 13:11:56
0,989008	0,98446	1	1	2010-05-15 13:12:03
0,984881	0,982796	1	1	2010-05-15 13:12:10
0,986126	0,979812	1	1	2010-05-15 13:12:17
0,986327	0,980423	1	1	2010-05-15 13:12:24
1	1	1	1	2010-05-15 13:12:31

				2010-05-15
1	1	1	1	13:12:38
				2010-05-15
1	1	1	1	13:12:45